Подключаемые модули аутентификации (PAM)

Аннотация

В этой статье описываются принципы и механизмы, лежащие в основе библиотеки Подключаемых Модулей Аутентификации (PAM - Pluggable Authentication Modules), и рассказывается, как настроить PAM, как интегрировать PAM в приложения и как создавать модули PAM.

Содержание

1. Введение . . . 1
2. Термины и соглашения . . . 2
3. Основы PAM . . . 5
4. Настройка PAM . . . 9
5. Модули PAM во FreeBSD . . . 12
6. Программирование приложений с PAM . . . 14
7. Программирование модуля PAM . . . 15
Приложение A: Пример PAM-приложения . . . 15
Приложение B: Пример PAM-модуля . . . 19
Приложение C: Пример функции взаимодействия PAM . . . 22
Для дальнейшего ознакомления . . . 24

1. Введение

Библиотека Pluggable Authentication Modules (PAM) является обобщённым API для служб, связанных с аутентификацией, которые позволяют системному администратору добавлять новые методы аутентификации простой установкой новых модулей PAM, и изменять политику аутентификации посредством редактирования конфигурационных файлов.

PAM описали и разработали Vipin Samar и Charlie Lai из Sun Microsystems в 1995 году, с тех он сильно не менялся. В 1997 году Open Group опубликовала предварительные спецификации на X/Open Single Sign-on (XSSO), что стандартизовало API для PAM и добавило расширения для одноразовой (или достаточно интегрированной) подписи. На момент написания этого документа эта спецификация ещё не была принята за стандарт.

Хотя эта статья посвящена в основном FreeBSD 5.x, в которой используется OpenPAM, она подойдёт для FreeBSD 4.x, использующей Linux-PAM, а также для других операционных систем, таких как Linux и Solaris™.

2. Термины и соглашения

2.1. Определения

Терминология, используемая в PAM, достаточно запутанна. Ни оригинальная работа Samar и Lai, ни спецификация XSSO не делают никаких попыток формально определить термины для различных объектов и участвующих в PAM сторон, а термины, которые они используют (но не определяют) иногда неверны и неоднозначны. Первой попыткой создать однозначную и непротиворечивую терминологию была работа, которую написал Andrew G. Morgan (автор Linux-PAM) в 1999 году. Хотя выбор терминологии, которую сделал Морган, был гигантским скачком вперёд, это, по мнению автора данной статьи, не означает её правильность. Далее делается попытка, в значительной степени на основе работы Моргана, дать точные и недвусмысленные определения терминов для всех участников и объектов PAM.

учётная запись (account)

Набор полномочий, которые аппликант запрашивает от арбитратора.

аппликант (applicant)

Пользователь или объект, запрашивающие аутентификацию.

арбитратор (arbitrator)

Пользователь или объект, имеющий привилегии, достаточные для проверки полномочий аппликанта и права подтвердить или отклонить запрос.

цепочка (chain)

Последовательность модулей, которые будут вызваны в ответ на запрос PAM. В цепочку включена информация о последовательности вызовов модулей, аргументах, которые нужно им передать, и о том, как интерпретировать результаты.

клиент (client)

Приложение, отвечающее за инициирование запроса на аутентификацию от имени аппликанта и получающее от него необходимую для аутентификации информацию.

подсистема (facility)

Одна из четырёх основных групп функциональности, которые даёт PAM: аутентификация, управление учётными записями, управление сеансом и обновление ключом аутентификации.

модуль (module)

Набор из одной или большего количества связанных функций, реализующих определённую подсистему аутентификации, собранный в один (обычно динамически загружаемый) двоичный файл, идентифицируемый по имени.

политика (policy)

Полный набор конфигурационных деклараций, описывающих, как обрабатывать запросы PAM к определённой услуге. Политика обычно состоит из четырёх цепочек, по

одной для каждой подсистемы, хотя некоторые службы используют не все четыре подсистемы.

сервер (server)

Приложение, выступающее от имени арбитратора для общения с клиентом, запрашивания аутентификационной информации, проверки полномочий аппликанта и подтверждающее или отклоняющее запрос.

сервис (service)

Класс серверов, предоставляющих похожую или связанную функциональность, и требующую подобную аутентификацию. Политики PAM задаются на основе сервисов, так что ко всем серверам, объявляющим одно и тоже имя сервиса, будет применяться одна и та же политика.

сеанс (session)

Контекст, в котором сервис оказывается аппликанту сервером. Одна из четырёх подсистем PAM, управление сеансом, касается исключительно настройке и очистке этого контекста.

ключ (token)

Блок информации, связанный с учётной записью, например, пароль или ключевая фраза, которую аппликант должен предоставить для своей идентификации.

транзакция (transaction)

Последовательность запросов от одного и того же аппликанта к одному и тому же экземпляру того же самого сервера, начиная с аутентификации и установления сеанса и заканчивая закрытием сеанса.

2.2. Примеры использования

Этот раздел предназначен для иллюстрации значений некоторых терминов, определённых выше, при помощи простых примеров.

