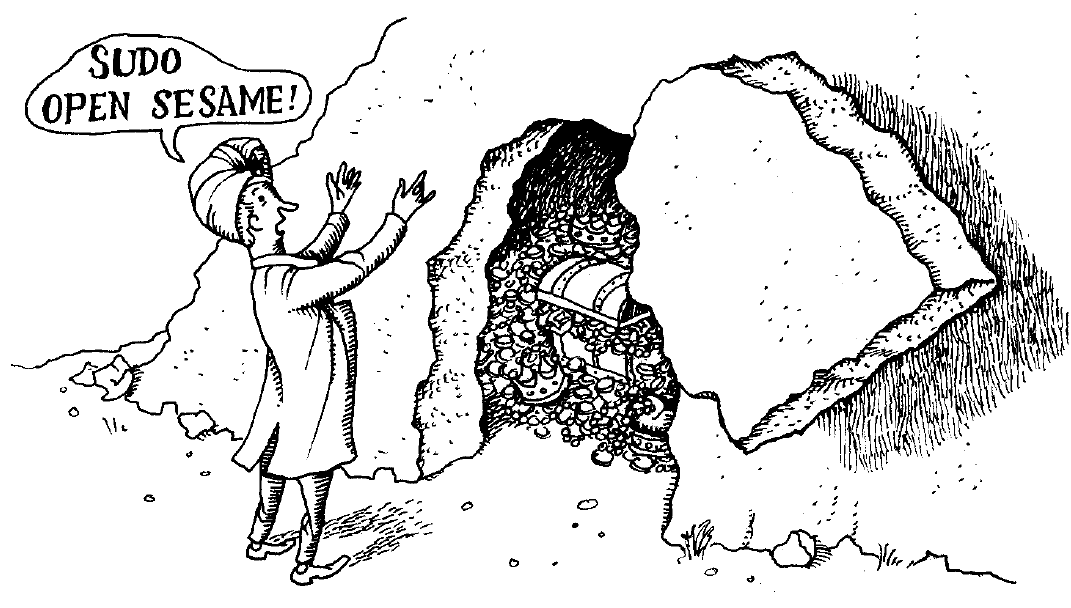
***3 Управление доступом и сила привилегий***



Эта глава посвящена "контролю доступа", а не "безопасности", под которой мы подразумеваем, что она фокусируется на механических деталях того, как ядро и его делегаты принимают решения, связанные с безопасностью. В Главе 27 *"Безопасность"* рассматривается более общий вопрос о том, как настроить систему или сеть, чтобы свести к минимуму вероятность нежелательного доступа злоумышленников.

Контроль доступа - это область активных исследований, он уже давно является одной из главных проблем разработки операционной системы. За последнее десятилетие в UNIX и Linux произошел "Кембрийский взрыв" новых опций в этой области. Основным драйвером этого всплеска было появление API ядра, который позволяет сторонним модулям дополнять или заменять традиционную систему контроля доступа UNIX. Этот модульный подход создает множество новых возможностей; контроль доступа теперь так же открыт для изменений и экспериментов, как и любой другой аспект UNIX.

Тем не менее, традиционная система остается стандартом UNIX и Linux, и она подходит для большинства установок. Даже для администраторов, которые хотят выйти на новые рубежи, необходимо тщательно изучить основы.

**3.1 Стандартное управление доступом UNIX**

Стандартная модель контроля доступа UNIX в течение десятилетий практически не изменилась. С некоторыми усовершенствованиями она по-прежнему является стандартной для дистрибутивов ОС общего назначения. Схема придерживается нескольких основных правил:

- Решения по управлению доступом зависят от того, какой пользователь пытается выполнить операцию или, в некоторых случаях, от членства этого пользователя в группе UNIX.

- У объектов (например, файлов и процессов) есть владельцы. Владельцы имеют широкий (но не обязательно неограниченный) контроль над своими объектами.

- Вы являетесь владельцем тех объектов, которые создаете.

- Специальная учетная запись, называемая "root", может действовать как владелец любого объекта в системе.

- Только "root" может выполнять определенные специальные административные операции1.

*1. Имейте в виду, что мы здесь описываем оригинальный дизайн системы контроля доступа. В наши дни не все эти утверждения остаются истинными в буквальном смысле. Например, процесс Linux, обладающий соответствующими возможностями (см. стр. 82), теперь может выполнять некоторые операции, которые ранее выполнялись только из-под пользователя root.*

Некоторые системные вызовы (например, **settimeofday**) могут выполняться только пользователем root (при выполнении таких системных вызовов просто проверяется "личность" текущего пользователя, и, если он окажется не суперпользователем, операция отвергается). Другие системные вызовы (например, **kill**) реализуют различные вычисления, которые требуют как соответствия владельцу, так и обеспечения специальных условий для суперпользователя. Наконец, файловые системы имеют свои собственные системы управления доступом, которые они реализуют совместно с уровнем VFS ядра. Они, как правило, более сложные, чем элементы управления доступом, найденные в другом месте ядра. Например, файловые системы гораздо чаще используют группы UNIX для контроля доступа.

*Дополнительную информацию о файлах устройств см. на стр. 130.*

Усложняет эту картину то, что ядро и файловая система тесно переплетены. Например, вы управляете и взаимодействуете с большинством устройств через файлы, которые представляют их в каталоге **/dev**. Поскольку файлы устройств являются объектами файловой системы, они подчиняются семантике управления доступом к файловой системе. Ядро использует этот факт в качестве основной формы контроля доступа для устройств.

**Управление доступом в файловой системе**

Каждый файл в стандартной модели имеет владельца и группу, иногда называемой "группой владельца". Владелец может устанавливать разрешения для файла. В частности, владелец может настолько их ограничить, что никто другой не сможет получить к нему доступ. Мы глубже рассматриваем разрешения файлов в Главе 5 *"Файловая система"* (см. стр. 132).

*См. стр. 254 для получения дополнительной информации о группах.*

Хотя владельцем файла всегда является один человек, многие люди могут быть владельцами файлов, если они все являются частью одной группы. Группы традиционно определяются в файле **/etc/group**, но в настоящее время информация о группах все чаще хранится в сетевых базах данных, таких как LDAP; дополнительную информацию см. в Главе 17 "Single Sign-On".

Владелец файла определяет, что могут сделать с ним члены группы-владельца. Эта схема позволяет делиться файлами между членами одного и того же проекта.

Вы можете определить владельца файла с помощью **ls -l**:

**$ ls -l ~garth/todo**

-rw-r----- 1 garth staff 1259 May 29 19:55 /Users/garth/todo

Этот файл принадлежит пользователю garth и группе staff. Буквы и дефисы в первом столбце означают права доступа к файлу; см. стр. 134 для получения подробной информации о том, как декодировать эту информацию. В этом случае коды означают, что garth имеет права на чтение и запись файла, а члены группы staff имеют только права на чтение.

*См. Главу 8 для получения дополнительной информации о файлах* ***passwd*** *и* ***group****.*

Ядро и файловая система отслеживают владельцев и группы как числа, а не как текстовые имена. В наиболее общем случае идентификаторы пользователей (uid) сопоставляются с именами пользователей в файле **/etc/passwd**, а идентификационные номера групп (GID) сопоставляются с именами групп в **/etc/group**. См. Главу 17 "Single Sign-On" для получения информации о более сложных вариантах.

Текстовые имена, соответствующие UID и GID, определяются только для удобства пользователей. Когда команды, такие как **ls**, должны отображать информацию о владельце в удобочитаемом формате, они должны искать каждое имя в соответствующем файле или базе данных.

**Владение процессом**

Владелец процесса может отправлять сигналы процессу (см. стр. 94) и также может уменьшить (ухудшить) его приоритет. Процессы на самом деле имеют несколько идентификаторов: реальный, текущий и сохраненный UID; реальный, текущий и сохраненный GID; и в Linux - "UID файловой системы", который используется только для определения разрешений доступа к файлам. Вообще говоря, реальные идентификаторы используются для учета (в настоящее время используются все меньше), а текущие - для определения прав доступа. Реальные и текущие идентификаторы, как правило, совпадают.

Сохраненные UID и GID - это парковочные места для идентификаторов, которые в настоящее время не используются, но которые остаются доступными процессу для вызова. Сохраненные идентификаторы позволяют программе повторно входить и выходить из привилегированного режима работы; эта предосторожность уменьшает риск непреднамеренного неправильного поведения.

*Подробнее о NFS см. в Главе 21.*

UID файловой системы обычно объясняется как особенность реализации NFS, сетевой файловой системы. Обычно он совпадает с текущим UID.

