Pluggable Authentication Modules

Антон Комаревцев

24 апреля 2004 г.

1 Введение.

В процессе разработки операционных систем в один прекрасный момент возникла задача авторизации пользователей, поскольку потребовалось хранить и обрабатывать «секретные» сведения. Как бы мы стали решать эту задачу? Скорее всего, для первого раза создали бы файл, в который записали бы пары

```
имя пользователя - секретный ключ
```

(так уж повелось, что обычно присутствует имя пользователя, которое не является секретом), а при входе пользователя в систему спрашивали бы у пользователя имя и пароль и искали соответствующую пару в нашем файле. Если пара найдена — допускаем пользователя к работе, если не найдена — отказываем в доступе. Так, безусловно, и было сделано на первом этапе. Потом подход пришлось усовершенствовать по двум причинам и в двух соответствующих направлениях:

1. Взломщики всегда были на шаг впереди разработчиков защиты и находили способ взломать любой новый новой механизм обеспечения безопасности.

2. Авторизация проникла глубже в систему. Потребовалось осуществлять авторизацию не только пользователей при входе, но и процессы, удаленные системы, а также пользователей в процессе работы.

В первом случае появлялись новые hash-функции для паролей, новые системы шифрования, shadow-пароли, *Trusted Computing Base (TCB)* [5] и т.д. Во втором случае функции для работы с файлами паролей были вынесены в отдельные библиотеки, а интерфейс их взаимодействия с программами (в первую очередь с программой `login`) стандартизован.

Например, в OpenBSD до настоящего времени вся авторизация основана на *BSD Authentication system* [1].

Но с начала «эпохи» прошло уже достаточно много лет, и пользователи операционных систем стали гораздо более жадными. Появлялись все новые требования к системе авторизации, для удовлетворения которых требовалась большая гибкость этой системы. В результате был создан механизм авторизации, основанный на *Pluggable Authentication Modules (PAM)* [4].

2 Для чего были созданы PAM.

Изначально программа `login` была ориентирована на работу с локальным файлом `/etc/passwd`, но постоянные изменения системы хранения паролей заставили разработчиков операционных систем задуматься над отделением программы входа в систему от механизма хранения паролей. Кроме того, необходимость в этом возникла с появлением различных систем авторизации пользователей (PKI, Kerberos) и различных программ входа в систему (login, xdm).

Вскоре стало ясно, что такое разделение позволило бы избавиться от переписывания модулей авторизации кроме `login` в таких программах, как `rlogin`, `telnet`, `ftp`.

Более того, с появлением различных систем авторизации появилась необходимость одному пользователю иметь разные пароли в разных системах авторизации, что требовало помнить их все, а также команды для авторизации в этих системах.

Кроме этого, системные администраторы хотели бы иметь возможность изменять условия входа в систему, системы авторизации и проч. не только без изменения кода основных программ, но даже и без перекомпиляции их. Разумеется, во времена создания языков, названных буквами латинского алфавита, о таком никто и не мечтал, но в средине девяностых годов это стало насущной проблемой.

В погоне за новыми доходами, стараясь удовлетворить большее число заказчиков, фирма Sun принялась за создание конфигурируемой системы для своих ОС. Разработчики PAM, по их словам, старались достичь следующих целей:

1. Необходимо дать системному администратору возможность самому выбирать механизм авторизации для входа в систему, будь то обычная проверка пароля, проверка биометрических параметров тела пользователя, или авторизация по smart-карте.

2. Этот механизм может различаться для различных методов входа в систему, например, для локального входа через `login` или удаленного входа с помощью `telnet`.

3. Необходимо учесть возможность передачи параметров авторизации от пользователя к системе проверки этих параметров на подлинность различными средствами. Например, это может быть обычная консоль, или графический режим, или удаленный ftp-клиент.

4. Существует возможность задать несколько систем опознания пользователя для одного приложения. Например, можно потребовать от пользователя при входе в систему пройти через обычный login и через Kerberos.

5. Однако необходимо иметь возможность упростить такой вход в систему, если есть желание: не требовать от пользователя вводить пароли ко всем системам, а передавать их другим системам авторизации автоматически.

6. Библиотеки, дающие или отвергающие вход в систему, должны обеспечивать интерфейс для вышележащих приложений, не зависящий от нижележащей системы авторизации, что позволило бы не переписывать прикладные программы при изменении системы авторизации.

7. Такой механизм должен иметь встраиваемые части, которые администратор мог бы добавлять или убирать на собственное усмотрение.

8. Механизм должен обеспечивать совместимость с уже написанными прикладными программами, а его API — быть платформенно-независимым.

Разработчики PAM переложили задачу создания конкретного интерфейса, предоставляемого приложеним, на OpenGroup [6]. Кроме того, разработанная ими система PAM даже не пытается решить задачу передачи по сети идентификационной информации и паролей удаленного клиента (rlogin, ftp).

3 Архитектура PAM.