2.2.1. Объединенные клиент и сервер

В этом простом примере показывается пользователь `alice`, выполняющий команду su(1) для того, чтобы стать пользователем `root`.

```
% whoami
alice

% ls -l `which su`
-r-sr-xr-x  1 root  wheel  10744 Dec  6 19:06 /usr/bin/su

% su -
Password: xi3kiune
# whoami
```

```
root
```

- Аппликантом является `alice`.
- Учётной записью является `root`.
- Процесс su(1) является как клиентом, так и сервером.
- Аутентификационным ключом является `xi3kiune`.
- Арбитратором выступает `root`, и именно поэтому у команды su(1) выставлен бит выполнения с правами `root`.

2.2.2. Клиент и сервер разделены

В примере ниже рассматривается пользователь `eve`, пытающийся установить ssh(1) -соединение с `login.example.com`, и успешно входя как пользователь `bob`. Боб должен был выбрать пароль получше!

```
% whoami
eve

% ssh bob@login.example.com
bob@login.example.com's password:
% god
Last login: Thu Oct 11 09:52:57 2001 from 192.168.0.1
Copyright (c) 1980, 1983, 1986, 1988, 1990, 1991, 1993, 1994
    The Regents of the University of California.  All rights reserved.
FreeBSD 4.4-STABLE (LOGIN) 4: Tue Nov 27 18:10:34 PST 2001

Welcome to FreeBSD!
%
```

- Аппликантом является `eve`.
- Клиентом является процесс ssh(1) пользователя Eve.
- Сервером является процесс sshd(8) на машине `login.example.com`
- Учётной записью является `bob`.
- Ключом аутентификации является `god`.
- Хотя этого не видно в примере, но арбитратором является `root`.

2.2.3. Пример политики

Следующее является политикой, используемой во FreeBSD по умолчанию для `sshd`:

```
sshd    auth        required    pam_nologin.so  no_warn
sshd    auth        required    pam_unix.so no_warn try_first_pass
sshd    account     required    pam_login_access.so
sshd    account     required    pam_unix.so
```

```
sshd    session     required    pam_lastlog.so  no_fail
sshd    password    required    pam_permit.so
```

- Эта политика применяется к службе `sshd` (что не обязательно ограничено сервером sshd(8)).
- `auth`, `account`, `session` и `password` являются подсистемами.
- pam_nologin.so, pam_unix.so, pam_login_access.so, pam_lastlog.so и pam_permit.so являются модулями. Из этого примера видно, что pam_unix.so реализует по крайней мере две подсистемы (аутентификацию и управление учётными записями).

3. Основы PAM

3.1. Подсистемы и примитивы

API для PAM предоставляет шесть различных примитивов для аутентификации, сгруппированных в четыре подсистемы, каждая из которых описывается ниже.

`auth`

Аутентификация. Эта подсистема, собственно говоря, реализует аутентификацию аппликанта и выяснение полномочий учётной записи. Она предоставляет два примитива:

- Функция pam_authenticate(3) аутентифицирует аппликанта, обычно запрашивая аутентификационный ключ и сравнивая его со значением, хранящимся в базе данных или получаемым от сервера аутентификации.
- Функция pam_setcred(3) устанавливает полномочия учётной записи, такие, как идентификатор пользователя, членство в группах и ограничения на использование ресурсов.

`account`

Управление учётной записью. Эта подсистема обрабатывает вопросы доступности учётной записи, не связанные с аутентификацией, такие как ограничения в доступе на основе времени суток или загрузки сервера. Он предоставляет единственный примитив:

- Функция pam_acct_mgmt(3) проверяет, доступна ли запрашиваемая учётная запись.

`session`

Управление сеансом. Эта подсистема отрабатывает задачи, связанные с установлением и закрытием сеанса, такие как учёт входов пользователей. Она предоставляет два примитива:

- Функция pam_open_session(3) выполняет действия, связанные с установлением сеанса: добавление записей в базы данных utmp и wtmp, запуск агента SSH и так далее.
- Функция pam_close_session(3) выполняет действия, связанные с закрытием сеанса: добавление записей в базы данных utmp и wtmp, завершение работы агента SSH и так

далее.

`password`

Управление паролем. Эта подсистема используется для изменения ключа аутентификации, связанного с учётной записью, по причине истечения его срока действия или желания пользователя изменить его. Она предоставляет единственный примитив:

- Функция pam_chauthtok(3) изменяет ключ аутентификации, опционально проверяя, что он труден для подбора, не использовался ранее и так далее.

3.2. Модули

Модули являются центральной концепцией в PAM; в конце концов, им соответствует буква "M" в сокращении "PAM". Модуль PAM представляет собой самодостаточный кусок программного кода, который реализует примитивы одной или большего количества подсистем одного конкретного механизма; к возможным механизмам для подсистемы аутентификации, к примеру, относятся базы данных паролей UNIX®, системы NIS, LDAP или Radius.

3.2.1. Именование модулей

Во FreeBSD каждый механизм реализуется в отдельном модуле с именем `pam_mechanism.so` (например, `pam_unix.so` для механизма UNIX®.) В других реализациях иногда отдельные модули используются для разных подсистем, и в их имя включается, кроме названия механизма, и имя подсистемы. К примеру, в Solaris™ имеется модуль `pam_dial_auth.so.1`, который часто используется для аутентификации пользователей, работающих по коммутируемым каналам связи.

3.2.2. Версии модулей

Изначальная реализация PAM во FreeBSD, которая была основана на Linux-PAM, не использовала номера версий для модулей PAM. Это будет приводить к проблемам при работе унаследованных приложений, которые могут быть скомпонованы со старыми версиями системных библиотек, так как способа подгрузить соответствующую версию требуемых модулей нет.