**Учетная запись суперпользователя**

Учетная запись root — это всемогущий пользователь-администратор UNIX. Он также известен как суперпользователь, хотя его фактическим именем является "root".

Определяющей характеристикой учетной записи суперпользователя является значение UID, равное нулю. Ничто не запрещает менять имя этой учетной записи или создавать другую запись с нулевым идентификатором, но такие действия ни к чему хорошему не приведут2. Их следствием будет возникновение новых брешей в системе защиты. Они также создают путаницу, когда другим людям приходится иметь дело со странным способом настройки вашей системы.

*2. Дженин Таунсенд (Jennine Townsend), один из наших стойких технических рецензентов, прокомментировал: «Такие плохие идеи, что я боюсь, что даже упоминание о них может кого-то вдохновить!»*

Традиционный UNIX позволяет суперпользователю (т.е. всякому процессу, текущий UID которого равен нулю) выполнять над файлом или процессом любую допустимую операцию3.

*3. "Допустимую" — это ключевое слово. Некоторые операции (например, запуск файла, для которого не задан бит выполнения "execute") запрещены даже суперпользователю.*

Некоторые примеры ограниченных операций:

- создание файлов устройств;

- настройка системных часов;

- увеличение лимитов использования ресурсов и повышение приоритетов процессов;

- задание имени компьютера;

- конфигурирование сетевых интерфейсов;

- открытие привилегированных сетевых портов (номера которых меньше 1024);

- выключение системы.

Примером полномочий суперпользователя является способность процесса, принадлежащего root, изменять свои UID и GID. Примером таких процессов может служить программа **login** и ее GUI-эквиваленты, которые запрашивают ваш пароль при входе в систему; они изначально запускаются с правами root. Если введенный вами пароль и имя пользователя являются правильными, программа входа в систему изменяет свой UID и GID на ваш UID и GID и запускает shell или GUI. Как только процесс суперпользователя изменил свое владение, чтобы стать обычным пользовательским процессом, он не сможет восстановить прежнее привилегированное состояние.

**Использование setuid и setgid**

Традиционная система управления доступом в UNIX дополнена системой смены полномочий, которая реализуется ядром в сотрудничестве с файловой системой. Эта система позволяет выполнять специально подготовленные файлы с использованием привилегий более высокого уровня (обычно это привилегии суперпользователя). Этот механизм позволяет разработчикам и администраторам создавать условия для непривилегированных пользователей, при которых они могут выполнять привилегированные операции.

Когда ядро запускает исполняемый файл с установленными битами разрешения "setuid" или "setgid" он изменяет текущий UID или GID создаваемого процесса на UID или GID файла, из которого запускается программа, а не UID и GID пользователя, который запустил программу. Таким образом, привилегии пользователя распространяются только на выполнение этой конкретной команды.

Например, пользователи должны иметь возможность изменять свои пароли. Но поскольку пароли хранятся в защищенном файле **/etc/master.passwd** или **/etc/shadow**, пользователям нужно использовать команду **passwd** с полномочиями setuid, чтобы "усилить" свои права доступа. Команда **passwd** проверяет, кто ее выполняет, и, в зависимости от результата, настраивает свое поведение соответствующим образом, поэтому обычные пользователи могут менять только собственные пароли, а суперпользователь - любые.

Программы, которые используют setuid, особенно те, которые запускают setuid для root, подвержены проблемам безопасности. Команды setuid, распространяемые с системой, теоретически защищены; однако дыры безопасности обнаруживались в прошлом и, несомненно, будут обнаружены в будущем.

Самый верный способ минимизировать количество проблем setuid — это минимизировать количество программ, использующих setuid. Подумайте дважды, прежде чем устанавливать программное обеспечение, которое должно запускать setuid, и избегайте использования setuid в сделанном вами программном обеспечении. Никогда не используйте setuid для программ, которые не были явно написаны с учетом выполнения setuid.

*См. стр. 767 для получения дополнительной информации о возможностях монтирования файловой системы.*

Вы можете отключить выполнение setuid и setgid в отдельных файловых системах, указав параметр **nosuid** для команды **mount**. Рекомендуется использовать этот параметр в файловых системах, которые содержат домашние каталоги пользователей или монтируются из менее надежных административных доменов.

**3.2 Управление учетной записью root**

Доступ к root необходим для администрирования системы, это также точка опоры для обеспечения безопасности системы. Правильное использование учетной записи root является критически важным навыком.

**Вход в root аккаунт**

Поскольку root является таким же аккаунтом, как и аккаунты остальных пользователей, многие системы дают вам возможность войти в систему непосредственно под учетной записью root. Однако это очень плохая идея, поэтому в Ubuntu такая возможность отключена изначально.

Во-первых, не будет сделано никаких записей о том, какие операции выполнял суперпользователь. Согласитесь, не очень приятно обнаружить, что вчера ночью в 3:00 вы что-то поменяли, но никак не можете вспомнить, что именно. Еще хуже, если такой доступ был несанкционированным и необходимо выяснить, какой ущерб системе нанес нарушитель. Во-вторых, сценарий регистрации суперпользователя не предполагает сбора дополнительной идентифицирующей информации. Когда под именем root в систему могут входить несколько пользователей, не существует способа определить, кто именно и когда это делал.

Вследствие упомянутых причин в большинстве систем вход в учетную запись root может быть запрещена в терминалах, в оконных системах и по сети, т.е. везде, кроме системной консоли. Мы рекомендуем следовать этой схеме. Смотрите раздел "*PAM: cooking spray or authentication wonder?*" начиная со страницы 590, чтобы узнать, как реализовать эту политику в вашей конкретной системе.

Если у root есть пароль (т.е. учетная запись root не отключена, см. стр. 78), этот пароль должен быть высокого качества. См. стр. 992 для получения дополнительной информации относительно выбора пароля.

**Команда su: замена идентификатора пользователя**

Лучший способ доступа к учетной записи root - использовать команду **su**. Если эта команда вызывается без аргументов, то она запрашивает пароль root, а затем запускает оболочку root. Привилегии root остаются в силе до тех пор, пока вы не завершите работу с ним, набрав <Control-D> или команду **exit**. **su** не записывает команды, выполняемые с правами администратора, но создает запись в журнале, с указанием кто и когда зашел под учетной записью root.

Команда **su** способна также подставлять вместо имени root имена других пользователей. Иногда единственный способ помочь пользователю в решении проблемы - войти с помощью команды **su** в его аккаунт, чтобы воспроизвести его окружение, в котором возникает проблема.

Зная чей-либо пароль, можно непосредственно зарегистрироваться в системе под его аккаунтом, введя команду **su -** *имя\_пользователя*. В ответ будет выдан запрос на ввод пароля. Дефис (-) в команде **su** позволяет запускать оболочку в режиме входа в систему.

Точная реализация этого режима зависит от конкретного интерпретатора команд, но обычно в этом режиме меняются количество и имена файлов запуска, считываемых интерпретатором. Например, в режиме регистрации интерпретатор **bash** считывает файл **~/.bash\_profile**, а вне этого режима - файл **~/.bashrc**. При диагностике проблем других пользователей режим регистрации позволяет максимально точно воспроизвести среду, в которой он работал.

В одних системах пароль root позволяет заходить с помощью команд **su** или **login** в любой аккаунт. В других нужно сначала стать суперпользователем, воспользовавшись командой **su**, а затем с помощью этой же команды перейти в другую учетную запись без ввода пароля.

Рекомендуем взять за правило при вводе команды указывать полный путь к ней (например **/bin/su** или **/usr/bin/su**) вместо того, чтобы полагаться на оболочку, которая найдет команду для вас.

Эта предосторожность даст вам некоторую защиту от произвольных программ с тем же названием **su**, которые, возможно, были прописаны в переменной path с целью сбора паролей4.

*4. По аналогичной причине рекомендуем не включать запись* "*.*" *(текущий каталог) в переменную оболочки path (которую вы можете увидеть с помощью* ***echo $PATH****). Несмотря на очевидное удобство, включение* "*.*" *приводит к тому, что можно непреднамеренно запустить* "*специальную*" *версию какой-нибудь системной команды, которую злоумышленник оставил в качестве приманки. Пользователю root следует быть бдительным вдвойне.*

Во многих системах, чтобы использовать команду **su**, необходимо быть членом группы "wheel".

Мы считаем, что **su** в значительной степени была заменена утилитой **sudo** (см. следующий раздел). Саму же команду **su** лучше всего оставить для крайних (аварийных) случаев. Она также полезна при фиксинге в ситуациях, в которых **sudo** был сломан или неправильно сконфигурирован.

