ФГБОУ ВПО «Воронежский государственный
технический университет»

Кафедра систем информационной безопасности

133-2015

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к практическим занятиям по дисциплинам
«Операционные системы»,
«Безопасность операционных систем»
для студентов специальностей
090301 «Компьютерная безопасность»,
090303 «Информационная безопасность
автоматизированных систем»
очной формы обучения

Воронеж 2015

Составители: д-р техн. наук А. Ю. Савинков, аспирант Г. А. Савенков

УДК 004.056.5

Методические указания к практическим занятиям по дисциплинам «Операционные системы», «Безопасность операционных систем» для студентов специальностей 090301 «Компьютерная безопасность», 090303 «Информационная безопасность автоматизированных систем» очной формы обучения / ФГБОУ ВПО «Воронежский государственный технический университет»; сост. А. Ю. Савинков, Г. А. Савенков. Воронеж, 2015. 43 с.

Методические указания посвящены исследованию задачи разграничения доступа, ее организации в различных современных операционных системах. На практику вынесено изучение основных команд пользовательского интерфейса для управления правами доступа в LINUX.

Методические указания подготовлены в электронном виде в текстовом редакторе MS Word 2013 и содержатся в файле Савинков_ПЗ_ОС+БОС.pdf.

Табл. 2. Ил. 4. Библиогр.: 11 назв.

Рецензент д-р техн. наук, проф. А. Г. Остапенко

Ответственный за выпуск зав. кафедрой д-р техн. наук, проф. А. Г. Остапенко

Издается по решению редакционно-издательского совета Воронежского государственного технического университета

1. ЗАДАЧА РАЗГРАНИЧЕНИЯ ДОСТУПА

Защита данных от несанкционированного доступа при хранении и обработке является одной из важнейших задач современной многопользовательской операционной системы. Для ее решения необходимо контролировать полномочия пользователей при обращениях к информационным ресурсам. Функциональность операционной системы по контролю полномочий пользователей при доступе ресурсам называют *разграничение доступа*.

В качестве защищаемых ресурсов в большинстве случаев (но не всегда) выступают файлы, размещенные на устройствах долговременной памяти. Но поскольку современные операционные системы используют файловый интерфейс для доступа практически ко всем видам ресурсов, включая аппаратные устройства (все ресурсы адресуются через файловые дескрипторы, а для передачи данных используются системные вызовы чтения и записи файлов), в дальнейшем можно без потери общности ассоциировать все ресурсы с *файлами*.

Помимо разграничения доступа, современные операционные системы, как правило, поддерживают функциональность *контроля доступа*. Концепция контроля доступа состоит в фиксации в системных журналах фактов доступа пользователей к ресурсам с указанием имени (идентификатора) пользователя, времени и вида доступа. Хотя сам по себе контроль доступа не защищает ресурсы, он позволяет проследить историю доступа к ресурсу и может помочь выявить нарушения разграничения доступа, в том числе проследить действия злоумышленника.

2. РЕАЛИЗАЦИЯ РАЗГРАНИЧЕНИЯ ДОСТУПА

Для реализации разграничения доступа в операционной системе должна быть доступна информация о разрешенных видах доступа для всех пользователей относительно всех ресурсов. В общем виде необходимую информацию можно представить в форме *таблицы разграничения доступа*, как показано в табл. 1.

Таблица 1

Таблица разграничения доступа

	ресурс 1	ресурс 2	…	ресурс M
пользователь 1	***недоступно***	***чтение***		***запись***
пользователь 2	***полный доступ***	***полный доступ***		***недоступно***
⋮				
пользователь N	***запись***	***запись***		***полный доступ***

Однако использование таблицы разграничения доступа не является эффективным способом реализации разграничения доступа, поскольку большинство записей таблицы не будут содержать полезной информации: доступ к соответствующим ресурсам уже будет запрещен косвенно, через доступ к каталогам (папкам), содержащим ресурсы, доступ к родительским каталогам этих каталогов и т.п. В этой связи более эффективным решением является использование списков.

Список можно связать с пользователем и перечислить в нем все ресурсы, к которым пользователь имеет доступ. Такой список возможностей пользователя по доступу к ресурсам называется *мандат возможностей*.

Другой подход состоит в том, чтобы связать список с ресурсом и перечислить в нем всех пользователей, которые имеют доступ к этому ресурсу. Такой список называется *список контроля доступа*.

На практике, использовать список контроля доступа в большинстве случаев удобнее, поскольку он более короткий и

более стабильный, чем мандат возможностей. Количество пользователей, зарегистрированных в системе, обычно существенно меньше, чем количество файлов. Поэтому список контроля доступа, даже перечисляющий всех пользователей, будет относительно коротким. Кроме того, состав пользователей системы меняется значительно реже, чем состав файлов, следовательно, списки контроля доступа, единожды связанные с файлами, с большой вероятностью могут оставаться неизменными на протяжении всего срока существования этих файлов, а вот мандаты возможностей пользователей пришлось бы часто корректировать.

Помимо списков контроля доступа и мандатов возможностей, существуют и другие варианты хранения информации для разграничения доступа. Например, в *UNIX* используется *маска доступа* – крайне упрощенная модель списка контроля доступа, предусматривающая разделение пользователей на три категории: 1) пользователь-владелец файла; 2) пользователь-член группы, коллективно владеющей файлом; 3) все остальные пользователи. Маска доступа вместе с идентификаторами пользователя-владельца и группы-владельца хранятся в составе метаданных файла.

Несмотря на то, что модель разграничения доступа, принятая в *UNIX*, существует уже более 30 лет, она все еще широко используется, в том числе в современных *UNIX*-подобных операционных системах, включая *Linux*. В свою очередь, *Linux* или системы на его основе, например, *MCBC*, нашли широкое применение в сетевом и телекоммуникационном оборудовании (маршрутизаторы, коммутаторы, точки доступа беспроводных сетей, модемы, радиостанции профессиональной и военной связи и т.п.), а также в качестве операционной системы на рабочих станциях и автоматизированных рабочих местах операторов, особенно в условиях, когда предъявляются повышенные требования к информационной безопасности.

В этой связи необходимо более детально рассмотреть организацию и работу системы разграничения доступа в *UNIX* и *UNIX*-подобных системах.

3. РАЗГРАНИЧЕНИЕ ДОСТУПА В UNIX И UNIX-ПОДОБНЫХ ОПЕРАЦИОННЫХ СИСТЕМАХ

Функционирование системы разграничения доступа в *UNIX* и *UNIX*-подобных системах (в частности, *Linux*) базируется на маске доступа, назначаемой каждому файлу. Поскольку в *UNIX* доступ к ресурсам, включая устройства ввода-вывода, осуществляется через файлы, проверка прав доступа к файлам позволяет контролировать доступ к любым ресурсам.

Как уже отмечалось, в *UNIX* для каждого файла назначаются права доступа для трех категорий пользователей:

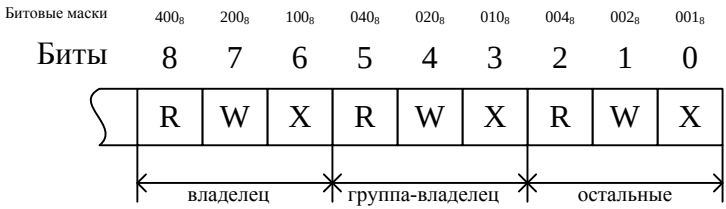
- права доступа для пользователя, определенного как владелец файла;
- права доступа для пользователей группы, определенной как группа-владелец файла;
- права доступа для всех остальных пользователей.

При создании нового файла его владельцем автоматически становится пользователь, создавший его. Группой-владельцем при этом становится первичная группа пользователя. В дальнейшем владелец файла может быть произвольно переопределен командой *chown* (от *change owner* – сменить владельца).

Таким образом, информация разграничения доступа, сопоставляемый с файлами, имеет простую структуру и фиксированный размер:

1. девять бит для хранения маски доступа для трех категорий пользователей;
2. два целых числа, идентифицирующих владельца файла и группу.

Эта информация встроена в метаданные файловой системы, конкретно в индексный узел (*inode*), описывающий файл. Первым полем структуры индексного узла является поле *mode* (режим), девять младших битов которого хранят информацию о правах доступа, как показано на рис. 1.