OpenPAM, с другой стороны, ищет модули, которые имеют тот же самый номер версии, что и библиотека PAM (на данный момент 2), и использует модуль без версии, только если модуль с известной версией не был загружен. Поэтому для старых приложений могут предоставляться старые модули, при этом новые (или заново построенные) приложения будут использовать все возможности последних версий модулей.

Хотя модули PAM в Solaris™ имеют номер версии, по-настоящему номер версии в них не отслеживается, потому что номер является частью имени и должен включаться в конфигурацию.

3.3. Цепочки и политики

Когда сервер инициирует PAM-транзакцию, библиотека PAM пытается загрузить политику для службы, указанной при вызове функции pam_start(3). Политика определяет, как должны обрабатываться запросы на аутентификацию, и задаётся в конфигурационном файле. Это составляет другую основополагающую концепцию PAM: возможность администратору настраивать политику безопасности системы (в самом широком её понимании) простым редактированием текстового файла.

Политика состоит из четырёх цепочек, по одной на каждый из методов PAM. Каждое звено представляет собой последовательность конфигурационных утверждений, задающих вызываемый модуль, некоторые (необязательные) параметры для передачи в модуль, и управляющий флаг, описывающий, как интерпретировать возвращаемый из модуля код.

Понимание смысла управляющего флага необходимо для понимания конфигурационных файлов PAM. Существуют пять различных управляющих флагов:

`binding`

Если модуль отработал успешно, и ни один из предыдущих модулей в цепочке не сработал отрицательно, то цепочка прерывается, а запрос подтверждается. Если же модуль отработает неудачно, то выполняется оставшаяся часть цепочки, однако запрос отвергается.

Этот управляющий флаг был добавлен компанией Sun в Solaris™ 9 (SunOS™ 5.9), и поддерживается в OpenPAM.

`required`

Если модуль возвратил положительный ответ, выполняется оставшаяся часть цепочки, запрос удовлетворяется, если никакой другой модуль не отработает отрицательно. Если же модуль возвратит отрицательный ответ, остаток цепочки тоже отрабатывается, но запрос отвергается.

`requisite`

Если модуль возвращает положительный ответ, выполняется оставшаяся часть цепочки, запрос удовлетворяется, если никакой другой модуль не отработает отрицательно. Если же модуль отрабатывает отрицательно, то отработка цепочки немедленно прекращается, а запрос отвергается.

`sufficient`

Если модуль возвратит положительный ответ, и ни один из предыдущих модулей в цепочке на отработал отрицательно, то отработка цепочки немедленно прекращается, а запрос удовлетворяется. Если модуль отработал отрицательно, то результат игнорируется и цепочка отрабатывается дальше.

Так как семантика этого флага может оказаться запутанной, особенно при его использовании с последним модулем в цепочке, рекомендуется вместо него использовать управляющий флаг `binding`, если реализация его поддерживает.

`optional`

Модуль отрабатывается, но результат выполнения игнорируется. Если все модули в цепочке помечены как `optional`, то удовлетворяться будут все запросы.

Когда сервер вызывает один из шести PAM-примитивов, PAM запрашивает цепочку подсистемы, к которой принадлежит примитив, и запускает каждый модуль, перечисленный в цепочке в порядке их перечисления, пока список не будет исчерпан либо не будет определено, что дальнейшей обработки не нужно (по причине достижение модуля, вернувшего положительный ответ при условии `binding` или `sufficient`, либо отрицательный с условием `requisite`). Запрос подтверждается, если только был вызван по крайней мере один модуль, и все неопциональные модули вернули положительный ответ.

Заметьте, что возможно, хотя это не распространено, перечислять один и тот же модуль несколько раз в одной цепочке. К примеру, модуль, просматривающий имена и пароли пользователя в сервере каталога может быть вызван несколько раз с различными параметрами, задающими различные серверы каталогов для связи. PAM считает различные появления одного модуля в той же самой цепочке разными и не связанными модулями.

3.4. Транзакции

Жизненный цикл типичной PAM-транзакции описан ниже. Заметьте, что в случае, если любой из перечисленных шагов оканчивается неудачно, сервер должен выдать клиенту соответствующее сообщение об ошибке и прервать транзакцию.

1. Если это необходимо, сервер получает полномочия арбитратора через независимый от PAM механизм-чаще всего по факту запуска пользователем `root` или с установленным setuid-битом `root`.
2. Сервер вызывает функцию pam_start(3) для инициализации библиотеки PAM и задания имени сервиса и целевой учётной записи, а также регистрации подходящего способа общения.
3. Сервер получает различную информацию, относящуюся к транзакции (такую, как имя пользователя аппликанта и имя хоста, на котором запущен клиент), и отправляет её в PAM при помощи функции pam_set_item(3).
4. Сервер вызывает функцию pam_authenticate(3) для аутентификации аппликанта.
5. Сервер вызывает функцию pam_acct_mgmt(3) для проверки того, что запрошенная учётная запись доступна и корректна. Если пароль верен, но его срок истёк, pam_acct_mgmt(3) возвратит результат `PAM_NEW_AUTHTOK_REQD`, а не `PAM_SUCCESS`.
6. Если на предыдущем шаге был получен результат `PAM_NEW_AUTHTOK_REQD`, то сервер вызывает функцию pam_chauthtok(3) для того, чтобы вынудить клиента изменить ключ аутентификации для запрошенной учётной записи.
7. Теперь, когда аппликант полностью аутентифицирован, сервер вызывает функцию pam_setcred(3) для получения полномочий запрошенной учётной записи. Сделать это возможно, потому что он работает как арбитратор, и оставляет за собой полномочия арбитратора.
8. После получения необходимых полномочий, сервер вызывает функцию

pam_open_session(3) для установления сеанса.