**Утилита sudo: ограниченный вариант команды su**

Без организации управления доступом на основе ролей (RBAC) или такой системы, как SELinux, трудно предоставить кому-то право выполнять конкретную административную операцию (например, создавать резервные копии), не предоставив возможность свободно работать в системе. Если же учетная запись root используется несколькими администраторами, то вы будете иметь смутное представление о том, кто ею пользуется и что он при этом делает.

Наиболее широко используемым решением этих проблем является программа под названием **sudo**, которая в настоящее время поддерживается Тоддом Миллером (Todd Miller). Она работает на всех приведенных нами системах, а ее исходный код можно найти на веб-сайте sudo.ws. Мы рекомендуем использовать её в качестве основного способа входа в учётную запись root.

Утилита **sudo** в качестве аргумента принимает командную строку, которая выполняется от имени пользователя root (или другой ограниченной учетной записи). Утилита обращается к файлу **/etc/sudoers** (**/usr/local/etc/sudoers** в FreeBSD), где содержится список пользователей, имеющих разрешение на выполнение **sudo**, и перечень команд, которые они могут выполнять на каждом хосте. Если запрашиваемая команда разрешена, **sudo** запрашивает собственный пароль пользователя и выполняет команду.

Дополнительно утилита **sudo** позволяет, не вводя пароль, выполнять другие команды, но только до тех пор, пока не наступит пятиминутный период бездействия (его продолжительность можно изменять). Этот тайм-аут служит скромной защитой от пользователей с привилегиями sudo, которые оставляют терминалы без присмотра.

*Подробнее о syslog читайте в Главе 10.*

**sudo** ведет журнал, где регистрируются выполненные команды, компьютеры, на которых они выполнялись, и вызвавшие их пользователи, а также каталоги, из которых запускались команды, и время их вызова. Эта информация может быть зарегистрирована **syslog** или сохраняться в любом другом файле по усмотрению пользователя. Мы рекомендуем использовать **syslog** для пересылки записей журнала на безопасный центральный узел.

Запись в журнале, содержащая данные о пользователе "randy", который выполнил команду **sudo /bin/cat /etc/sudoers**, может выглядеть следующим образом.

Dec 7 10:57:19 tigger sudo: randy: TTY=ttypO ; PWD=/tigger/users/randy;

USER=root ; COMMAND=/bin/cat /etc/sudoers

*Пример конфигурации*

Файл **sudoers** разработан таким образом, что одна версия может использоваться сразу на нескольких разных хостах. Вот типичный пример:

# Define aliases for machines in CS & Physics departments

Host\_Alias CS = tigger, anchor, piper, moet, sigi

Host\_Alias PHYSICS = eprince, pprince, Icarus

# Define collections of commands

Cmnd\_Alias DUMP = /sbin/dump, /sbin/restore

Cmnd\_Alias WATCHDOG = /usr/local/bin/watchdog

Cmnd\_Alias SHELLS = /bin/sh, /bin/dash, /bin/bash

# Permissions

mark, ed PHYSICS = ALL

herb CS = /usr/sbin/tcpdump : PHYSICS = (operator) DUMP

lynda ALL = (ALL) ALL, !SHELLS

%wheel ALL, !PHYSICS = NOPASSWD: WATCHDOG

Первые два набора строк определяют группы хостов и команды, на которые имеются ссылки в спецификациях прав доступа позже в файле. Списки могут быть включены явно в спецификации, но псевдонимы облегчают чтение и понимание файла **sudoers**; они также упрощают внесение изменений в файл в будущем. Также возможно определить псевдонимы для групп пользователей и для групп пользователей, для которых могут выполняться команды.

В каждую спецификацию прав доступа включается информация о следующем:

- пользователях, к которым относится запись;

- компьютерах, на которых следует учитывать записи;

- командах, которые могут выполняться указанными пользователями;

- пользователях, от имени которых могут выполняться команды.

Первая строка спецификаций применяется к пользователям "mark" и "ed", которые регистрируются в системе на компьютерах группы "PHYSICS" (eprince, pprince и icarus). Встроенный псевдоним "ALL" разрешает им вводить любые команды. Поскольку дополнительный список пользователей в скобках не указан, утилита **sudo** будет выполнять команды только от имени суперпользователя.

Вторая запись позволяет пользователю "herb" запускать утилиту **tcpdump** на компьютерах группы "CS", а также выполнять **dump-**команды на компьютерах группы "PHYSICS". Однако, вторая группа команд может запускаться только пользователем "operator", а не "root". Реальная команда, которую пришлось бы ввести пользователю "herb", выглядит примерно так:

ubuntu$ **sudo -u operator /usr/sbin/dump 0u /dev/sda1**

Пользователь "lynda" имеет право выполнять команды от имени любого пользователя на любом компьютере, но она не может запускать несколько оболочек. Означает ли это, что "lynda" не может получить доступ к оболочке root? Конечно же нет:

ubuntu$ **cp -p /bin/sh /tmp/sh**

ubuntu$ **sudo /tmp/sh**

Вообще говоря, попытка разрешить "все команды, кроме..." обречена на провал, по крайней мере, с технической точки зрения. Тем не менее, может оказаться целесообразным настроить файл **sudoers** таким образом, что это послужит хотя бы напоминанием о том, что вызывать интерпретатор команд в режиме суперпользователя не рекомендуется.

В последней строке пользователям группы "wheel" разрешается выполнять локальную команду **watchdog** от имени суперпользователя на всех компьютерах, за исключением eprince, pprince и icarus. Более того, от пользователей не требуется вводить пароль.

Обратите внимание на то, что команды в файле **/etc/sudoers** приводятся с указанием полного имени, чтобы пользователи не могли выполнять свои собственные программы и сценарии от имени суперпользователя. Разрешается также указывать допустимые аргументы команд (в приведенных выше примерах это не показано).

Для ручного изменения содержимого файла **/etc/sudoers** предназначена специальная команда **visudo**, которая проверяет, не редактируется ли файл кем-то посторонним, затем открывает его в редакторе (**vi** или любой редактор, указанный вами в переменной среды EDITOR), а затем проверяет синтаксис отредактированного файла перед его установкой. Последний этап особенно важен, поскольку ошибка в файле **/etc/sudoers** может не позволить повторно вызвать утилиту **sudo** для исправления файла.

Использование **sudo** имеет следующие преимущества.

- Благодаря регистрации команд значительно повышается степень административного контроля над системой.

- Пользователи могут выполнять специальные задачи, не имея неограниченных привилегий.

- Настоящий пароль суперпользователя могут знать всего один-два человека5.

- Использование **sudo** быстрее, чем выполнение **su** или вход в систему под пользователем root.

- Права пользователя могут быть ограничены, без изменения пароля суперпользователя.

- Ведется список всех пользователей с правами пользователя root.

- Меньше вероятность того, что оболочка root будет оставлена без присмотра.

- Управлять доступом ко всей сети можно с помощью одного файла.

*5. Или даже никто, если вы используете один из проверенных вариантов системы хранения паролей.*

*Дополнительную информацию о взломе паролей можно найти на стр. 1000*

**sudo** имеет и недостатки. Самый большой из них заключается в том, что любая брешь в системе защиты того или иного привилегированного пользователя эквивалентна нарушению безопасности самой учетной записи root. Противостоять этому нелегко. Можно лишь предупредить тех, кто имеет право выполнять утилиту sudo, о необходимости максимально защищать свои учетные записи. Можно также регулярно запускать какой-нибудь программный взломщик паролей, проверяя "стойкость" паролей привилегированных пользователей.

*Советы по выбору пароля см. со стр. 992.*

Другой недостаток - возможность обмануть утилиту **sudo** с помощью таких уловок, как временный выход в интерпретатор команд из разрешенной программы либо выполнение команд **sudo sh** или **sudo su**, если они допустимы (эти команды отображаются в журналах, поэтому вы, по крайней мере, узнаете, что они были запущены).

***sudo*** *и современный контроль доступа*

Для многих встроенных систем управления доступом на основе ролей (RBAC) **sudo** (как способ разделения привилегий суперпользователя) является лучшим инструментом. И вот почему:

- Вы можете точно определить, как именно будут разделены привилегии. Ваш вариант разделения может быть более грубым или тонким, чем привилегии, определенные для вас с помощью встроенной системы RBAC.

- Простота конфигурации - общепризнанный факт (в основном, это касается установки, обслуживания и понимания происходящего).