R – право чтения файла
W – право записи в файл
X – право исполнения программы из файла

Рис. 1. Битовые маски прав доступа в индексном узле

Идентификаторы пользователя, владеющего файлом, и группы-владельца файла также хранятся в индексном узле как целые числа.

Проверка прав доступа при обращении к объекту выполняется путем сравнения идентификатора пользователя и идентификаторов всех его групп (пользователь может одновременно входить в несколько групп) с соответствующими идентификаторами из индексного узла файла и последующим сопоставлением типа запрошенной операции с разрешенными видами доступа к файлу.

3.1. Права доступа для каталогов файловой системы

Применительно к обычному файлу, содержащему данные, определение прав доступа на запись, чтение и исполнение очевидно: право на чтение разрешает читать информацию из файла, право на запись позволяет редактировать файл и добавлять в него новые данные (при этом наличие права на запись не дает автоматическое право на чтение, чтение должно быть разрешено явно), право на исполнение позволяет запускать программу, хранящуюся в данном файле.

Для каталогов права доступа определяют следующие возможности:

- право на чтение для каталога позволяет получить только список имен файлов, содержащихся в данном каталоге, однако не позволяет получить более подробную информацию о файлах каталога, например их размеры;
- право на запись для каталога позволяет создавать и удалять файлы в каталоге (при этом права доступа к самим файлам не учитываются, так что пользователь может удалить файл, для которого он не имеет ни каких прав доступа);
- право на исполнение для каталога дает право просматривать атрибуты файлов, кроме того, право на исполнение требуется, чтобы перейти в данный каталог, т.е. сделать его текущим для пользователя.

Комбинация прав доступа «запрет чтения» и «разрешение исполнения» для каталога приводит к интересному эффекту, известному как *теневой каталог*.

С файлами такого каталога можно работать без ограничений, поскольку право на чтение необходимо только для получения списка имен файлов и не требуется для выполнения других операций с каталогом и его файлами.

Однако пользователь не сможет получить список файлов для этого каталога, т.е. он сможет работать только с файлами, для которых он заранее знает имена.

Организация теневых каталогов может быть полезной с точки зрения обеспечения информационной безопасности, в случае если к какому-то каталогу необходимо предоставить свободный доступ (например, из сети), но при этом нежелательно, чтобы непосвященные пользователи использовали его файлы.

3.2. Учет пользователей в UNIX

Для решения задачи разграничения доступа, операционная система должна вести учет пользователей, допущенных к работе в системе, и иметь возможность идентификации конкретного пользователя при попытках доступа к ресурсам.

В настоящее время известно множество версий *UNIX* и *UNIX*-подобных операционных систем, которые могут существенно различаться в деталях реализации. Тем не менее, базовые принципы учета пользователей во всех таких системах похожи (хотя в деталях, в том числе в составе, названиях и формате соответствующих файлов, могут быть различия). Поэтому мы рассмотрим традиционный подход, реализованный в традиционных версиях *UNIX*, также унаследованный в *Linux*. На основе понимания традиционного подхода, детали реализации, свойственные, например, современным версиям *FreeBSD* или *OpenBSD*, могут быть легко изучены по соответствующей документации.

Учетные записи пользователей в *UNIX* хранятся в обычном текстовом файле */etc/passwd*, по одной строке на каждого пользователя. Файл */etc/passwd* доступен для чтения любому пользователю, а для записи только системному администратору. Учетные записи пользователей хранятся в файле в следующем формате:

```
name:password:UID:GID:коментарий:дом_
                каталог:shell
```

Таким образом, учетная запись пользователя в операционной системе *UNIX* содержит семь полей, разделенных символом *':'*, и содержащих атрибуты пользователя. Прокомментируем назначение каждого из этих полей.

`name` – учетное имя пользователя. При каждом входе в систему пользователь указывает свое учетное имя, после чего система построчно просматривает файл */etc/passwd* в поисках зарегистрированного пользователя с данным именем. Если зарегистрированный пользователь найден, то система переходит

к процедуре аутентификации, если же учетная запись для пользователя с указанным именем не обнаружена в файле */etc/passwd*, то пользователь не зарегистрирован в системе, и ему будет отказано во входе в систему.

`password` – зашифрованный пароль пользователя. Если при входе некоторого пользователя в систему по его имени будет найдена учетная запись, то система зашифрует пароль, переданный пользователем, и сравнит этот зашифрованный пароль с зашифрованным паролем, хранящимся в файле */etc/passwd*. Если зашифрованные пароли совпадут, то пользователю будет разрешено войти в систему.

Особенностью алгоритма шифрации пароля в операционной системе *UNIX* является то, что алгоритм шифрации никогда не отобразит символ исходного пароля в символ '*'. Это значит, что если в поле `password` какого-либо пользователя стоит символ '*', то этот пользователь никогда не сможет войти в систему. Установить в поле `password` символ '*' является самым простым способом для системного администратора заблокировать вход пользователя в систему.

Как будет показано далее, хранение даже зашифрованного пароля в общедоступном файле */etc/passwd* является уязвимостью. Поэтому современные версии *UNIX* и *UNIX*-подобных систем вместо зашифрованного пароля ставят в поле `password` символы '!' или 'x', а зашифрованный пароль хранят в отдельном файле, доступном на чтение и запись только системному администратору.

Имя и внутренний формат файла паролей различаются в разных версиях *UNIX* и *UNIX*-подобных систем. Например, в *Linux* и *Solaris* пароли вынесены в файл */etc/shadow*, Системы *xBSD* обычно используют файл */etc/master.passwd*.

`комментарий` – произвольный текст, характеризующий пользователя, например, настоящее имя, номер телефона и т.п. Это информация для администратора системы, а не для операционной системы.

дом_каталог – домашний каталог пользователя. После входа в систему этот каталог будет установлен для пользователя текущим. Пользователь обычно имеет полный доступ к этому каталогу и всем его подкаталогам и должен располагать здесь все свои файлы.

shell – имя программы, которая будет автоматически запущена от имени пользователя при его входе в систему, после процедуры аутентификации. Обычно этой программой является один из нескольких возможных командных интерпретаторов, реализующих для пользователя командный интерфейс.

По существу, именно запуск командного интерпретатора означает фактический вход пользователя в систему. После запуска командного интерпретатора в системе появляется процесс, который исполняется от имени вновь вошедшего пользователя, и способен создавать новые процессы в интересах и по инициативе данного пользователя.

Главным компонентом программного интерпретатора является цикл приема команд, при этом командный интерпретатор большую часть времени находится в состоянии ожидания ввода пользовательской команды. Интерпретатор команд может запускать другие программы, имена которых указываются пользователем в командной строке, причем командный интерпретатор может передавать этим программам параметры, указанные в командной строке после имени программы. Отметим здесь, что большинство команд консоли в системе *UNIX* реализовано в виде внешних программ. Для запуска новой программы командный интерпретатор создает новый процесс и устанавливает для него программу из указанного файла. Новый процесс наследует все права родительского процесса, а так как процесс командного интерпретатора является прямым или косвенным предком для всех процессов данного пользователя, то права, которые получил командный интерпретатор при своем запуске на этапе входа пользователя в систему, в конечном итоге определяют права всех программ, запускаемых пользователем.

Завершение командного интерпретатора означает выход пользователя из системы.

UID – числовой идентификатор пользователя. При входе в систему пользователи идентифицируются по именам, но после аутентификации и входа в систему, операционная система различает пользователей по их числовым идентификаторам. Этот идентификатор будет ассоциироваться с файлами и процессами, создаваемыми пользователем, и будет использоваться системой для контроля прав доступа конкретного пользователя к различным ресурсам.

GID – числовой идентификатор первичной группы пользователя. Пользователь одновременно может быть членом нескольких групп, но только одна из них, т.н. *первичная группа*, указывается в атрибутах пользователя в файле *etc/passwd*. Идентификатор первичной группы соответствует идентификатору одной из групп, перечисленных в файле *etc/group*. Формат файла *etc/group* следующий:

```
name:password:GID:пользователь,пользо
                  ватель,…
```

Здесь *name* – имя группы, *password* – необязательный зашифрованный пароль (если пароль установлен (т.е. данное поле не пустое), то при присоединении к группе командой *newgrp* пользователь должен будет ввести пароль), *GID* – числовой идентификатор группы, далее следует перечень идентификаторов членов группы, разделенных запятыми. Для исключения уязвимости, связанной с возможностью получения злоумышленником зашифрованных паролей, современные версии *UNIX* и *UNIX*-подобных систем хранят пароли в отдельном файле *etc/gshadow*, доступном только администратору.