9. Теперь сервер выполняет тот сервис, который затребовал клиент-например, предоставляет аппликанту оболочку.
10. После того, как сервер закончил обслуживание клиента, он вызывает функцию pam_close_session(3) для закрытия сеанса.
11. Наконец, сервер вызывает функцию pam_end(3) для оповещения библиотеки PAM о том, что работа с ней завершена и какие-либо выделённые в течение сеанса ресурсы можно освободить.

4. Настройка PAM

4.1. Файлы политик PAM

4.1.1. Файл /etc/pam.conf

Традиционно файлом политик PAM является /etc/pam.conf. Он содержит все политики PAM для вашей системы. Каждая строка файла описывает один шаг в цепочке, как показано ниже:

```
login   auth    required        pam_nologin.so  no_warn
```

Поля следуют в таком порядке: имя службы, имя подсистемы, управляющий флаг, имя модуля и параметры модуля. Любые дополнительные поля интерпретируются как дополнительные параметры модуля.

Для каждой пары сервис/подсистема составляется отдельная цепочка, и тогда получается, что, хотя порядок следования строк для одной и той же услуги и подсистемы является значимым, порядок перечисления отдельных сервисов не значим. В примерах из оригинальной работы по PAM строки конфигурации сгруппированы по подсистемам, в поставляемом с Solaris™ файле pam.conf именно так и сделано, но в стандартном конфигурационном файле из поставки FreeBSD строки настроек сгруппированы по сервисам. Подходит любой из этих способов; они имеют один и тот же смысл.

4.1.2. Каталог /etc/pam.d

OpenPAM и Linux-PAM поддерживают альтернативный механизм настройки, который для FreeBSD является предпочтительным. В этой схеме каждая политика содержится в отдельном файле с именем, соответствующем сервису, к которому она применяется. Эти файлы размещаются в каталоге /etc/pam.d/.

Такие файлы политик, ориентированные на сервисы, имеют только четыре поля, вместо пяти полей в файле pam.conf: поле имени сервиса опущено. Таким образом, вместо примера строки файла pam.conf из предыдущего раздела получится следующая строка в файле /etc/pam.d/login:

```
auth    required        pam_nologin.so  no_warn
```

Как следствие такого упрощённого синтаксиса, возможно использование одних и тех же политик для нескольких сервисов, связывая каждое имя сервиса с тем же самым файлом политик. К примеру, для использования той же самой политики для сервисов `su` и `sudo`, можно сделать следующее:

```
# cd /etc/pam.d
# ln -s su sudo
```

Это работает, потому что имя сервиса определяется именем файла, а не его указанием в файле политики, так что один и тот же файл может использоваться для нескольких сервисов с разными названиями.

Так как политика каждого сервиса хранится в отдельном файле, то механизм pam.d делает установку дополнительных политик для программных пакетов сторонних разработчиков очень лёгкой задачей.

4.1.3. Порядок поиска политик

Как вы видели выше, политики PAM могут находиться в нескольких местах. Что будет, если политики для одного и того же сервиса имеются в разных местах?

Необходимо осознать, что система конфигурации PAM ориентирована на цепочки.

4.2. Структура строки настройки

Как это объяснено в Файлы политик PAM, каждая строка файла /etc/pam.conf состоит из четырёх или большего количества полей: имени сервиса, имени подсистемы, управляющего флага, имени модуля и дополнительных параметров модуля, которые могут отсутствовать.

Имя сервиса обычно (хотя не всегда) является именем приложения, которое этот сервис обслуживает. Если вы не уверены, обратитесь к документации по конкретному приложению для определения используемого имени сервиса.

Заметьте, что если вы используете /etc/pam.d/ вместо /etc/pam.conf, то имя сервиса задается именем файла политики, и опускается из строк настройки, которые в таком случае начинаются с названия подсистемы.

Имя подсистемы представляет собой одно из четырёх ключевых слов, описанных в Подсистемы и примитивы.

Точно также управляющий флаг является одним из четырёх ключевых слов, описанных в разделе Цепочки и политики, в котором рассказано, как интерпретировать возвращаемый из модуля код. В Linux-PAM поддерживается альтернативный синтаксис, который позволяет указать действие, связанной с каждый возможным кодом возврата, но этого следует

избегать, так как он не является стандартным и тесно связан со способом диспетчеризации вызовов сервисов в Linux-PAM (а он значительно отличается от способа взаимодействия в Solaris™ и OpenPAM). Не вызывает удивления тот факт, что в OpenPAM этот синтаксис не поддерживается.

4.3. Политики

Для корректной настройки PAM необходимо понимать, как происходит интерпретация политик.

В момент, когда приложение вызывает функцию pam_start(3), библиотека PAM загружает политику для указанного сервиса и выстраивает четыре цепочки модулей (по одной для каждой подсистемы). Если одна или большее количество этих цепочек являются пустыми, то будут выполняться подстановки соответствующих цепочек из политики для сервиса `other`.