- Утилита **sudo** выполняется во всех системах UNIX и Linux. Вам не нужно беспокоиться об использовании различных систем на разных платформах.

- Вы можете совместно использовать единственный на всю сеть файл конфигурации.

- Вы получаете бесплатную возможность высококачественного процесса регистрации в системе.

Основной недостаток sudo-ориентированного управления доступом состоит в том, что потенциальная плоскость атаки расширяется, включая учетные записи всех администраторов системы, если будет нарушена защита учетной записи суперпользователя.

**sudo** хорошо работает как инструмент для благих намерений администраторов, которым необходим общий доступ к привилегиям root. Он также отлично подходит для того, чтобы не администраторы могли выполнять некоторые специфические операции. Несмотря на синтаксис конфигурации, который предполагает иное, это, к сожалению, небезопасный способ определить ограниченные области автономии или поместить определенные операции за их пределы.

Даже не пытайтесь выполнить эти настройки. Если вам нужна эта функциональность, вам лучше включить одну из систем управления доступом, описанных на стр. 83.

*Стандартная установка*

На протяжении многих лет система конфигурации **sudo** накопила много функций. Она также расширилась, чтобы учитывать множество необычных ситуаций и крайних случаев. В результате текущая документация передает впечатление сложности, которая не всегда является оправданной.

Поскольку важно, чтобы **sudo** был надежным и безопасным, естественно задаться вопросом, можете ли вы подвергать свои системы дополнительному риску, если вы не используете расширенные функции **sudo** и не устанавливаете правильные значения для всех параметров. Ответ - нет. 90% файлов **sudoers** выглядят примерно так:

User\_Alias ADMINS = alice, bob, charles

ADMINS ALL = (ALL) ALL

Это вполне приемлемая конфигурация, и во многих случаях нет необходимости ее усложнять. Мы упоминали несколько дополнительных функций, с которыми вы можете ознакомиться в следующих разделах, но все они являются инструментами для решения проблем, в конкретных ситуациях. Больше ничего не требуется для обеспечения общей надежности.

*Управление окружением*

Многие команды проверяют значения переменных среды и изменяют их поведение в зависимости от того, что они находят. В случае, если команды выполняются с правами root, этот механизм может быть как полезным, так и потенциальной точкой для атаки.

Например, несколько команд запускают программу, указанную в переменной среды EDITOR, для вызова текстового редактора. Если эта переменная указывает на вредоносную программу хакера вместо редактора, вполне вероятно, что в конечном итоге вы запустите эту программу с правами root6.

*6. Чтобы было понятнее - сценарий в этом случае заключается в том, что ваша учетная запись была скомпрометирована, но злоумышленник не знает ваш фактический пароль и поэтому не может напрямую запускать* ***sudo****. К сожалению, это обычная ситуация - все, что требуется, - это окно терминала, оставленное на мгновение без присмотра.*

Чтобы минимизировать этот риск, поведение **sudo** по умолчанию заключается в передаче только минимальной, "дезинфицированной" среды для команд, которые она запускает. Если вашей системе нужны дополнительные переменные окружения, которые нужно передать, вы можете их перечислить, добавив их в список env\_keep в файле **sudoers**. Например, записи:

Defaults env\_keep += "SSH\_AUTH\_SOCK"

Defaults env\_keep += "DISPLAY XAUTHORIZATION XAUTHORITY"

сохраняют несколько переменных среды, используемых X Windows и пересылают SSH ключи.

Можно настроить разные списки env\_keep для разных пользователей или групп, но конфигурация в таком случае быстро усложняется. Мы предлагаем придерживаться единого универсального списка и быть относительно консервативным с исключениями, которые вы прописываете в файле **sudoers**.

Если вам нужно сохранить переменную среды, которая не указана в файле **sudoers**, вы можете установить ее явно в командной строке с помощью **sudo**. Например, команда

$ **sudo EDITOR=emacs vipw**

редактирует файл системных паролей с помощью **emacs**. Эта функция имеет некоторые потенциальные ограничения, но они не влияют на пользователей, которые могут запускать ВСЕ команды.

***sudo*** *без паролей*

Неприятно наблюдать как часто **sudo** настроен на выполнение команды от имени root без необходимости вводить пароль. Для справки, эта конфигурация выполняется с помощью ключевого слова NOPASSWD в файле **sudoers**. Например:

ansible ALL = (ALL) NOPASSWD: ALL *# Не делайте так*

*См. Главу 23 для получения дополнительной информации об Ansible.*

Иногда это делается из-за лени, но, как правило, основная потребность заключается в том, чтобы разрешить некоторый тип автоматического выполнения **sudo**. Наиболее распространенные случаи - удаленная конфигурация через систему, такую как Ansible, или при запуске команд из cron.

Излишне говорить, что эта конфигурация опасна, поэтому избегайте ее, если можете. По крайней мере, ограничьте выполнение беспарольного доступа к определенному набору команд, если сможете.

*См. стр. 591 для получения дополнительной информации о конфигурации PAM.*

Другим вариантом, который хорошо работает в контексте удаленного выполнения, является замена введенных вручную паролей на аутентификацию с помощью ssh-агентов и обмен SSH-ключами. Вы можете настроить этот метод аутентификации через PAM на сервере, где будет выполняться **sudo**.

Большинство систем не содержат модуль PAM, который по умолчанию использует SSH аутентификацию, но его легко найти. Ищите пакет pam\_ssh\_agent\_auth.

Пересылка SSH ключей имеет собственный набор проблем безопасности, но это, безусловно, лучше чем полное отсутствие аутентификации.

*Первенство*

Данный вызов **sudo** потенциально может быть адресован нескольким записям в файле **sudoers**. Например, рассмотрим следующую конфигурацию:

User\_Alias ADMINS = alice, bob, charles

User\_Alias MYSQL\_ADMINS = alice, bob

%wheel ALL = (ALL) ALL

MYSQL\_ADMINS ALL = (mysql) NOPASSWD: ALL

ADMINS ALL = (ALL) NOPASSWD: /usr/sbin/logrotate

Здесь администраторы могут выполнять команду **logrotate** от имени любого пользователя без указания пароля. Администраторы MySQL могут запускать любую команду как пользователь mysql без ввода пароля. Любой человек в группе wheel может выполнить любую команду под любым UID, но сначала должен пройти аутентификацию с помощью пароля.

Если пользователь alice находится в группе wheel, она потенциально покрывается каждой из последних трех строк. Откуда вы знаете, какой из них будет определять поведение **sudo**?

Правило заключается в том, что sudo всегда подчиняется последней соответствующей записи, причем соответствие определяется всеми 4-мя записями пользователя, хоста, целевого пользователя и команды. Каждый из этих элементов должен соответствовать строке конфигурации, иначе строка просто игнорируется.

Поэтому исключения NOPASSWD должны соответствовать их более общим аналогам, как показано выше. Если бы порядок последних трех строк был изменен, то бедной alice пришлось бы вводить пароль независимо от того, какую команду **sudo** она пыталась запустить.

***sudo*** *без терминала управления*

Помимо того, что возникает проблема аутентификации без пароля, автоматическое выполнение **sudo** (например, из **cron**) часто происходит без обычного терминала управления. В этом нет ничего плохого, но это странная ситуация, которую **sudo** может проверить и отклонить, если опция requiretty включена в файле **sudoers**.

Этот параметр не используется по умолчанию с точки зрения **sudo**, но некоторые дистрибутивы ОС включают его в их файлы **sudoers** по умолчанию, поэтому их стоит проверить и удалить. Найдите строку в виде:

Defaults requiretty

и инвертируйте его значение:

Defaults !requiretty

Опция requiretty предлагает небольшую символическую защиту от определенных сценариев атаки. Тем не менее, её легко обойти и, таким образом, она даёт мало реальных преимуществ в плане безопасности. По нашему мнению, опция requiretty должна быть отключена, как само собой разумеющееся, потому что она часто является источником проблем.

*Конфигурация* ***sudo*** *на уровне сайта*

Поскольку файл **sudoers** включает текущий хост в качестве критерия соответствия для строк конфигурации, вы можете использовать один главный файл **sudoers** в административном домене (то есть в регионе вашего сайта, в котором имена хостов и учетных записей пользователей гарантированно эквивалентны друг другу). Этот подход делает первоначальную настройку **sudoers** немного более сложным, но это отличная идея по нескольким причинам. Вам следует это реализовать в своей системе.