3.2.1. Безопасное хранение паролей

По мере развития вычислительной техники, стала очевидна уязвимость хранения даже зашифрованных паролей в общедоступном файле. Поскольку алгоритм шифрования паролей известен, зная результат шифрования (хэш), можно подобрать исходный пароль, дающий тот же результат шифрования. Интересно, что подобранный пароль может и не совпадать с оригинальным паролем пользователя, но поскольку хэш пароля совпадает, этот подобранный пароль может быть использован для входа в систему.

Ситуация, когда различные исходные пароли дают один и тот же хэш, называются *коллизиями алгоритма шифрования*.

Заметим, что в 70-е годы XX века, когда было предложено хранение зашифрованных паролей в общедоступном файле, шифрация одного пароля занимала более секунды, и поиск подходящего пароля перебором полагался трудноосуществимым из-за больших временных затрат. По мере развития вычислительной техники, затраты времени на шифрацию пароля сократились в десятки тысяч раз, что заставило считаться с возможностью подбора паролей.

Одним из первых и при этом одним из наиболее известных случаев внедрения вредоносного кода с использованием свободного доступа к зашифрованным паролям является *Червь Морриса*. Червь Морриса искал пароли перебором. В качестве пароля он пробовал учетное имя пользователя, учетное имя пользователя, записанное наоборот и словарь из 400 наиболее употребительных паролей. В результате в ноябре 1988 года Червь Морриса поразил более 6000 узлов ARPANET, практически парализовав работу сети Интернет в США. После данного случая в подсистему безопасности *UNIX* внесли существенные изменения:

- исключили возможность свободного доступа к зашифрованным паролям (перенесли хэши паролей в защищенный файл, к которому имеет доступ только администратор);

– ввели таймаут (паузу) после ввода ошибочного пароля, чтобы замедлить перебор при попытке подбора пароля.

Как показывает практика (анализ случаев проникновения в систему за счет подбора паролей и специально поставленные эксперименты над базами зашифрованных паролей), подобрать пароль зачастую оказывается не очень сложно ввиду изначальной уязвимости паролей, выбираемых пользователями.

Можно выделить три основные уязвимости паролей:

– слишком короткий пароль;
– включение в пароль данных о пользователе, которые могут быть широко известны (день рождения, собственное имя, клички домашних животных, домашний адрес, имена друзей, детей и родителей, различные личные и семейные даты);
– использование в качестве пароля слова, которое может быть найдено в словаре.

Слишком короткий пароль может быть установлен полным перебором. Например, пусть в пароле 4 символа. Пусть в качестве символов пароля могут использоваться строчные и заглавные латинские буквы (26 + 26), цифры (10), строчные и заглавные русские буквы (33 + 33), специальные символы (10). Итого 138 различных символов. Тогда существует $138^4 \approx 363$ млн. вариантов пароля из 4-х символов. Хотя это число на первый взгляд и кажется большим, современный компьютер сможет выполнить перебор всех возможных вариантов пароля за несколько часов. А вот пароль из 8 символов будет иметь уже более $1.3 \cdot 10^{17}$ вариантов и на полный перебор понадобятся тысячи лет.

Подбор пароля существенно упрощается, если пользователь включает в него сведения, которые могут быть использованы как априорная информация, чтобы исключить полный перебор. Например, все возможные варианты написания даты рождения пользователя, его имени, а также различные комби-

нации этих данных образуют не более нескольких тысяч вариантов.

Многие пользователи используют в качестве пароля какое-либо слово или комбинацию нескольких слов. Словарный запас среднестатического человека составляет порядка 10 тыс. слов. С учетом их комбинации, возможности написания слов в обратном направлении, с использованием строчных и заглавных букв и т.п. можно несколько увеличить число вариантов пароля, но все равно это будет несколько миллионов вариантов, которые относительно легко перебрать.

Для того, чтобы затруднить злоумышленнику подбор пароля, необходимо соблюдать следующие простые правила:

- не использовать в качестве пароля осмысленные слова и фразы, тем более слова или фразы, имеющие отношение к конкретному пользователю;
- не использовать короткие пароли (желательно использовать пароли, по крайней мере, из 8-и символов);
- рекомендуется одновременно включать в пароль строчные, заглавные и служебные символы, а также цифры.

Для упрощения задачи выбора пароля разработано значительное число утилит генерации паролей, например, *pwg* (входит в комплект поставки некоторых версий *Linux*) или *PWGen* (существует как для *Linux*, так и для *Windows* [1]). Существуют также онлайновые генераторы паролей, доступные в сети Интернет, например, [2].

Главный недостаток автоматически сгенерированных паролей (при условии использования криптографически безопасного генератора случайных чисел.), состоит в сложности их запоминания. Обычный человек не может долгое время помнить комбинацию из десятка случайных символов, поэтому автоматически сгенерированные пароли приходится записывать. В тех случаях, когда доступ злоумышленника к записанному паролю исключен, такое решение является приемлемым. Например, пароль доступа в домашнюю сеть Wi-Fi мо-

жет храниться на листе бумаги в ящике письменного стола. Злоумышленник, не имея доступа в квартиру, не сможет воспользоваться этой записью, чтобы войти в сеть Wi-Fi с лестничной площадки или улицы. Но если пароль необходимо вводить в присутствии посторонних людей, например, для доступа к электронной почте из общественного места (кафе, общественный транспорт, зал ожидания аэропорта и т.п.), использование пароля, записанного на бумаге, уже не безопасно.

Для хранения паролей также разработаны соответствующие утилиты – *менеджеры паролей*. Некоторые из них могут запускаться со сменных носителей, на которых также хранится зашифрованная база паролей (в отличие от хранения паролей в операционных системах, здесь используется обратимое шифрование, т.е. пароль может быть расшифрован в исходный вид). Главная уязвимость менеджеров паролей состоит в том, что надежность хранения всех паролей теперь связана с единственным паролем, защищающим базу данных менеджера паролей.

Для облегчения запоминания автоматически сгенерированных паролей (чтобы не записывать их и не использовать менеджеры паролей) на правила генерации паролей можно наложить дополнительные ограничения. В частности, потребовать, чтобы пароли были бы *произносимыми*, т.е. состояли бы из последовательности слогов, свойственных человеческому языку. Такие пароли можно относительно легко запомнить, но и их стойкость по отношению к перебору ниже.

Для проверки безопасности (стойкости) паролей могут быть использованы программы, которые пытаются подобрать пароль, используя те же приемы (словари, списки распространенных паролей и др.), что и злоумышленники. Например, программа *John the Ripper* [3]).

3.2.1.1. Хранение паролей в UNIX и управление паролями

Как уже было отмечено, современные версии *UNIX* и *UNIX*-подобных систем хранят хэши паролей в отдельном файле, доступном для чтения и записи только системному администратору. Имя и внутренний формат этого файла различаются в разных системах. Рассмотрим формат файла */etc/shadow*, применяемый в *Linux*.

Каждая строка файла хранит хэш одного пароля и сопутствующую информацию в следующем формате:

```
name:password:дни_1:дни_2:дни_3:дни_4:дни_5:дни_6:резерв
```

Таким образом, запись в файле *etc/shadow* содержит девять полей, разделенных символом *':'*. Прокомментируем назначение каждого из этих полей.

`name` – учетное имя пользователя, как в файле *etc/passwd*.

`password` – хэш пароля. Формат данного поля будет рассмотрен в разделе, описывающим шифрование паролей.

`дни_1` – дата изменения пароля, хранится как количество дней, прошедших с 1 января 1970 года до момента последнего изменения пароля.

`дни_2` – количество дней после изменения пароля, в течение которых пользователь не может изменить пароль. Администратор может ограничить время действия паролей, чтобы принудить пользователей к изменению пароля (если пароль используется слишком долго, у злоумышленников растут шансы узнать его), но некоторые пользователи после изменения пароля пытаются сразу же вернуть старый пароль. Данное ограничение не позволяет им сделать это.

`дни_3` – срок действия пароля, определяется как количество дней после изменения пароля.

`дни_4` – количество дней для выдачи предупреждения о необходимости смены пароля. Данное поле определяет коли-

чество дней оставшихся до конца действия пароля, в течение которых при входе в систему будет выдаваться предупреждение о необходимости изменить пароль;

дни_5 – количество дней после истечения срока действия пароля, в течение которых пользователь может изменить пароль при входе в систему. Работа в системе со старым паролем невозможна, но при попытке входа в систему пользователю будет предложено изменить пароль, после чего он сможет продолжить работу.