Когда затем приложение вызывает одну из шести примитивов PAM, библиотека PAM выделяет из цепочки нужную подсистему и вызывает функцию, соответствующую сервису, в каждом модуле, перечисленном в цепочке, в том порядке, в каком они перечислены в конфигурации. После каждого обращения к функции сервиса, тип модуля и возвращённый из этой функции код результата выполнения используются для того, что делать дальше. За некоторыми исключениями, которые будут описаны ниже, применяется такая таблица:

Таблица 1. Сводная таблица отработки цепочек PAM

	PAM_SUCCESS	**PAM_IGNORE**	**other**
binding	if (!fail) break;	-	fail = true;
required	-	-	fail = true;
requisite	-	-	fail = true; break;
sufficient	if (!fail) break;	-	-
optional	-	-	-

Если переменная `fail` принимает истинное значение в конце отработки цепочки, или когда достигнут "break", диспетчер возвращает код ошибки, возвращённый первым модулем, отработавшим неудачно. В противном случае возвращается `PAM_SUCCESS`.

Первым исключением является то, что код ошибки `PAM_NEW_AUTHTOK_REQD` интерпретируется как успешный результат, кроме случая, когда модуль отработал успешно, и по крайней мере один модуль возвратил `PAM_NEW_AUTHTOK_REQD`, тогда диспетчер возвратит результат `PAM_NEW_AUTHTOK_REQD`.

Вторым исключением является то, что pam_setcred(3) считает, что модули `binding` и `sufficient` являются равнозначными `required`.

Третьим и последним исключением является то, что функция pam_chauthtok(3) отрабатывает полную цепочку дважды (один раз для предварительных проверок, и ещё раз для реального задания пароля), и на подготовительной фазе она считает, что модули `binding`

и `sufficient` являются равнозначными `required`.

5. Модули PAM во FreeBSD

5.1. pam_deny(8)

Модуль pam_deny(8) является одним из простейших доступных модулей; на любой запрос он возвращает результат `PAM_AUTH_ERR`. Он полезен для быстрого отключения сервиса (добавьте его на верх каждой цепочки) или завершения цепочек модулей `sufficient`.

5.2. pam_echo(8)

Модуль pam_echo(8) просто передаёт свои параметры в функцию взаимодействия как сообщение `PAM_TEXT_INFO`. В основном полезна для отладки, но также может использоваться для вывода сообщений, таких как "Unauthorized access will be prosecuted" до запуска процедуры аутентификации.

5.3. pam_exec(8)

Модуль pam_exec(8) воспринимает первый переданный ему параметр как имя программы для выполнения, а остальные аргументы передаются этой программе в качестве параметров командной строки. Одним из возможных применений является его использование для запуска в момент регистрации в системе программы монтирования домашнего каталога пользователя.

5.4. pam_ftpusers(8)

Модуль pam_ftpusers(8)

5.5. pam_group(8)

Модуль pam_group(8) принимает или отвергает аппликантов в зависимости от их членства в определённой файловой группе (обычно `wheel` для su(1)). В первую очередь предназначен для сохранения традиционного поведения утилиты BSD su(1), хотя имеет и много других применений, таких как отключение определённых групп пользователей от некоторого сервиса.

5.6. pam_guest(8)

Модуль pam_guest(8) позволяет осуществлять гостевые входы с использованием фиксированных имён входа в систему. На пароль могут накладываться различные ограничения, однако действием по умолчанию является ввод любого пароля при использовании имени, соответствующего гостевому входу. Модуль pam_guest(8) можно легко использовать для реализации анонимных входов на FTP.

5.7. pam_krb5(8)

Модуль pam_krb5(8)

5.8. pam_ksu(8)

Модуль pam_ksu(8)

5.9. pam_lastlog(8)

Модуль pam_lastlog(8)

5.10. pam_login_access(8)

Модуль pam_login_access(8) предоставляет реализацию примитива для управления учётными записями, который вводит в действие ограничения на вход, задаваемые в таблице login.access(5).

5.11. pam_nologin(8)

Модуль pam_nologin(8) отвергает любые входы не пользователем root, если существует файл /var/run/nologin. Обычно этот файл создаётся утилитой shutdown(8), когда до запланированного завершения работы системы остаётся менее пяти минут.

5.12. pam_passwdqc(8)

Модуль pam_passwdqc(8)

5.13. pam_permit(8)

Модуль pam_permit(8) является одним из самых простым из имеющихся; на любой запрос он отвечает `PAM_SUCCESS`. Он полезен в качестве замены пустого места для сервисов, когда одна или большее количество цепочек в противном случае останутся пустыми.

5.14. pam_radius(8)

Модуль pam_radius(8)

5.15. pam_rhosts(8)

Модуль pam_rhosts(8)

5.16. pam_rootok(8)

Модуль pam_rootok(8) возвращает положительный результат в том и только в том случае, если реальный id пользователя процесса, его вызвавшего (предполагается, что его запускает аппликант) равен 0. Это полезно для несетевых сервисов, таких как su(1) или passwd(1), к которым пользователь `root` должен иметь автоматический доступ.