Основное преимущество этого подхода заключается в том, что нет никакой тайны о том, кто имеет какие разрешения и для каких хостов. Все записано в одном главном файле. Например, когда администратор покидает вашу организацию, нет необходимости отслеживать все хосты, на которых у этого пользователя могли быть права **sudo**. Когда необходимы изменения, вы просто изменяете главный файл **sudoers** и распространяете его.

Естественным следствием этого подхода является то, что разрешения **sudo** могут быть лучше выражены в терминах учетных записей пользователей, а не групп UNIX. Например, запись:

%wheel ALL = ALL

интуитивно понятна, но она откладывает перечисление привилегированных пользователей на каждой локальной машине. Вы не можете, взглянув на эту запись определить, кто ею охвачен, без необходимости исследовать рассматриваемую машину. Поскольку идея состоит в том, чтобы хранить всю необходимую информацию в одном месте, лучше избегать такого типа группировки при совместном использовании файла **sudoers** в сети. Конечно, если членство в ваших группах жёстко координируется по всему сайту, лучше использовать группы.

Распространение файла **sudoers** лучше всего достигается с помощью более расширенной системой управления конфигурациями, как описано в Главе 23. Но если вы еще не достигли такого уровня организации, вы легко можете распространять его вручную. Однако будьте осторожны: установка поддельного файла **sudoers** — это путь к катастрофе. Это также своеобразный способ отслеживания целостности файлов; см. страницу 1061.

В отсутствие системы управления конфигурациями лучше всего использовать скрипт "pull" который запускается из **cron** на каждом хосте. Используйте **scp** для копирования текущего файла **sudoers** из известного центрального репозитория, затем проверьте его с помощью **visudo -c -f** *newsudoers* перед установкой, чтобы убедиться, что формат является приемлемым для локального **sudo**. **scp** проверяет ключ хоста удаленного сервера, гарантируя, что файл **sudoers** поступает с доверенного хоста, а не с поддельного сервера.

Требования к имени хоста могут быть немного изощренными при совместном использовании файла **sudoers**. По умолчанию **sudo** использует вывод команды **hostname** в качестве текста, который нужно сопоставить. В зависимости от соглашений, используемых в вашем сайте, это имя может включать или не включать часть домена (например, "anchor" или "anchor.cs.colorado.edu"). В любом случае имена хостов, указанные в файле **sudoers**, должны совпадать с реальными именами хостов. Вы можете включить опцию fqdn в файле **sudoers**, чтобы попытаться привести локальные имена хостов в их полные формы.

Сопоставление имен хостов становится еще более требовательным в облаке, где имена инстансов часто по умолчанию генерируются по алгоритмическим шаблонам. **sudo** понимает простые символы, соответствующие шаблонам (универсальность (globbing)) в именах хостов, поэтому рассмотрите возможность использования схемы именования, которая содержит некоторые указания на классификацию безопасности каждого хоста с точки зрения **sudo**.

Кроме того, вы можете использовать виртуальные сетевые функции облачного провайдера для разделения узлов по IP-адресу, а затем сопоставлять IP-адреса вместо имен хостов из файла **sudoers**.

**Отключение учетной записи root**

Если ваш сайт стандартизирует использование **sudo**, вы вряд ли будете использовать его для реальных паролей root. Большая часть вашей административной команды никогда не будет иметь возможности использовать их.

Этот факт ставит вопрос о том, нужен ли вообще пароль root. Если вы решите, что это не так, вы можете полностью отключить root, установив зашифрованный пароль root в \* или в некоторую другую фиксированную произвольную строку. В Linux **passwd -l** "блокирует" учетную запись, добавляя "!" к зашифрованному паролю.

"\*" и "!" являются просто договоренностью; никакое программное обеспечение не проверяет их явно. Их эффект происходит из их недействительных хэшей паролей. В результате попытки проверить пароль root просто терпят неудачу.

Основной эффект блокировки учетной записи root заключается в том, что root не может войти в систему, даже в консоли. Ни один из пользователей не может успешно запустить **su**, потому что для этого также требуется проверка пароля root. Однако учетная запись root продолжает существовать, и все программное обеспечение, которое обычно выполняется от имени root, продолжает это делать. В частности, **sudo** работает нормально.

Основное преимущество отключения учетной записи root заключается в том, что вам не нужно записывать пароль root и управлять им. Вы также исключаете возможность компрометации пароля root, но это более приятный побочный эффект, чем веская причина для работы без пароля вообще. Редко используемые пароли уже подвержены низкому риску компрометации.

Особенно полезно хранить реальный пароль root на физических компьютерах (в отличие от облачных или виртуальных инстансов, см. Главы 9 и 24). Реальным компьютерам может потребоваться помощь, когда возникают проблемы с оборудованием или настройками, которые мешают процессу **sudo** или загрузке. В этих случаях хорошо иметь традиционную доступную учетную запись root в качестве аварийного резерва.

Ubuntu поставляется с заблокированной учетной записью root, и весь административный доступ осуществляется через **sudo** или его GUI эквивалент. Если вы предпочитаете использовать root, правильнее будет установить пароль для root в Ubuntu, а затем разблокировать учетную запись с помощью **sudo passwd -u root**.

**Системные учетные записи, отличные от root**

Только пользователь root имеет для ядра Linux особый статус. Есть, однако, еще несколько псевдоаккаунтов, которые применяются для системных целей. Эти фиктивные учетные записи можно идентифицировать по значениям UID, которые обычно меньше 100. Как правило, учетные записи с UID меньше 10 являются системными учетными записями, а значения UID от 10 до 100 отведены для псевдопользователей, связанных с определенными компонентами программного обеспечения.

*См. стр. 250 для получения дополнительной информации о файлах* ***shadow*** *и* ***master.passwd****.*

Обычно принято заменять зашифрованное поле пароля этих специальных пользователей в файле **shadow** или **master.passwd** звездочкой, чтобы нельзя было войти в систему под этими аккаунтами. Чтобы защититься от средств атаки на основе дистанционного входа в систему (когда вместо паролей используются файлы SSH-ключей), укажите в качестве командных интерпретаторов (вместо **/bin/bash** или **/bin/sh**) **/bin/false** или **/bin/nologin**.

Как и в случае с учетными записями пользователей, большинство систем определяют множество системных групп, которые имеют одинаково низкие GID.

Файлы и процессы, которые являются частью операционной системы, но которые не должны принадлежать пользователю root, иногда назначаются пользователям "bin" или "daemon". Считается, что это поможет избежать риска, связанного с действиями от имени суперпользователя. Однако ввиду не убедительности подобной аргументации в современных системах часто используется только учетная запись root.

Основное преимущество создания псевдоаккаунтов и псевдогрупп заключается в том, что их использование более безопасно, чем использование root для получения доступа к определенным группам ресурсов. Например, базы данных часто реализуют собственные сложные системы контроля доступа. С точки зрения ядра они работают как псевдопользователь, такой как "mysql", который владеет всеми ресурсами, связанными с базой данных.

*См. стр. 800 для получения дополнительной информации об учетной записи nobody.*

Сетевая файловая система — NFS (Network File System) — использует учетную запись "nobody" для представления суперпользователей в других системах. Чтобы лишить суперпользователей в удаленных системах их исключительных прав, излишних в данном контексте, NFS должна на время сеанса удаленного доступа заменить UID 0 чем-то другим, но не локальным UID 0. Этой цели как раз и служит учетная запись "nobody". В NFSv4 эта учетная запись также применяется к удаленным пользователям, которые не соответствуют действительной локальной учетной записи.

Пользователю nobody не нужны специальные права доступа, и он не должен владеть никакими файлами. Если бы ему принадлежали какие-то файлы, то над ними получили бы контроль суперпользователи, регистрирующиеся в удаленных системах.

**3.3 Развитие стандартной модели контроля доступа**

В предыдущих разделах описываются основные концепции традиционной модели управления доступом. Несмотря на то, что систему управления доступом можно описать буквально двумя страницами, она выдержала проверку временем благодаря своей простоте, предсказуемости и способности удовлетворять большинство требований, предъявляемых к управлению доступом на среднестатистическом узле. Все UNIX- и Linux-версии продолжают поддерживать эту модель, которая широко используется в наши дни.

Так как эта модель фактически реализована и поставляется на современных операционных системах, она включает в себя ряд важных доработок. Три уровня программного обеспечения способствуют нынешнему статус-кво:

- Стандартная модель, описанная в этом пункте.

- Расширения, которые обобщают и точно настраивают эту базовую модель.

- Расширения ядра, которые реализуют альтернативные подходы.