дни_6 – дата блокировки учетной записи. Хранится как количество дней, прошедших с 1 января 1970 года.

Параметры срока действия пароля поясняются на следующем рис. 2.

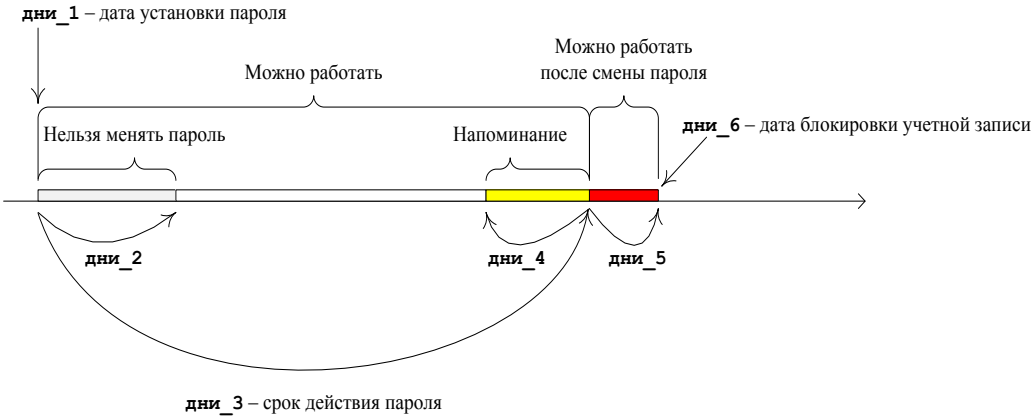


Рис. 2. Временные интервалы на сроке действия пароля

Таким образом, использование файла */etc/shadow* позволяет не только повысить безопасность хранения паролей, но и реализовать дополнительные возможности по управлению паролями, что позволяет еще больше повысить уровень безопасности в системе.

Администратор не должен вносить изменения непосредственно в файл */etc/shadow*. Для управления паролями используется команда *passwd*, имеющая следующий формат:

```
passwd [параметры] [имя_пользователя]
```

Для получения подробной информации о параметрах см. *man passwd*. Кроме того, подробное описание команды *passwd* на русском языке доступно в Интернете в рамках проекта *OpenNET* [4].

Здесь отметим только несколько параметров, полезных для понимания возможностей использования команды *passwd*.

Спецификатор *'-m'* позволяет задать количество дней после изменения пароля, в течение которых пользователь не может изменить пароль (поле *дни_2* файла */etc/shadow*). Число дней указывается после *'-m'*, ноль означает, что пользователь может изменять свой пароль в любое время.

Спецификатор *'-x'* позволяет задать срок действия пароля (поле *дни_3* файла */etc/shadow*).

Спецификатор *'-w'* позволяет задать количество дней, оставшихся до конца действия пароля, в течение которых при входе в систему будет выдаваться предупреждение о необходимости изменить пароль (поле *дни_4* файла */etc/shadow*).

Спецификатор *'-i'* позволяет задать количество дней после истечения срока действия пароля, в течение которых пользователь может изменить пароль при входе в систему (поле *дни_5* файла */etc/shadow*).

Если параметр *имя_пользователя* не указан, то команда относится к текущему пользователю.

После набора команды *passwd* с параметрами в командной строке необходимо нажать клавишу <*Enter*>, после чего в ответ на запросы системы ввести старый и новый пароли.

3.2.1.1.1. Алгоритмы шифрования паролей

Первоначальный алгоритмом шифрования (хеширования) паролей в *UNIX* был реализован *Робертом Моррисом* и *Кеном Томсоном* на основе алгоритма *DES* (*Data Encryption Standard*).

Алгоритм *DES* является блочным алгоритмом шифрования. Он оперирует с 56-битным ключом и 64-битным блоком данных. Идея использования алгоритма *DES* для хеширования паролей состоит в шифровании блока из 8 нулевых байтов с ключом, в качестве которого используется пользовательский пароль. Поскольку длина ключа в *DES* составляет 56 бит, максимальная длина пароля составляет восемь 7-битных символов (коды ASCII в диапазоне [33, 126] – !"#$%&'()*+,-./0…9:;<=>?@A...Z[\]^_`a...z{|}~).

Создатели системы хранения паролей *UNIX* опасались, что после появления микросхем с аппаратной поддержкой шифрования *DES*, подобрать пароль к хэшу можно будет очень быстро. Поэтому в стандартный алгоритм *DES* (см., например, [5]) были внесены изменения:

- цикл шифрования (битовых перестановок) DES выполняется 25 раз (без начальной и конечной перестановок), при этом входными данными для нового цикла шифрования является результат предыдущего;
- добавлен дополнительный открытый ключ, который называют *соль* (в оригинале – *salt*, в русскоязычных источниках иногда называют *привязка* или *затравка*), влияющий на таблицу битовых перестановок и на конечную перестановку алгоритма *DES*.

Размер соли при традиционном хешировании паролей в *UNIX* составляет 12 бит. Соль хранится вместе с хэшем пароля, т.е. ее можно считать известной, но благодаря использованию соли существует 4096 вариантов хеширования одного и того же пароля, в результате:

- усложняется использование заранее подготовленных хэшей наиболее употребительных паролей;

– даже если различные пользователи установят одинаковый пароль, это нельзя будет заметить по виду хэша, т.к. хэши одного пароля с разной солью будут различными;
– алгоритм хеширования уже не является алгоритмом *DES*, хотя и основан на нем, следовательно, исключается использование аппаратных ускорителей *DES*.

В *UNIX* хеширование паролей выполняет функция *crypt*() из стандартной библиотеки *POSIX* (заголовочный файл *unistd.h*). Реализация *crypt*() зависит от операционной системы. Например, реализацию в *GNU/Linux* можно просмотреть по ссылке [6].

Функция *crypt*() определена следующим образом.

```
char* crypt(const char* key, const char* salt)
```

Аргументы:

`key` – строка пароля, содержащая до 8 символов ASCII в диапазоне [33, 126];

`salt` – строка, содержащая два символа соли.

Возвращаемое значение – строка зашифрованного пароля, содержащая 13 символов, первые два символа – соль, оставшиеся 11 символов – результат шифрования блока из 64 нулевых бит с использованием модифицированного алгоритма DES с пользовательским паролем в качестве ключа.

Символы соли и возвращаемого значения представлены в 6-битной кодировке. 6-битному двоичному числу соответствует диапазон [0, 63] в десятичном представлении, который отображается на таблицу кодов ASCII следующим образом: [0, 11] → ASCII[46, 57] (./0…9); [12, 37] → ASCII[65, 90] (A…Z); [38, 63] → ASCII[97, 122] (a…z). Таким образом, в представление зашифрованного пароля могут входить строчные и заглавные латинские буквы, цифры, а также символы '.' и '/'.

Для формирования текстового представления хэша пароля, 64 бита зашифрованного блока разделяются на фрагмен-

ты по 6 бит, начиная со старших битов. Таким образом, формируется 10 6-битных символов и остается еще 4 бита. Для получения 6-битного символа они дополняются двумя младшими нулями.

В современных реализациях *UNIX* и *Linux* могут использоваться другие алгоритмы хеширования паролей, обеспечивающие совместимость с другими системами или повышающие защищенность (более длинный пароль, большая длина соли и хэша). Для того, чтобы различать способы хеширования, определен следующий формат строки зашифрованного пароля:

$***id***$*salt***$***hash*

Символ '$' не может быть получен в 6-битном представлении, следовательно, он не может встретиться в хэше на основе DES. Таким образом, если хэш начинается с символа '$', то это один из альтернативных алгоритмов хеширования, в противном случае используется традиционный алгоритм на основе DES.

За символом '$' располагается идентификатор используемого алгоритма хеширования, длина которого может быть различной. Признаком конца идентификатора является второй символ '$'.

Между вторым и третьим символами '$' располагается соль, а после третьего символа '$' идет, собственно, результат хеширования пароля.

В табл. 2 перечислены алгоритмы хеширования, используемые в настоящее время.