5.17. pam_securetty(8)

Модуль pam_securetty(8)

5.18. pam_self(8)

Модуль pam_self(8) возвращает положительный результат тогда и только тогда, когда имена аппликанта соответствуют целевой учётной записи. Больше всего это пригодится в несетевых сервисах, таких как su(1), в которых идентификация аппликанта может быть с лёгкостью проверена.

5.19. pam_ssh(8)

Модуль pam_ssh(8) предоставляет как сервис аутентификации, так и сеанса. Сервис аутентификации позволяет пользователям, имеющим секретные ключи SSH, защищённые паролями, в своих каталогах ~/.ssh, аутентифицироваться посредством этих паролей. Сеансовый сервис запускает ssh-agent(1) и загружает ключи, которые были расшифрованы на фазе аутентификации. Такая возможность, в частности, полезна для локальных входов в систему, как в систему X (посредством xdm(1) или другого X-менеджера входов, умеющего работать с PAM), так и на консоль.

5.20. pam_tacplus(8)

Модуль pam_tacplus(8)

5.21. pam_unix(8)

Модуль pam_unix(8) реализует традиционную аутентификацию UNIX® на основе паролей, использующую функцию getpwnam(3) для получения пароля целевой учётной записи и сравнивающую её с тем, что представил аппликант. Он также предоставляет средства управления учётными записями (отслеживая время действия учётной записи и пароля) и смены паролей. Наверное, это самый полезный модуль, так как подавляющее большинство администраторов хотят сохранить исторически сложившееся поведение по крайней мере некоторых сервисов.

6. Программирование приложений с PAM

Этот раздел ещё не написан.

7. Программирование модуля PAM

Этот раздел ещё не написан.

Приложение A: Пример PAM-приложения

Далее следует минимальная реализация программы su(1) с использованием PAM. Заметьте, что в ней используется специфичная для OpenPAM функция взаимодействия openpam_ttyconv(3), объявление которой расположено в файле security/openpam.h. Если вы собираетесь строить это приложение в системе с другой библиотекой PAM, вам необходимо будет создать собственную функцию взаимодействия. Надёжную функцию взаимодействия неожиданно трудно написать; та, что находится в Пример функции взаимодействия PAM, хороша в качестве отправной точки, но в реальных приложениях использоваться не может.

```
/*-
 * Copyright (c) 2002,2003 Networks Associates Technology, Inc.
 * All rights reserved.
 *
 * This software was developed for the FreeBSD Project by ThinkSec AS and
 * Network Associates Laboratories, the Security Research Division of
 * Network Associates, Inc.  under DARPA/SPAWAR contract N66001-01-C-8035
 * ("CBOSS"), as part of the DARPA CHATS research program.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. The name of the author may not be used to endorse or promote
 *    products derived from this software without specific prior written
 *    permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * $P4: //depot/projects/openpam/bin/su/su.c#10 $
```

```
 * $FreeBSD: head/en_US.ISO8859-1/articles/pam/su.c 38826 2012-05-17 19:12:14Z hrs $
 */

#include <sys/param.h>
#include <sys/wait.h>

#include <err.h>
#include <pwd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <syslog.h>
#include <unistd.h>

#include <security/pam_appl.h>
#include <security/openpam.h> /* for openpam_ttyconv() */

extern char **environ;

static pam_handle_t *pamh;
static struct pam_conv pamc;

static void
usage(void)
{

    fprintf(stderr, "Usage: su [login [args]]\n");
    exit(1);
}

int
main(int argc, char *argv[])
{
    char hostname[MAXHOSTNAMELEN];
    const char *user, *tty;
    char **args, **pam_envlist, **pam_env;
    struct passwd *pwd;
    int o, pam_err, status;
    pid_t pid;

    while ((o = getopt(argc, argv, "h")) != -1)
        switch (o) {
        case 'h':
        default:
            usage();
        }

    argc -= optind;
    argv += optind;

    if (argc > 0) {
```

```
        user = *argv;
        --argc;
        ++argv;
    } else {
        user = "root";
    }

    /* initialize PAM */
    pamc.conv = &openpam_ttyconv;
    pam_start("su", user, &pamc, &pamh);

    /* set some items */
    gethostname(hostname, sizeof(hostname));
    if ((pam_err = pam_set_item(pamh, PAM_RHOST, hostname)) != PAM_SUCCESS)
        goto pamerr;
    user = getlogin();
    if ((pam_err = pam_set_item(pamh, PAM_RUSER, user)) != PAM_SUCCESS)
        goto pamerr;
    tty = ttyname(STDERR_FILENO);
    if ((pam_err = pam_set_item(pamh, PAM_TTY, tty)) != PAM_SUCCESS)
        goto pamerr;

    /* authenticate the applicant */
    if ((pam_err = pam_authenticate(pamh, 0)) != PAM_SUCCESS)
        goto pamerr;
    if ((pam_err = pam_acct_mgmt(pamh, 0)) == PAM_NEW_AUTHTOK_REQD)
        pam_err = pam_chauthtok(pamh, PAM_CHANGE_EXPIRED_AUTHTOK);
    if (pam_err != PAM_SUCCESS)
        goto pamerr;

    /* establish the requested credentials */
    if ((pam_err = pam_setcred(pamh, PAM_ESTABLISH_CRED)) != PAM_SUCCESS)
        goto pamerr;

    /* authentication succeeded; open a session */
    if ((pam_err = pam_open_session(pamh, 0)) != PAM_SUCCESS)
        goto pamerr;

    /* get mapped user name; PAM may have changed it */
    pam_err = pam_get_item(pamh, PAM_USER, (const void **)&user);
    if (pam_err != PAM_SUCCESS || (pwd = getpwnam(user)) == NULL)
        goto pamerr;

    /* export PAM environment */
    if ((pam_envlist = pam_getenvlist(pamh)) != NULL) {
        for (pam_env = pam_envlist; *pam_env != NULL; ++pam_env) {
            putenv(*pam_env);
            free(*pam_env);
        }
        free(pam_envlist);
    }
```

```
    /* build argument list */
    if ((args = calloc(argc + 2, sizeof *args)) == NULL) {
        warn("calloc()");
        goto err;
    }
    *args = pwd->pw_shell;
    memcpy(args + 1, argv, argc * sizeof *args);

    /* fork and exec */
    switch ((pid = fork())) {
    case -1:
        warn("fork()");
        goto err;
    case 0:
        /* child: give up privs and start a shell */

        /* set uid and groups */
        if (initgroups(pwd->pw_name, pwd->pw_gid) == -1) {
            warn("initgroups()");
            _exit(1);
        }
        if (setgid(pwd->pw_gid) == -1) {
            warn("setgid()");
            _exit(1);
        }
        if (setuid(pwd->pw_uid) == -1) {
            warn("setuid()");
            _exit(1);
        }
        execve(*args, args, environ);
        warn("execve()");
        _exit(1);
    default:
        /* parent: wait for child to exit */
        waitpid(pid, &status, 0);

        /* close the session and release PAM resources */
        pam_err = pam_close_session(pamh, 0);
        pam_end(pamh, pam_err);

        exit(WEXITSTATUS(status));
    }

pamerr:
    fprintf(stderr, "Sorry\n");
err:
    pam_end(pamh, pam_err);
    exit(1);
}
```