Эти категории не столько архитектурные слои, сколько исторические артефакты. Ранние UNIX-производные системы использовали стандартную модель, но ее недостатки были широко известны уже тогда. Со временем сообщество начало разрабатывать обходные пути для решения некоторых более насущных проблем. В интересах поддержания совместимости и поощрения широкого распространения, изменения, как правило, структурировались для совершенствования традиционной системы. Некоторые из этих настроек (например, PAM) теперь считаются стандартами UNIX.

За последнее десятилетие были сделаны большие успехи в области модуляции систем контроля доступа. Эта эволюция позволяет еще более радикально изменить контроль доступа. Мы рассматриваем некоторые из наиболее распространенных подключаемых параметров для Linux и FreeBSD, начиная со страницы 83.

Сейчас мы рассмотрим некоторые из прозаических расширений, которые поставляются с большинством систем. Во-первых, мы рассмотрим проблемы, которые пытаются решить эти расширения.

**Недостатки стандартной модели**

Несмотря на свою элегантность, стандартная модель имеет некоторые очевидные недостатки.

- Начнем с того, что учетная запись root представляет собой потенциальную точку отказа. Если она скомпрометирована, целостность всей системы нарушается и, фактически, нет ограничений ущерба, который может нанести злоумышленник.

- Единственный способ разделить специальные привилегии суперпользователя состоит в написании setuid-программ. К сожалению, как показывает непрекращающийся поток обновлений для средств защиты, довольно трудно написать по-настоящему безопасное программное обеспечение. Любая программа setuid является потенциальной целью для атаки.

- Стандартная модель не является достаточно прочной для использования в сети. Ни один компьютер, к которому непривилегированный пользователь имеет физический доступ, не может гарантировать, что он точно представляет принадлежность выполняемых процессов. Кто может утверждать, что такой-то пользователь не переформатировал диск и не инсталлировал собственную операционную систему со своими UID?

- В стандартной модели определение группы является привилегированной операцией. Например, универсальный пользователь не может выразить намерение, что только alice и bob должны иметь доступ к определенному файлу.

- Поскольку многие правила, связанные с управлением доступом, встроены в код отдельных команд и демонов (классическим примером является программа **passwd**), невозможно переопределить поведение системы, не модифицируя исходный код и не перекомпилируя программы. В реальном мире это непрактично и подвержено ошибкам.

- Стандартная модель также практически не поддерживает аудит или протоколирование. Вы можете видеть, к каким группам UNIX принадлежит пользователь, но вы не можете определить, что такое членство в группах позволяет пользователю делать. Кроме того, нет реального способа отслеживать использование повышенных привилегий или посмотреть, какие операции они выполнили.

**PAM: подключаемые модули аутентификации**

*См. стр. 250 для получения дополнительной информации о файлах* ***shadow*** *и* ***master.passwd****.*

Учетные записи пользователей традиционно защищены паролями, хранящимися (в зашифрованном виде) в файле **/etc/shadow** или **/etc/master.passwd** или в эквивалентной сетевой базе данных. Многим программам может потребоваться проверка учетных записей, включая **login**, **sudo**, **su** и любую программу, которая принимает логины на рабочей станции с графическим интерфейсом.

Эти программы действительно не должны иметь жестко запрограммированных ожиданий о том, как пароли должны быть зашифрованы или проверены. В идеале они даже не должны предполагать, что пароли вообще используются. Что делать, если вы хотите использовать биометрическую идентификацию, сетевую систему идентификации или двухфакторную аутентификацию? Подключаемые модули аутентификации приходят на помощь!

PAM - это оболочка для различных библиотек аутентификации, специфичных для различных методов аутентификации. Администраторы определяют методы аутентификации, которые они хотят использовать в системе, а также соответствующие контексты для каждого из них. Программы, требующие аутентификации пользователя, просто вызывают систему PAM, а не реализуют собственные методы аутентификации. В свою очередь PAM вызывает библиотеку аутентификации, указанную системным администратором.

Грубо говоря, PAM - это технология аутентификации, а не технология контроля доступа. То есть вместо того, чтобы решать вопрос "Имеет ли пользователь X разрешение на выполнение операции Y?", она помогает ответить на предшествующий вопрос: "Откуда я знаю, что это действительно пользователь X?".

PAM является важным компонентом цепочки контроля доступа в большинстве систем, а конфигурирование PAM является общей административной задачей. Более подробную информацию о PAM вы можете найти в Главе "Single Sign-On", начиная со страницы 590.

**Сетевой протокол криптографической аутентификации Kerberos**

Как и РАМ, протокол Kerberos призван решать проблемы аутентификации, а не управления доступом. Но если РАМ можно назвать фреймворком аутентификации, то Kerberos — это конкретный метод аутентификации. В сайтах, которые используют Kerberos, PAM и Kerberos, как правило, работают вместе, PAM — это оболочка и Kerberos - фактическая реализация.

Kerberos использует доверенный сторонний сервер для выполнения задач аутентификации в масштабах всей сети. Вместо самоаутентификации на своем компьютере вы предоставляете свои учетные данные службе Kerberos. Затем она выдает вам зашифрованные учетные данные, которые вы можете представить другим службам в качестве подтверждения вашей личности.

Kerberos — это зрелая технология, которая широко используется на протяжении десятилетий. Она является стандартной системой аутентификации, используемой в Windows, и является частью системы Active Directory. Больше о протоколе Kerberos читайте на стр. 586.

**Списки управления доступом к файловой системе**

Управление доступом к файловой системе - важнейшая часть систем UNIX и Linux, и поэтому ее совершенствование составляет первоочередную цель разработчиков этих систем. Особое внимание было уделено поддержке списков доступа к файлам и каталогам (access control lists - ACL) как обобщению традиционной модели привилегий пользователь/группа/все\_остальные, устанавливаемых сразу для нескольких пользователей и групп.

Списки ACL являются частью реализации файловой системы, поэтому их поддержку в явном виде должна обеспечивать используемая вами файловая система. К счастью, в настоящее время все основные файловые системы UNIX и Linux поддерживают ACL в той или иной форме.

*Дополнительную информацию о NFS см. в Главе 21 "Сетевая файловая система".*

Поддержка ACL-списков в общем случае реализуется в двух формах: в виде проекта стандарта POSIX, который (хоть и без формального подтверждения) получил широкое признание, и система, стандартизированная NFSv4, который адаптирован к ACL-спискам Microsoft Windows. Оба стандарта ACL более детально описаны в Главе "Файловые системы", начиная со страницы 140.

**Linux capabilities**

Системы capabilities разделяют полномочия учетной записи root на несколько (~30) отдельных разрешений.

Linux-версия возможностей происходит из несуществующего проекта POSIX 1003.1e, который кое-как развивается несмотря на то, что никогда не был официально утвержден в качестве стандарта. В дополнение к этому неся клеймо зомби, возможности Linux вызывают раздражение теоретиков из-за несоответствия академической концепции системы возможностей. Неважно; они уже здесь, и Linux называет их возможностями, поэтому мы тоже будем.

Возможности могут быть унаследованы от родительского процесса. Они также могут быть включены или отключены атрибутами, установленными в исполняемом файле, в процессе, напоминающем выполнение setuid. Процессы могут отказаться от возможностей, которые они не планируют использовать.

Традиционные полномочия root — это просто объединение всех возможных capabilities, поэтому существует довольно прямое сопоставление между традиционной моделью и моделью возможностей. Модель возможностей просто более детализирована.

Например, возможность Linux, называемая CAP\_NET\_BIND\_SERVICE, управляет способностью процесса связываться с привилегированными сетевыми портами (с номерами до 1 024). Некоторые демоны, которые традиционно работают как root, нуждаются только в этой конкретной суперсиле. В мире возможностей такой демон теоретически может работать как непривилегированный пользователь и получать возможность привязки портов из своего исполняемого файла. Пока демон явно не проверяет, работает ли он от имени root, он даже не должен быть осведомлен о возможностях.

Действительно ли все это делается в реальном мире? Ну, нет. Как это обычно происходит, возможности эволюционировали, чтобы стать более содействующей технологией, чем системой, ориентированной на пользователя. Они широко используются в системах более высокого уровня, таких как AppArmor (см. стр. 87) и Docker (см. Главу 25), но редко используются самостоятельно.

Для администраторов полезно просмотреть man-страницу **capabilities**(7), чтобы получить представление о том, что входит в каждую из групп возможностей.

**Пространство имен Linux**

Linux может разделить процессы на иерархические разделы ("пространства имен"), из которых они видят только подмножество системных файлов, сетевых портов и процессов. Помимо других эффектов, эта схема действует как форма упреждающего контроля доступа. Вместо того чтобы основывать решения по управлению доступом на потенциально тонких критериях, ядро просто отрицает существование объектов, которые не видны изнутри данного пространства.