Таблица 2

Алгоритмы хеширования, используемые в настоящее время

Алгоритм	Идентификатор	Длина соли	Длина хэша (только поле *hash*, без символов '$', идентификатора и соли)
MD5	1	8 символов (48 бит)	22 символа (фактически, 128 бит в 6-битной кодировке)
SHA256	5	8 символов (48 бит)	43 символа (фактически, 256 бит в 6-битной кодировке)
SHA512	6	8 символов (48 бит)	86 символов (фактически, 512 бит в 6-битной кодировке)

3.2.1.1.2. Подключаемые модули аутентификации (PAM)

Подключаемые модули аутентификации (*Pluggable Authentication Modules, PAM*) – это разделяемые библиотеки, которые позволяют приложениям использовать различные низкоуровневые методы аутентификации, обычно реализуемые в виде автономных программных модулей, посредством единого высокоуровневого API [7].

Концепция PAM была предложена *Sun Microsystems* в октябре 1995 году и опубликована в документе RFC 86.0 [8].

Исторически, первой реализацией *PAM* стала *Linux-PAM*, которая вошла в дистрибутив *Red Hat Linux 3.0.4*, вышедший в августе 1996 года. В настоящее время *Linux-PAM* используется в большинстве дистрибутивов *Linux*.

Позже была предложена альтернативная реализация *PAM* с открытым исходным кодом, называнная *OpenPAM* [9]. Она была разработана под руководством Дага-Эрлинга Сморграва (Dag-Erling Smorgrav) в NAI Labs, в рамках исследова-

тельской программы DARPA-CHATS. *OpenPAM* заменила *Linux-PAM* в системе *FreeBSD*. Первым официальным релизом *FreeBSD* с *OpenPAM* стал *FreeBSD 5.1*, вышедший в июне 2003. В настоящее время *OpenPAM* используется в таких системах, как *NetBSD, FreeBSD*, а также в *MAC OS X*.

До появления *PAM*, приложения, которым была необходима аутентификация пользователей, в том числе на этапе входа в систему, должны были работать непосредственно с файлами, хранящими информацию о пользователях, группах и паролях: */etc/passwd*, */etc/group*, */etc/shadow*, */etc/gshadow* и др. Это приводило к дублированию кода, усложнению программ, проблемам совместимости при реализации альтернативных методов аутентификации, шифрования и хеширования паролей.

Использование *PAM* обеспечивает отделение реализации механизмов аутентификации от прикладных программ, за счет чего улучшается переносимость и совместимость прикладных программ, упрощается их реализация.

Общая архитектура и концепция использования *PAM* приложениями по RFC 86.0 показана на рис. 3.

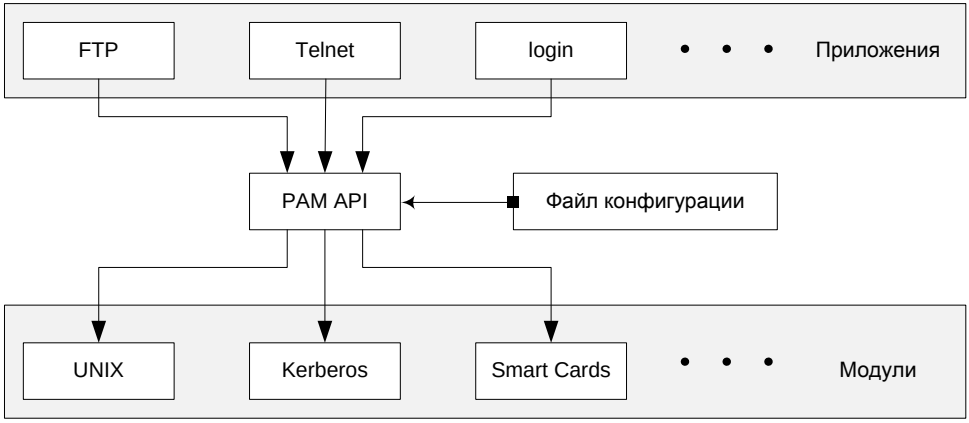


Рис. 3. Общая архитектура *PAM* по RFC 86.0

Доступ приложений к функциям PAM реализуется через библиотеку функций, прототипы которых определены в файле */usr/include/security/pam_appl.h*. Подробное описание функций библиотеки *PAM* доступно в сети Интернет по адресу [10].

Перед началом работы с *PAM* приложение должно открыть новый сеанс работы вызовом библиотечной функции *pam_start*(), определенной следующим образом:

```
int pam_start(const char* service, const
                    char* user,
          const struct pam_conv* conv,
           pam_handle_t** pam_handle)
```

Аргументы:

`service` – имя сервиса аутентификации, для которого открывается сеанс работы (обычно, в каталоге */etc/pam.d* содержится файл с таким именем);

`user` – имя пользователя, для которого требуется выполнить аутентификацию;

`conv` – указатель на экземпляр структуры **pam_conv**, содержащий адрес пользовательской функции обратного вызова, через которую *PAM* сможет вызывать пользовательское приложение, например, чтобы запросить пароль;

`pam_handle` – возвращаемое значение, указатель на идентификатор (handle) открытого сеанса, который должен использоваться в качестве аргумента при вызове других функций PAM.

При успешном открытии нового сеанса функция возвращает значение *PAM_SUCCESS*, в противном случае – код ошибки.

После завершения работы с PAM, приложение должно закрыть сеанс посредством вызова функции *pam_end*(), определенной следующим образом.

```
int pam_end(pam_handle_t* pam_handle, int
                status)
```

Аргументы:

`pam_handle` – идентификатор (handle) сеанса, полученный при вызове *pam_start*();

`status` – статус выполнения (код ошибки) последней функции *PAM*, выполненной с использованием `pam_handle`.

Для непосредственного выполнения попытки аутентификации предназначена функция *pam_authenticate*(), определенная следующим образом.

```
int pam_authenticate(pam_handle_t*
        pam_handle, int flags)
```

При вызове функции параметр `pam_handle` – идентификатор (handle) сеанса, полученный при вызове *pam_start*(), флаги см. в документации (может быть 0).

В случае успешной аутентификации функция *pam_authenticate*() возвращает *PAM_SUCCESS*, в противном случае – код, определяющий причину провала аутентификации (неизвестный пользователь, доступ к данному сервису для пользователя запрещен, неверный пароль и т.п.).

Заметим, что при выполнении процедуры аутентификации через функцию обратного вызова, указанную в параметре `conv` функции *pam_start*(), может быть запрошен пароль, который должен быть передан в открытом (незашифрованном) виде. Разработчик приложения должен принять меры к скорейшей очистке памяти, содержащей незашифрованный пароль. При вводе пароля с клавиатуры, рекомендуется отключить эхопечать вводимых символов.

Рассмотрим теперь модули *PAM*. Они делятся на четыре типа, в зависимости от выполняемой функции:

– модули аутентификации (*auth*) – выполняют аутентификацию пользователей на основе переданных в них учетных данных пользователей и признаков (*to-*

ken) аутентификации (в настоящее время в качестве признака аутентификации обычно используется пароль, но могут быть и другие признаки: электронные ключи, биометрические данные и т.п.); также модули аутентификации отвечают за предоставление приложениям специальных одноразовых разрешений (*credentials*), при этом реализация и использование одноразовых разрешений не стандартизированы и зависят от реализации модуля;

- модули управления учетными записями (*account*) – используются для дополнительной проверки учетных записей, например, на истечение срока действия или на наличие специальных ограничений для конкретного пользователя (обычно модули управления учетными записями задействуются после успешной аутентификации пользователя модулем аутентификации);
- модули управления сеансами (*session*) – используются для запуска и завершения сеанса пользователя после аутентификации (обычно модули управления сеансами вызываются в третью очередь, после модулей аутентификации и модулей управления учетными записями): выделяют память для хранения контекста пользователя, делают соответствующие записи в журналах, запускают необходимые программы и службы, например, агент SSH;
- модули управления паролями (*password*) – позволяет изменять признаки аутентификации (*token*), непосредственно в аутентификации и предоставления доступа не участвует, вызывается только при необходимости изменить пароль (или другой признак аутентификации, используемый вместо пароля), например, модуль управления паролями может вызываться, если модуль управления учетными записями возвратит код *PAM_NEW_AUTHTOKEN_REQD* –

пароль корректный, но должен быть изменен (см. раздел о файле */etc/shadow*, поле **дни_5**).