Приложение B: Пример PAM-модуля

Далее приведена минимальная реализация pam_unix(8), предоставляющая только сервисы аутентификации. Она должна строиться и работать с большинством из реализаций PAM, но использует возможности расширений OpenPAM, если они присутствуют: отметьте использование функции pam_get_authtok(3), которая кардинально упрощает организацию ввода пароля пользователем.

```
/*-
 * Copyright (c) 2002 Networks Associates Technology, Inc.
 * All rights reserved.
 *
 * This software was developed for the FreeBSD Project by ThinkSec AS and
 * Network Associates Laboratories, the Security Research Division of
 * Network Associates, Inc.  under DARPA/SPAWAR contract N66001-01-C-8035
 * ("CBOSS"), as part of the DARPA CHATS research program.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. The name of the author may not be used to endorse or promote
 *    products derived from this software without specific prior written
 *    permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * $P4: //depot/projects/openpam/modules/pam_unix/pam_unix.c#3 $
 * $FreeBSD: head/en_US.ISO8859-1/articles/pam/pam_unix.c 38826 2012-05-17 19:12:14Z
hrs $
 */

#include <sys/param.h>

#include <pwd.h>
```

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#include <security/pam_modules.h>
#include <security/pam_appl.h>

#ifndef _OPENPAM
static char password_prompt[] = "Password:";
#endif

#ifndef PAM_EXTERN
#define PAM_EXTERN
#endif

PAM_EXTERN int
pam_sm_authenticate(pam_handle_t *pamh, int flags,
    int argc, const char *argv[])
{
#ifndef _OPENPAM
    struct pam_conv *conv;
    struct pam_message msg;
    const struct pam_message *msgp;
    struct pam_response *resp;
#endif
    struct passwd *pwd;
    const char *user;
    char *crypt_password, *password;
    int pam_err, retry;

    /* identify user */
    if ((pam_err = pam_get_user(pamh, &user, NULL)) != PAM_SUCCESS)
        return (pam_err);
    if ((pwd = getpwnam(user)) == NULL)
        return (PAM_USER_UNKNOWN);

    /* get password */
#ifndef _OPENPAM
    pam_err = pam_get_item(pamh, PAM_CONV, (const void **)&conv);
    if (pam_err != PAM_SUCCESS)
        return (PAM_SYSTEM_ERR);
    msg.msg_style = PAM_PROMPT_ECHO_OFF;
    msg.msg = password_prompt;
    msgp = &msg;
#endif
    for (retry = 0; retry < 3; ++retry) {
#ifdef _OPENPAM
        pam_err = pam_get_authtok(pamh, PAM_AUTHTOK,
            (const char **)&password, NULL);
#else
```

```
        resp = NULL;
        pam_err = (*conv->conv)(1, &msgp, &resp, conv->appdata_ptr);
        if (resp != NULL) {
            if (pam_err == PAM_SUCCESS)
                password = resp->resp;
            else
                free(resp->resp);
            free(resp);
        }
#endif
        if (pam_err == PAM_SUCCESS)
            break;
    }
    if (pam_err == PAM_CONV_ERR)
        return (pam_err);
    if (pam_err != PAM_SUCCESS)
        return (PAM_AUTH_ERR);