Внутри раздела применяются обычные правила контроля доступа, и в большинстве случаев изолированные процессы даже не осознают, что они были ограничены. Поскольку ограничение является необратимым, процессы могут выполняться как корневые внутри раздела, не опасаясь, что они могут представлять опасность для других частей системы.

Этот хитрый трюк является одной из основ контейнеризации программного обеспечения и его наиболее известной реализации, Docker. Полная система намного сложнее и включает в себя такие расширения, как доступ к файловой системе copy-on-write. Больше о контейнерах мы рассказываем в Главе 25.

Пространство имен, как форма контроля доступа, является относительно грубым подходом. Построение правильно настроенного окружения для процессов, в которых они будут жить, также несколько сложно. В настоящее время эта технология применяется в основном к дополнительным сервисам в отличие от встроенных компонентов операционной системы.

**3.4 Современный контроль доступа**

Учитывая широкий спектр вычислительных сред в мире и неоднозначный успех усилий по продвижению стандартной модели, разработчики ядра неохотно выступали в качестве посредников в более широких дебатах по поводу контроля доступа. В мире Linux ситуация достигла своего апогея в 2001 году, когда Агентство национальной безопасности США предложило интегрировать свою улучшенную систему Linux (SELinux) в ядро в качестве стандарта.

По нескольким причинам разработчики ядра сопротивлялись этому слиянию. Вместо принятия SELinux или другой, альтернативной системы, они разработали Linux Security Modules API, интерфейс уровня ядра, который позволяет системам контроля доступа интегрироваться в качестве подгружаемых модулей ядра.

Системы на основе LSM (Loadable Security Modules) не имеют никакого эффекта до тех пор, пока пользователи не загрузят и не активируют их. Этот факт снижает барьеры для включения в стандартное ядро, и Linux теперь поставляется с SELinux и четырьмя другими системами (AppArmor, Smack, TOMOYO и Yama), готовыми к работе.

Разработки на стороне BSD были примерно параллельны разработкам Linux, во многом благодаря работе Роберта Уотсона над TrustedBSD. Этот код был включен в FreeBSD начиная с версии 5. Он также предоставляет технологию песочницы приложений, используемую в macOS и iOS от Apple.

Когда несколько модулей контроля доступа активны одновременно, операция должна быть одобрена всеми из них, чтобы быть разрешенной. К сожалению, система LSM требует явного сотрудничества между активными модулями, и ни один из текущих модулей не включает эту функцию. На данный момент системы Linux фактически ограничены выбором одного дополнительного модуля LSM.

**Отдельные экосистемы**

Контроль доступа по своей сути является проблемой на уровне ядра. За исключением списков контроля доступа файловой системы (см. стр. 140), между системами практически отсутствует стандартизация в отношении альтернативных механизмов контроля доступа. В результате каждое ядро имеет свой собственный массив доступных реализаций, и ни одна из них не является кроссплатформенной.

Поскольку дистрибутивы Linux имеют общее ядро, все дистрибутивы Linux теоретически совместимы со всеми различными предложениями безопасности Linux. Но на практике это не так: все эти системы нуждаются в поддержке на уровне пользователя в виде дополнительных команд, модификаций компонентов на уровне пользователя и профилей безопасности для демонов и служб. Следовательно, каждый дистрибутив имеет только один или два механизма контроля доступа, которые он активно поддерживает (если это так).

**Принудительный контроль доступа**

Стандартная модель UNIX считается формой "избирательного контроля доступа", поскольку она позволяет владельцам объектов с контролем доступа, устанавливать на них разрешения. Например, вы можете разрешить другим пользователям просматривать содержимое вашего домашнего каталога или написать программу setuid, которая позволит другим пользователям отправлять сигналы вашим процессам.

Избирательный контроль доступа не дает никаких особых гарантий безопасности данных пользовательского уровня. Недостатком выдачи прав пользователям устанавливать разрешения является то, что пользователи могут устанавливать разрешения; никто не знает, что они могут делать со своими собственными файлами. И даже при самых лучших намерениях и обучении пользователи могут совершать ошибки.

Системы обязательного контроля доступа (он же MAC - Mandatory access control) позволяют администраторам писать политики контроля доступа, которые переопределяют или дополняют избирательные разрешения традиционной модели. Например, можно установить правило, что домашние каталоги пользователей доступны только их владельцам. Тогда не имеет значения, если пользователь делает частную копию конфиденциального документа и небрежно относится к разрешениям документа; никто другой не может заглянуть в домашний каталог этого пользователя в любом случае.

Возможности MAC — это технология, позволяющая реализовать модели безопасности, такие как "многоуровневая система безопасности" Министерства обороны. В этой модели политики безопасности управляют доступом в соответствии с воспринимаемой чувствительностью контролируемых ресурсов. Пользователям назначается Классификация безопасности из структурированной иерархии. Они могут читать и записывать элементы на том же уровне классификации или ниже, но не могут получить доступ к элементам на более высоком уровне классификации. Например, пользователь с доступом "секретно" может читать и записывать "секретные" объекты, но не может читать объекты, классифицированные как "совершенно секретные".

Если вы не обрабатываете конфиденциальные данные для правительственного учреждения, маловероятно, что вы когда-либо столкнетесь с такими всеобъемлющими "сторонними" моделями безопасности или вам потребуется их развернуть. Чаще всего MAC используется для защиты отдельных служб, и в противном случае он остается вне поля зрения пользователей.

Хорошо реализованная политика MAC основана на принципе наименьших привилегий (разрешение доступа только при необходимости), так же как правильно настроенный файервол позволяет пропускать только специально распознанные службы и клиенты. MAC может предотвратить компрометацию системы программным обеспечением, имеющим уязвимости (например, переполнение буфера), ограничив область нарушения несколькими конкретными ресурсами, требуемыми этим программным обеспечением.

MAC, к сожалению, стал чем-то вроде модного слова, синонимом "расширенному контролю доступа". Даже общий API безопасности FreeBSD называется интерфейсом MAC несмотря на то, что некоторые плагины не предлагают никаких реальных функций MAC.

Доступные системы MAC варьируются от полных замен для стандартной модели до легких расширений, которые адресуют конкретные домены и варианты использования. Общим направлением среди реализаций MAC является то, что они обычно добавляют централизованные, написанные администратором (или поставляемые поставщиком) политики в систему управления доступом наряду с обычным сочетанием разрешений файлов, списков управления доступом и атрибутов процесса.

Независимо от области применения, MAC представляет собой потенциально значительное отклонение от стандартной системы. В этом случае программы, ожидающие иметь дело со стандартной моделью безопасности UNIX, могут "сильно удивиться". Прежде чем приступить к полномасштабному развертыванию MAC, убедитесь, что вы понимаете правила логирования этого модуля и знаете, как определить и устранить проблемы, связанные с MAC.

**Управление доступом на основе ролей**

Еще одна функция, обычно проверяемая системами контроля доступа, — это управление доступом на основе ролей (RBAC), теоретическая модель, описанная в 1992 году Дэвидом Феррайоло и Риком Куном. Основная идея заключается в том, чтобы добавить слой косвенных вычислений для управления доступом. Разрешения вместо того, чтобы назначаться непосредственно пользователям, назначаются промежуточным надстройкам, известным как "роли", а роли, в свою очередь, назначаются пользователям. Чтобы принять решение о контроле доступа, система перечисляет роли текущего пользователя и проверяет, есть ли у какой-либо из этих ролей соответствующие разрешения.

Роли похожи по своей концепции на группы UNIX, но они более общие, потому что их можно использовать вне контекста файловой системы. Роли также могут иметь иерархическое отношение друг к другу, что значительно упрощает администрирование. Например, можно определить роль "старший администратор", которая имеет все разрешения "администратора" плюс дополнительные разрешения X, Y и Z.

Многие варианты UNIX, включая Solaris, HP-UX и AIX, включают в себя некоторую форму встроенной системы RBAC. Linux и FreeBSD не имеют отдельного, родного средства RBAC. Тем не менее, он встроен в несколько более полных опций MAC.

**SELinux: Повышенная Безопасность Linux**

SELinux является одной из старейших реализаций Linux MAC и является продуктом Агентства национальной безопасности США. В зависимости от точки зрения это может быть источником либо утешения, либо подозрений7.