– Модули *PAM* экспортируют простой интерфейс, состоящий всего из одной-двух функций: *pam_sm_authenticate*() и *pam_sm_setcred*() у модулей аутентификации, *pam_sm_acct_mgmt*() у модулей управления учетными записями, *pam_sm_open_session*() и *pam_sm_close_session*() у модулей управления сеансами, *pam_sm_chauthtok*() у модулей управления паролями.

Данные функции прямо соответствуют функциям библиотеки *PAM*, которая просто переадресуют вызовы прикладных программ к функциям модулей.

Какие именно модули будут задействованы при обслуживании запроса приложения, определяется сценарием (конфигурацией) для сервиса аутентификации, имя которого было указано при вызове функции *pam_start*().

Согласно RFC 86.0, файл конфигурации, содержащий сценарии, должен называться */etc/pam.conf*. Каждая строка этого файла описывает один модуль *PAM*, требуемый для реализации сервиса аутентификации.

Формат строки файла конфигурации следующий:

```
имя_сервиса тип_модуля флаг файл_модуля
                [параметры]
```

Таким образом, строка конфигурации содержит пять полей, одно из которых необязательное, разделенных символами пробела или табуляции.

имя_сервиса – имя сервиса, указываемое при вызове функции *pam_start*().

тип_модуля – один из четырех типов модуля: *auth, account, session* или *password*.

флаг – определяет, как следует реагировать на ошибку (отказ) или успех при выполнении модуля. Определены следующие флаги: *requisite* (необходимый), *required* (обязатель-

ный), *sufficient* (достаточный) и *optional* (необязательный). Смысл этих флагов будет раскрыт далее.

`файл_модуля` – имя файла, содержащего модуль.

`параметры` – необязательное поле дополнительных, зависимых от реализации, параметров модуля. Все, что записано данном поле, передается в функцию модуля как аргумент командной строки. При необходимости, можно передать несколько аргументов, разделяя их пробелами.

В файле сценария для одного сервиса может быть перечислено несколько модулей одного типа. Тогда, при вызове приложением соответствующей функции *PAM*, будут вызваны все указанные модули в порядке их перечисления. Поскольку все эти модули работают независимо, они могут возвратить различные результаты проверок: часть модулей может выполниться успешно, другая часть – с ошибкой.

Для того, чтобы определить, какой результат следует передать приложению при завершении функции *PAM*, используются флаги модулей.

Функция *PAM* возвращает приложению успешный статус при выполнении любого из следующих условий:

- все обязательные и необходимые модули в списке завершились успешно;
- достаточный модуль выполнился успешно, и все предшествующие ему обязательные и необходимые модули в списке завершились успешно;
- в списке нет обязательных и необходимых модулей, ни один из достаточных модулей не завершился успешно, но хотя бы один из необязательных модулей завершился успешно.

Если один из необходимых модулей завершается с ошибкой, дальнейшее выполнение цепочки модулей прекращается и приложению немедленно возвращается код ошибки.

Если обязательный модуль возвращает ошибку, то выполнение цепочки продолжается дальше. После завершения выполнения всей цепочки (если она не прервется ошибкой необходимого модуля), приложению возвращается код ошибки

выполнения обязательного модуля. Если несколько обязательных модулей завершились с ошибкой, приложению возвращается код первой ошибки.

Таким образом, различие в поведении необходимого и обязательного модуля состоит в том, что ошибка в необходимом модуле прерывает дальнейшее выполнение цепочки, а ошибка в обязательном модуле – нет. Но в любом случае, при ошибке в необходимом или обязательном модуле приложению из функции *PAM* возвращается ошибка.

Различие в поведении достаточных и необязательных модулей состоит в том, что при успехе в достаточном модуле, если еще не было ошибок в обязательных модулях, выполнение цепочки прекращается и приложению из функции PAM возвращается успешный статус. Успех в необязательном модуле не прерывает выполнение цепочки. Но если в цепочке нет необходимых и обязательных модулей, то успех хотя бы в одном из необязательных модулей после выполнения всей цепочки приведет к успешному завершению функции *PAM*.

На рис. 4 приведен алгоритм, поясняющий выполнение цепочки модулей в зависимости от результата работы модуля и его флага.

В *Linux-PAM* и *OpenPAM* используется альтернативный подход к определению сценариев работы сервисов аутентификации. Конфигурация для каждого сервиса хранится в отдельном файле в каталоге */etc/pam.d*, при этом имя файла соответствует имени сервиса. Каждая строка такого файла описывает один модуль *PAM*, требуемый для реализации сервиса аутентификации в следующем формате.

```
тип_модуля флаг файл_модуля [параметры]
```

Такой подход позволяет упростить управление сценариями аутентификации при большом количестве сервисов.

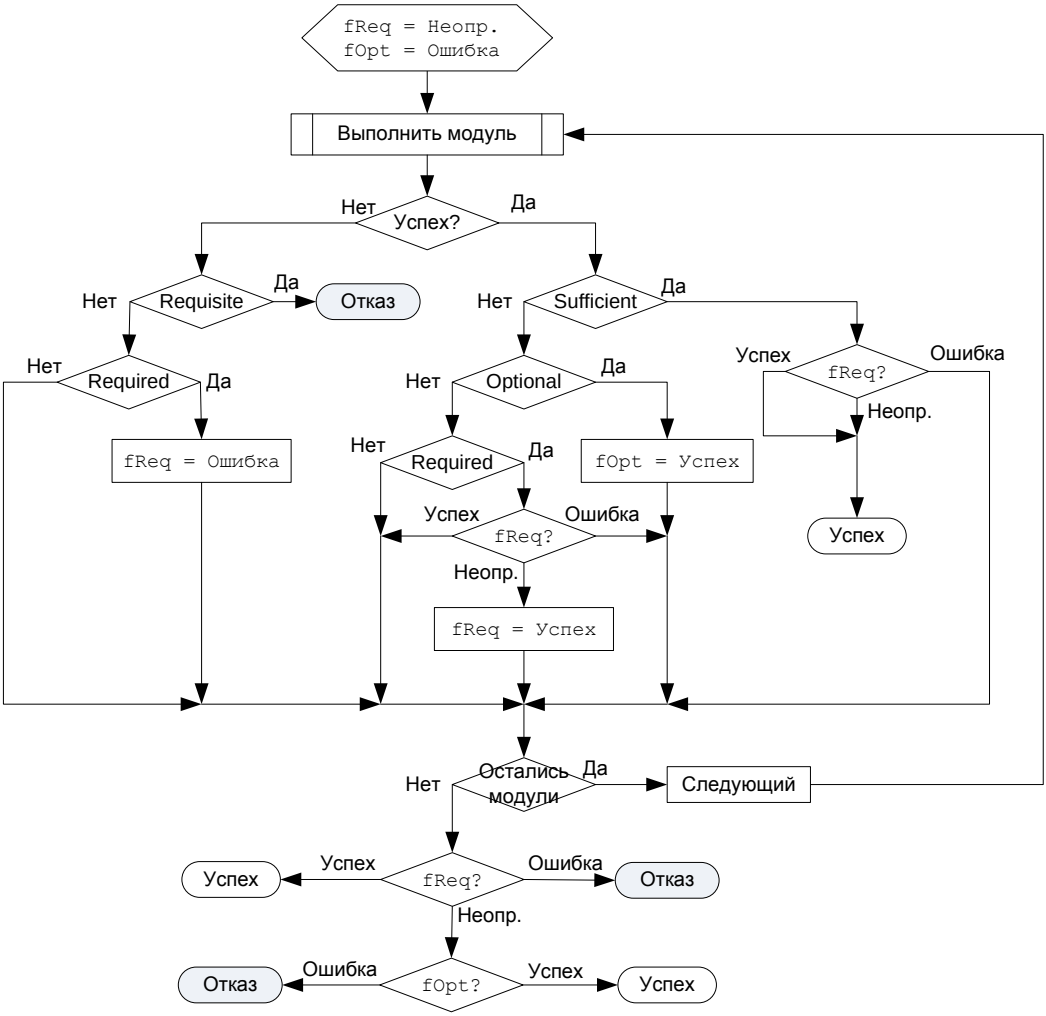


Рис. 4. Алгоритм выполнения цепочки модулей *PAM*

3.3. Псевдопользователи

Не все пользователи, перечисленные в файле */etc/passwd*, соответствуют пользователям-людям. С точки зрения операционный системы, пользователь не обязательно человек. Пользователь – это некоторый объект, который может владеть файлами, запускать программы и имеет определенные полномочия на доступ к системным ресурсам. Зарегистрированный пользователь, имеющий учетную запись в файле */etc/passwd*, но не являющийся человеком, называется *псевдопользователем*. Псевдопользователи играют существенную роль в работе операционной системы *UNIX*, от их имени запускаются многие системные сервисы, они владеют системными файлами. Так как псевдопользователь не должен входить в систему для интерактивной работы, то в файле */etc/passwd* в записях псевдопользователей вместо имени оболочки, например */bin/bash*, указывается специальная программа-заглушка, например, */bin/false* (просто завершается), или */sbin/nologon* (выводит сообщение о невозможности входа в систему и завершается).