    /* compare passwords */
    if ((!pwd->pw_passwd[0] && (flags & PAM_DISALLOW_NULL_AUTHTOK)) ||
        (crypt_password = crypt(password, pwd->pw_passwd)) == NULL ||
        strcmp(crypt_password, pwd->pw_passwd) != 0)
        pam_err = PAM_AUTH_ERR;
    else
        pam_err = PAM_SUCCESS;
#ifndef _OPENPAM
    free(password);
#endif
    return (pam_err);
}

PAM_EXTERN int
pam_sm_setcred(pam_handle_t *pamh, int flags,
    int argc, const char *argv[])
{

    return (PAM_SUCCESS);
}

PAM_EXTERN int
pam_sm_acct_mgmt(pam_handle_t *pamh, int flags,
    int argc, const char *argv[])
{

    return (PAM_SUCCESS);
}

PAM_EXTERN int
pam_sm_open_session(pam_handle_t *pamh, int flags,
    int argc, const char *argv[])
{
```

```
    return (PAM_SUCCESS);
}

PAM_EXTERN int
pam_sm_close_session(pam_handle_t *pamh, int flags,
    int argc, const char *argv[])
{

    return (PAM_SUCCESS);
}

PAM_EXTERN int
pam_sm_chauthtok(pam_handle_t *pamh, int flags,
    int argc, const char *argv[])
{

    return (PAM_SERVICE_ERR);
}

#ifdef PAM_MODULE_ENTRY
PAM_MODULE_ENTRY("pam_unix");
#endif
```

Приложение C: Пример функции взаимодействия PAM

Функция взаимодействия, приводимая ниже, является значительно упрощённой версией функции openpam_ttyconv(3) из OpenPAM. Она полнофункциональна, и должна послужить источником идей о том, как должна себя вести функция взаимодействия, однако она слишком проста для реальных приложений. Даже если вы не используете OpenPAM, можете сгрузить исходный код и использовать openpam_ttyconv(3) в своих целях; мы надеемся, что она достаточно надёжна в качестве функции для взаимодействия с терминальными устройствами.

```
/*-
 * Copyright (c) 2002 Networks Associates Technology, Inc.
 * All rights reserved.
 *
 * This software was developed for the FreeBSD Project by ThinkSec AS and
 * Network Associates Laboratories, the Security Research Division of
 * Network Associates, Inc.  under DARPA/SPAWAR contract N66001-01-C-8035
 * ("CBOSS"), as part of the DARPA CHATS research program.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
```

```
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. The name of the author may not be used to endorse or promote
 *    products derived from this software without specific prior written
 *    permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * $FreeBSD: head/en_US.ISO8859-1/articles/pam/converse.c 38826 2012-05-17 19:12:14Z
hrs $
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include <security/pam_appl.h>

int
converse(int n, const struct pam_message **msg,
    struct pam_response **resp, void *data)
{
    struct pam_response *aresp;
    char buf[PAM_MAX_RESP_SIZE];
    int i;

    data = data;
    if (n <= 0 || n > PAM_MAX_NUM_MSG)
        return (PAM_CONV_ERR);
    if ((aresp = calloc(n, sizeof *aresp)) == NULL)
        return (PAM_BUF_ERR);
    for (i = 0; i < n; ++i) {
        aresp[i].resp_retcode = 0;
        aresp[i].resp = NULL;
        switch (msg[i]->msg_style) {
        case PAM_PROMPT_ECHO_OFF:
            aresp[i].resp = strdup(getpass(msg[i]->msg));
```

```
            if (aresp[i].resp == NULL)
                goto fail;
            break;
        case PAM_PROMPT_ECHO_ON:
            fputs(msg[i]->msg, stderr);
            if (fgets(buf, sizeof buf, stdin) == NULL)
                goto fail;
            aresp[i].resp = strdup(buf);
            if (aresp[i].resp == NULL)
                goto fail;
            break;
        case PAM_ERROR_MSG:
            fputs(msg[i]->msg, stderr);
            if (strlen(msg[i]->msg) > 0 &&
                msg[i]->msg[strlen(msg[i]->msg) - 1] != '\n')
                fputc('\n', stderr);
            break;
        case PAM_TEXT_INFO:
            fputs(msg[i]->msg, stdout);
            if (strlen(msg[i]->msg) > 0 &&
                msg[i]->msg[strlen(msg[i]->msg) - 1] != '\n')
                fputc('\n', stdout);
            break;
        default:
            goto fail;
        }
    }
    *resp = aresp;
    return (PAM_SUCCESS);
 fail:
        for (i = 0; i < n; ++i) {
                if (aresp[i].resp != NULL) {
                        memset(aresp[i].resp, 0, strlen(aresp[i].resp));
                        free(aresp[i].resp);
                }
        }
        memset(aresp, 0, n * sizeof *aresp);
    *resp = NULL;
    return (PAM_CONV_ERR);
}
```

Для дальнейшего ознакомления

Статьи

Making Login Services Independent of Authentication Technologies Vipin Samar. Charlie Lai. Sun Microsystems.

X/Open Single Sign-on Preliminary Specification. The Open Group. 1-85912-144-6. June 1997.

Pluggable Authentication Modules. Andrew G. Morgan. 1999-10-06.

Руководства пользователя

Администрирование PAM. Sun Microsystems.

Веб страницы

Домащняя страница OpenPAM Dag-Erling Smørgrav. ThinkSec AS.

Домашняя страница Linux-PAM Andrew Morgan.

Домашняя страница Solaris PAM (Solaris PAM homepage). Sun Microsystems.