*7. Если вы более подозрительны, стоит отметить, что в рамках дистрибутива ядра Linux кодовая база SELinux открыта для проверки.*

SELinux использует максималистский подход, и он реализует практически все возможности MAC и RBAC, которые можно себе представить. Несмотря на то, что он завоевал прочные позиции в нескольких дистрибутивах, его, как известно, трудно администрировать и устранять неполадки. Эта неопубликованная цитата из предыдущей версии страницы Википедии SELinux показывает разочарование, испытываемое многими системными администраторами:

*Интересно, что хотя заявленный смысл существования SELinux состоит в том, чтобы облегчить создание индивидуальных политик контроля доступа, специально настроенных на организационные методы правила хранения данных, вспомогательные программные средства настолько редки и недружественны, что поставщики выживают в основном за счет "консалтинга", который обычно принимает форму постепенных изменений стандартных политик безопасности.*

Несмотря на свою административную сложность, внедрение SELinux медленно растет, особенно в таких сферах, как правительство, финансы и здравоохранение, которые обеспечивают строгие и конкретные требования безопасности. Это также стандартная часть платформы Android.

Наше общее мнение относительно SELinux заключается в том, что он способен принести больше вреда, чем пользы. К сожалению, этот вред может проявляться не только в потере времени и ухудшении работы системных администраторов, но и, как это ни парадоксально, в нарушении безопасности. Сложные модели трудно объяснить, и SELinux на самом деле не является легким для понимания; хакеры, которые сосредотачиваются на нем, понимают систему гораздо более глубже, чем средний сисадмин.

В частности, разработка политики SELinux — это сложная задача. Например, чтобы защитить новый демон, политика должна тщательно перечислить все файлы, каталоги и другие объекты, к которым процессу нужен доступ. Для сложных программ, таких как **sendmail** или **httpd**, эта задача может быть довольно сложной. По крайней мере одна компания предлагает трехдневный курс по разработке политик.

К счастью, многие общие политики доступны в интернете, и большинство дистрибутивов с поддержкой SELinux поставляются с разумными настройками по умолчанию. Они могут быть легко установлены и настроены для вашей конкретной среды. Полнофункциональный редактор политик, призванный облегчить применение политик, можно найти по адресу seedit.sourceforge.net.

SELinux хорошо поддерживается как Red Hat (и, следовательно, CentOS), так и Fedora. В Red Hat он включен по умолчанию.

Debian и SUSE Linux также имеют некоторую поддержку SELinux, но вы должны установить дополнительные пакеты. Сама система менее агрессивна в своей конфигурации по умолчанию.

Ubuntu унаследовала некоторую поддержку SELinux от Debian, но в течение последних нескольких выпусков Ubuntu была сосредоточена на AppArmor (см. стр. 87). Некоторые рудиментарные пакеты, связанные с SELinux, все еще доступны, но они, как правило, не являются актуальными.

/etc/selinux/config — это элемент управления верхнего уровня для SELinux. Интересные строки

SELINUX=enforcing

SELINUXTYPE=targeted

Первая строка имеет три возможных значения: принудительное, разрешающее или отключенное (enforcing, permissive, disabled). Параметр enforcing обеспечивает применение загруженной политики и запрещает нарушения. permissive позволяет производить нарушения, но регистрирует их в системном журнале, что полезно для отладки и разработки политики. disabled полностью отключает SELinux.

SELINUXTYPE ссылается на имя базы данных политики, которая будет применена. По сути, это имя подкаталога в **/etc/selinux**. Одновременно может быть активна только одна политика, и доступные наборы политик зависят от системы.

Политика Red Hat по умолчанию является целевой, которая определяет дополнительную безопасность для нескольких демонов, которые Red Hat явно защитил, но оставляет остальную часть системы в покое. Раньше существовала отдельная политика под названием strict, которая применяла MAC ко всей системе, но теперь эта политика была объединена в targeted. Удалите модули unconfined и unconfineduser с помощью **semodule -d** для применения MAC для всей системы.

Red Hat также определяет политику mls, которая реализует многоуровневую безопасность в стиле DoD. Вы можете установить его отдельно с помощью **yum install selinux-policy-mls**.

Если вы заинтересованы в разработке собственных политик SELinux, ознакомьтесь с утилитой **audit2allow**. Она строит определения политики из логов нарушений. Идея заключается в том, чтобы разрешить защиту подсистемы таким образом, чтобы ее нарушения регистрировались, но не применялись. Затем вы можете провести подсистему через все этапы и построить политику, которая позволяет все, что подсистема фактически сделала. К сожалению, трудно гарантировать полный охват всех этапов и кода с помощью такого специального подхода, поэтому автогенерированные профили вряд ли будут идеальными.

**AppArmor**

AppArmor является продуктом компании Canonical, Ltd., выпускающей дистрибутивы Ubuntu. Он поддерживается Debian и Ubuntu, но также был принят в качестве стандарта дистрибутивами SUSE. Ubuntu и SUSE включают его при установке по умолчанию, хотя набор защищенных служб не является обширным.

AppArmor реализует форму MAC и предназначен в качестве дополнения к традиционной системе управления доступом UNIX. Несмотря на то, что возможна любая конфигурация, AppArmor не является ориентированной на пользователя системой. Его главная цель - обеспечение безопасности сервисов; то есть ограничение ущерба, который могут нанести отдельные программы, если они будут скомпрометированы или "взбесятся".

Защищенные программы по-прежнему подвержены всем ограничениям, налагаемым стандартной моделью, но, кроме того, ядро фильтрует их деятельность через определенный и специфичный для конкретной задачи профиль AppArmor. По умолчанию AppArmor отклоняет все запросы, поэтому профиль должен явно указывать все, что разрешено делать процессу.

Программы без профилей, такие как пользовательские оболочки, не имеют особых ограничений и работают так, как если бы AppArmor не был установлен.

Эта роль обеспечения безопасности сервиса по существу является той же конфигурацией, которая реализована SELinux в целевой среде Red Hat. Однако AppArmor разработан более конкретно для обеспечения безопасности сервиса, поэтому он обходит некоторые из наиболее загадочных нюансов SELinux.

Профили AppArmor хранятся в **/etc/apparmor.d**, и они относительно читаемы даже без детального знания системы. Например, вот профиль для демона **cups-browsed**, входящего в систему печати на Ubuntu:

#include <tunables/global>

/usr/sbin/cups-browsed {

#include <abstractions/base>

#include <abstractions/nameservice>

#include <abstractions/cups-client>

#include <abstractions/dbus>

#include <abstractions/p11-kit>

/etc/cups/cups-browsed.conf r,

/etc/cups/lpoptions r,

/{var/,}run/cups/certs/\* r,

/var/cache/cups/\* rw,

/tmp/\*\* rw,

# Site-specific additions and overrides. See local/README.

#include <local/usr.sbin.cups-browsed>

}

Большая часть этого кода является модульным шаблоном. Например, этот демон должен выполнять поиск имен хостов, поэтому профиль интерполирует **abstractions/nameservice**, что дает доступ к библиотекам разрешения имен, **/etc/nsswitch.conf**, **/etc/hosts**, сетевым портам, используемым с LDAP, и так далее.

Информация профилирования, специфичная для данного демона, состоит (в данном случае) из списка файлов, к которым демон может получить доступ, а также разрешений, назначенных для каждого файла. Синтаксис сопоставления шаблонов немного своеобразен: **\*\*** может соответствовать нескольким компонентам пути, а {var/,} соответствует тому, появляется ли var/ в этом месте или нет.

Даже этот простой профиль довольно сложен, если заглянуть под капот. Когда все инструкции #include расширены, длина профиля составляет почти 750 строк. (И мы выбрали этот пример для краткости).

AppArmor обращается к файлам и программам по имени пути, что делает профили читаемыми и независимыми от любой конкретной реализации файловой системы. Однако такой подход является своего рода компромиссом. Например, AppArmor не распознает жесткие ссылки как указывающие на одну и ту же базовую сущность.

**3.5 Рекомендуемая литература**

Ferraiolo, David F., D. Richard Kuhn, and Ramaswamy Chandramouli. *Role-Based Access Control (2nd Edition)*. Boston, MA: Artech House, 2007.

Haines, Richard. *The SELinux Notebook (4th Edition)*. 2014. Этот компендиум информации, связанной с SELinux, наиболее близок к официальной документации. Он доступен для скачивания с сайта freecomputerbooks.com.

Vermeulen, Sven. *SELinux Cookbook*. Birmingham, UK: Packt Publishing, 2014. Эта книга включает в себя множество практических советов по работе с SELinux. Она охватывает как сервисное обеспечение, так и модели безопасности, ориентированные на пользователя