3.4. Стандартные пользователи и группы в UNIX

Сразу после установки операционная система содержит несколько предопределенных пользователей и групп. Точный состав предопределенных пользователей и групп зависит от модификации операционной системы, поэтому перечислим несколько часто встречающихся:

- пользователи:
  - *root* – администратор системы, его UID = 0;
  - *operator* – пользователь, который может выполнять непривилегированные (для которых не требуются права администратора) операции по обслуживанию системы, например, монтировать диски на коллективной машине, выполнять резервное копирование данных и т.п.;

- пользователи для выполнения специальных действий:
  - *shutdown* – для этого пользователя в качестве оболочки установлена программа */sbin/shutdown,* таким образом, вход этого пользователя в систему приводит к завершению работы системы;
  - *halt* – для этого пользователя в качестве оболочки установлена программа */sbin/halt*, таким образом, вход этого пользователя в систему приводит к выключению машины;
  - *sync* – для этого пользователя в качестве оболочки установлена программа */bin/sync*, таким образом, вход этого пользователя в систему приводит к принудительному сбросу дискового кэша на диск;
- псевдопользователи:
  - *adm* – от его имени ведутся системные журналы, он владеет соответствующими каталогами и файлами;
  - *bin* – может владеть исполняемыми файлами, реализующих внешние команды командного интерпретатора, если ими не владеет *root*;
  - *lp* – от его имени работает система печати;
  - *daemon* – от его имени работают системные сервисы;
  - *nobody* – от его имени обычно запускают общедоступные сервисы, например *HTTP* или *FTP*;
- группы:
  - *root* – группа администраторов системы, ее GID = 0;
  - *users* – группа, в которую по-умолчанию включаются все обычные пользователи системы;
  - *wheel* – пользователи, входящие в эту группу, имеют право поднять свои привилегии до уровня *root* с помощью вызова *sudo*.

Администратор по своему усмотрению может добавлять в систему новые группы или новых пользователей, и назначать им произвольные полномочия.

3.5. Атрибуты файла SUID (Set-UID) и SGID (Set-GID)

Как мы уже говорили, с точки зрения информационной безопасности целесообразно ограничить доступ пользователей к некоторым файлам системы, например, к файлу */etc/shadow*, содержащему зашифрованные пароли. С другой стороны, некоторые программы, даже исполняемые от имени простого пользователя, должны иметь доступ к таким файлам, например, программа смены пароля */bin/passwd* должна модифицировать файл */etc/shadow*.

Выходом в данном случае является использование специальных атрибутов файла, которые называются *SUID* и *SGID*, которые при запуске программы предоставляют процессу, исполняющему программу, вместо полномочий пользователя, запустившего данную программу, полномочия пользователя или группы, владеющих файлом программы.

Например, файлом программой смены паролей */bin/passwd* владеет системный администратор. Если для файла */bin/passwd* установить атрибут *SUID*, то какой бы пользователь не запустил программу из этого файла, процесс, исполняющий программу, получит полномочия суперпользователя и сможет модифицировать файл */etc/shadow*.

По аналогии, атрибут *SGID* предписывает установить для процесса, исполняющего программу, права группы, владеющей файлом программы, а не группы пользователя, запустившего программу на исполнение.

Уязвимость такого подхода состоит в возможности получения доступа к конфиденциальным данным через дамп памяти, формируемый при аварийном завершении программы. Злоумышленник может попытаться ввести некорректные исходные данные, чтобы вызвать аварийное завершение программы, или использовать сигнал *SIGQUIT* для немедленного

завершения программы с выдачей дампа памяти. По этой причине дамп памяти для программ, работающих с расширенными полномочиями, полученными за счет использования *SUID*/*SGID*, не формируется.

3.6. Команда *sudo*

Иногда для обычного пользователя возникает необходимость выполнить действие, требующие права администратора, например, установить драйвер. В *UNIX* и *UNIX*-подобный системах есть команда *su*, позволяющая сменить пользователя (войти под именем другого пользователя), не завершая текущий сеанс. Для этого нужно казать имя пользователя, под именем которого требуется войти в систему, и его пароль. В частности, можно войти в систему под именем администратора (*root*).

Недостаток такого подхода состоит в необходимости разглашения пароля администратора для некоторых пользователей, чем может воспользоваться злоумышленник. Поэтому современные версии *UNIX* и *UNIX*-подобных систем реализуют альтернативный механизм на основе команды *sudo* (*superuser do – выполнить от имени суперпользователя*), которая не требует знания пароля администратора.

Список пользователей, имеющих право использования команды *sudo* с указанием того, что они могут выполнять, хранится в файле */etc/sudoers*. Доступ к этому файлу на запись не имеет даже системный администратор (у пользователя *root* и группы *root* есть только право на чтение, остальные пользователи не имеют доступа к файлу). Для редактирования файла */etc/sudoers* необходимо использовать специальную команду *visudo*.

Файл */etc/sudoers* представляет собой набор команд, написанных на специальном языке. Подробнее см. *man sudoers*. Кроме того, подробное описание формата команд для файла */etc/sudoers* на русском языке доступно в Интернете в рамках проекта *OpenNET* по ссылке [11].

Например:

```
root ALL = (ALL) NOPASSWD: ALL
```

разрешает пользователю *root* выполнять на любом компьютере любую команду от имени любого пользователя без ввода пароля;

```
operator ALL = DUMPS, KILL, PRINTING,
SHUTDOWN, HALT,\
REBOOT, /usr/oper/bin/
```

разрешает пользователю *operator* выполнять на любом компьютере резервное копирование, завершение процессов, управление системой печати, завершение работы системы, выключение компьютера и любую команду из каталога */usr/oper/bin*.

Если спецификатор *NOPASSWD* не установлен, то пользователь должен будет ввести свой пароль. Например, пусть в файле */etc/sudoers* есть такая запись:

```
ivan ALL = (ALL) ALL
```

т.е. пользователь *ivan* может выполнять на любом компьютере любые команды после ввода своего пароля. Например, пусть пользователь *ivan* хочет добавить в ядро модуль *mydrv.ko*. Тогда его диалог с системой будет выглядеть следующим образом:

```
ivan@localhost ~$ sudo insmod
./mydrv.ko
Password: ← ввести пароль пользователя ivan
ivan@localhost ~$
```

После введения правильного пароля пользователь в течение 5 минут может выполнять команды *sudo* без дополнительной авторизации (без ввода пароля).

4. СПРАВКА ПО ОСНОВНЫМ КОМАНДАМ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА ДЛЯ УПРАВЛЕНИЯ ПРАВАМИ ДОСТУПА В *LINUX*

4.1. Команда *su*

Команда *su* (*substitute user – заменить пользователя*) позволяет войти в систему от имени другого пользователя, не закрывая текущий сеанс. Формат команды:

```
su [-] [имя_пользователя [аргумент …]]
```

Если при вызове команды указан дефис *'-'*, то будут установлены новые параметры окружения, как будто новый пользователь вошел в систему, иначе будет сохранено старое окружение.

Если имя пользователя не указано, то подразумевается *root*.

Для работы от имени нового пользователя будет запущена оболочка, указанная в файле */etc/passwd* для соответствующего пользователя. Все аргументы, указанные после имени пользователя, будут переданы оболочке в качестве командной строки.

Для завершения работы от имени нового пользователя и возврата в свой сеанс (в свою оболочку) необходимо нажать сочетание клавиш <*Ctrl*>+<*D*>.

Все попытки (успешные и неуспешные) смены пользователя командой *su* фиксируются в журнале в файле */var/adm/sulog*.

Как уже отмечалось, основной недостаток использования команды *su* для выполнения действий от имени администратора связан с необходимостью разглашения пароля администратора, поэтому рекомендуется использовать команду *sudo*.

4.2. Команда *sudo*

Команда *sudo* (*superuser do – выполнить от имени суперпользователя*) позволяет выполнить действие, требующее дополнительных, например, административных прав. Специфика команды sudo позволяет указать для каждого пользователя конкретные действия, которые ему можно выполнять на конкретном компьютере, в том числе действия, выполняемые от имени другого пользователя. При этом пользователь не должен вводить пароли других пользователей.

Упрощенный формат команды *sudo* следующий (полную спецификацию команды *sudo* можно получить по команде *man sudo*):

```
sudo [-g имя|#gid] [-u имя|#pid] command
```

Спецификатор *'-g'* указывает, что дальше идет имя или идентификатор группы, спецификатор *'-u'* указывает, то дальше идет имя или идентификатор пользователя. Таким образом, можно указать группу или пользователя, от имени которого следует выполнять команду.

4.3. Команда *chown*

Команда *chown* (*change owner – сменить владельца*) позволяет установить нового владельца файла. Формат команды (упрощенный):

```
chown [пользователь][:группа] файл
```

Устанавливает для указанного файла указанного пользователя (или группу или пользователя и группу одновременно) в качестве владельца.

Владельца файла и права доступа к файлу можно узнать по команде *ls –l файл*.

4.4. Команда *chmod*

Команда *chmod* (*change mode – сменить режим [доступа]*) позволяет установить права доступа для файла. Формат команды (упрощенный):

```
chmod [+-] маска файл
```

Здесь маска определяет права доступа для владельца, группы-владельца и всех остальных пользователей в порядке чтение-запись-исполнение. Маска должна быть записана в восьмеричном формате. Установленный бит означает разрешение соответствующего действия, сброшенный - запрещение.

Префикс '+' или '-' может использоваться для модификации существующей маски доступа. '+' означает установить права доступа, соответствующие установленным битам маски, и не изменять права доступа, соответствующие сброшенным битам маски. '-' означает отменить права доступа, соответствующие установленным битам маски, и не изменять права доступа, соответствующие сброшенным битам маски.

Кроме того, команда *chmod* позволяет устанавливать биты SUID и SGID. Биту SUID соответствует маска (восьмеричное значение) 4000, биту SGID – 2000.

Владельца файла и права доступа к файлу можно узнать по команде *ls –l файл*.

4.5. Команда *passwd*

Команда *passwd* позволяет установить новый пароль. Для смены пароля текущего пользователя нужно просто набрать команду *passwd* без параметров. После ввода команды система запросит текущий пароль пользователя, а после его корректного ввода – новый пароль.

Также команда *passwd* позволяет управлять паролем и учетной записью пользователя: блокировать учетную запись, устанавливать время действия пароля и др. Полное описание команды см. *man passwd*.

4.6. Команда *newgrp*

Команда *newgrp* позволяет изменить текущую группу. Формат команды:

```
newgrp [-] [группа]
```

Если при вызове команды указан дефис '-', то будут установлены параметры окружения по-умолчанию, иначе будет сохранено старое окружение.

Если имя группы не указано, то подразумевается первичная группа пользователя из файла */etc/passwd*. Если пользователь пытается подключиться к группе, для которой в файле */etc/group* установлен пароль, то необходимо будет ввести пароль.

5. ЗАДАНИЯ ДЛЯ ПРАКТИЧЕСКИХ РАБОТ

1. Создать учетную пользователя/группу пользователей с различными уровнями доступа.

2. Изменить пароли в различных группах пользователей.

3. Проработать различные варианты аутентификации для приложений с учетом PAM.

4. Сделать вход в систему и завершение сеанса.

5. Изучить базовые права доступа, используя команду ls с ключом вывода расширенной информации.

6. Рассмотреть режим обычного пользователя.

7. Сделать переход в режим суперпользователя.

8. Изучить БД пользователей в –nix системах.

9. Добавить нового пользователя. Создать для данной учетной записи пароль.

10. Удалить созданную раннее учетную запись.

11. Создайте текстовый файл и задайте права на него таким образом, чтобы он мог просматриваться только владельцем и никем не мог редактироваться.

12. Найдите все исполняемые файлы с установленным suid-битом.

13. Получите имена всех пользователей системы, у которых в качестве командной оболочки используется программа /bin/false.

14. Выясните, чем отличается реакция операционной системы (выводимое сообщение) на различные ошибки аутентификации (например, неправильный пользователь, неверный пароль и т.д.).

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Дайте определение термину «разграничение доступа». Как реализуется разграничение доступа в Unix и Unix – подобных системах?

2. Как осуществляется хранение и управление паролями в Unix и Unix – подобных системах?

3. Как реализуются алгоритмы шифрования паролей в Unix и Unix – подобных системах?

4. Что представляет собой блок PAM?

5. Что представляет собой псевдопользователь? Дайте классификацию основным пользователям в Unix.

6. Для чего нужны атрибуты файла SUID (Set-UID) и SGID (Set-GID)?

7. Перечислить основные команды для Unix – подобных систем и дать характеристику.

8. Сравните права доступа к директориям /bin и /tmp. Какие операции сможет совершать в них простой пользователь?

9. Что смогут делать другие пользователями с файлами в домашней директории пользователя, если он задаст всем остальным пользователям право на запись в директорию, но удалит право исполнения на неё?

10. Чем отличаются номинальный и действительный субъект? Как они соотносятся с объектом безопасности? Что представляют собой субъект и объект безопасности в UNIX?

11. Что такое политика безопасности? Какие требования выдвигаются по отношению к ней?

12. Какие существуют наиболее распространённые схемы доступа? В чём заключаются основные отличия между ними? Какая схема доступа используется в UNIX?

13. Какие существуют права доступа в UNIX? Какие из них являются специфичными для простых файлов, а какие для директорий?

14. Что такое подмена идентификатора субъекта? Как такое право устанавливается и где применяется?

15. Опишите процесс аутентификации пользователя в UNIX.

16. Каким образом хранится информация обо всех пользователях системы?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. PWGen for Windows. Generator of cryptographically-strong password [Электронный ресурс] – Режим доступа: http://pwgen-win.sourceforge.net/.
2. Онлайн генератор паролей [Электронный ресурс] – Режим доступа: http://passwords.lance.com.ua/.
3. John the Ripper password cracker [Электронный ресурс] – Режим доступа: http://www.openwall.com/john/.
4. Интерактивная система просмотра системных руководств (man-ов) [Электронный ресурс] – Режим доступа: http://www.opennet.ru/cgi-bin/opennet/man.cgi?topic=passwd&category=1.
5. DES (Data Encryption Standard) [Электронный ресурс] – Режим доступа: http://ru.wikipedia.org/wiki/DES.
6. Glibc — GNU C Library [Электронный ресурс] – Режим доступа: http://ftp.gnu.org/gnu/libc/glibc-2.11.2.tar.bz2.
7. Pluggable Authentication Modules (PAM, подключаемые модули аутентификации) [Электронный ресурс] – Режим доступа: http://ru.wikipedia.org/wiki/Pluggable Authentication_Modules.
8. Unified login with pluggable authentication modules (PAM) [Электронный ресурс] – Режим доступа: http://www.opengroup.org/rfc/rfc86.0.html.
9. OpenPAM [Электронный ресурс] – Режим доступа: http://www.openpam.org.
10. Security. PAM library [Электронный ресурс] – Режим доступа: http://publib.boulder.ibm.com/infocenter/aix/v6r1/index.jsp?topic=%2Fcom.ibm.aix.security%2Fdoc%2Fsecurity%2Fpam_library.htm.
11. Интерактивная система просмотра системных руководств (man-ов) [Электронный ресурс] – Режим доступа: http://www.opennet.ru/man.shtml?topic=sudoers&category=5&russian=0.

СОДЕРЖАНИЕ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к практическим занятиям по дисциплинам
«Операционные системы»,
«Безопасность операционных систем»
для студентов специальностей
090301 «Компьютерная безопасность»,
090303 «Информационная безопасность
автоматизированных систем»
очной формы обучения

Составители:
Савинков Андрей Юрьевич
Савенков Григорий Анатольевич

В авторской редакции

Подписано к изданию 06.04.2015.
Уч.-изд. л. 2,7.

ФГБОУ ВПО «Воронежский государственный
технический университет»
394026 Воронеж, Московский просп., 14