Как я уже упоминал, существуют с одной строны приложения, а с другой — динамически встраиваемые модули. Самое большее, что мы можем сделать, это стандартизировать их интерфейс к библиотеке PAM. Возникает задача связывания этих модулей. Для ее разрешения библиотека PAM имеет две точки подключения:

- PAM API — интерфейс для взаимодействия с приложениями (telnet, ftp),
- PAM SPI (Service Provider Interface) — интерфейс для взаимодействия с модулями.

Между ними находится конфигурируемая библиотека PAM.

Выделяются четыре стандартных типа модулей [4, 8]:

1. *Authentication.* Такие модули предназначены для принятия решения о том, может ли пользователь получить доступ к запрашиваемому ресурсу или нет в смысле процесса, называемого authentication.

2. *Account.* Эти модули используются для управлением аккаунтом пользователя. Например, они могут определять возможность входа в систему в зависимости от времени суток.

3. *Session.* Модули предназначены для обеспечения существования сессии пользователя: ведение журналов, запуск и остановка необходимых сервисов и проч.

4. *Password.* Модули предназначены для управления паролем. В частности, для его изменения и проверки на «слабость».

Как уже говорилось, среди прочих возникают две проблемы, которые требуют одновременного использования нескольких модулей:

- Поддержка нескольких видов проверки при входе в систему (например, RSA, Kerberos),
- Уменьшение числа паролей, необходимых для прохождения модулей разных типов.

Было принято решение, что несколько модулей в PAM будут выстраиваться в стек и проходиться один за другим. С точки зрения логики передачи управления от модуля к модулю фирма Sun в настоящее время выделяет следующие типы модулей:

1. *Required.* Для успешного выполнения цепочки требуется успешное выполнение этого модуля. Но если он завершился с ошибкой, цепочка продолжает выполнятся, хотя гарантированно также завершится с ошибкой.
2. *Requisite.* Как и в предыдущем случае, для успешного выполнения цепочки требуется удачное прохождение данного модуля. Если же он завершился с ошибкой, цепочка сразу прерывается.
3. *Sufficient.* Если данный модуль завершается успешно, авторизация также успешно заканчивается. В противном случае выполнение цепочки продолжается и отрицательный результат данного модуля не принимается во внимание.
4. *Optional.* Результат выполнения не влияет на результат всей цепочки.
5. *Binding.* Если модуль завершается успешно, цепочка сразу же успешно завершается. Если модуль завершается неудачно, выполнение цепочки продолжается, но результат всей цепочки неудачный.

Последний модуль был добавлен недавно, в версии Sun Solaris 9. Результат выполнения такого модуля определяет во многих случаях результат выполнения цепочки целиком.

4 Использование PAM. Точка зрения администратора.

При настройке PAM для каждого приложения описывается очередь из модулей. Конфигурационные файлы могут различаться для разных реализаций PAM, но везде для каждого приложения в нужном порядке указываются имена модулей, их типы (авторизации, проверки паролей и т.д.), правила участия в очереди и параметры.

Ниже приведена часть файла настроек (`pam.conf`) для Solaris:

```
# -------  -----------  ------------  -----------       -------
# SERVICE  MODULE TYPE  CONTROL FLAG  MODULE PATH       OPTIONS
# -------  -----------  ------------  -----------       -------

  login    auth         required      pam_kerb_auth.so  debug
  login    auth         required      pam_unix_auth.so  use_mapped_pass
  login    auth         required      pam_rsa_auth.so   try_first_pass
```

С учетом сказанного в предыдущем разделе, думаю, нет необходимости пояснять правила построения конфигурационного файла.

В Linux вместо одного файла настройки используется каталог `/etc/pam.d`, куда кладутся файлы настройки, по одному на каждый сервис, использующий PAM. Например, содержимое файла `login` для стандартного сервиса входа в систему может быть таким:

```
#%PAM-1.0
auth     required       pam_securetty.so
auth     include        system-auth
auth     required       pam_nologin.so
auth     optional       pam_mail.so
account  include        system-auth
password include        system-auth
session  include        system-auth
session  optional       pam_lastlog.so nowtmp
session  optional       pam_motd.so
session  optional       pam_console.so
```

Здесь за ненадобностью отсутствует колонка с именем сервиса, которое вынесено теперь в имя файла. Значения остальных колонок — такие же, как и в первом примере.

5 Использование PAM. Взгляд разработчика.

Я не хочу переписывать документацию по PAM для Linux [3], прочитать ее и использовать не составляет никакого труда. Думаю, будет гораздо больше пользы, если я рассмотрю небольшой пример работающего PAM-модуля. Вот его текст:

// pam_keys_test.c

```
#include <security/pam_modules.h>
#include <stdarg.h>
#include <stdio.h>
#include <time.h>
```

```
#define PAM_SM_AUTH
#define MAX_V 30

PAM_EXTERN int pam_sm_authenticate(pam_handle_t * pamh, int flags,
int argc, const char **argv)
{
  char *arg_file_name, *arg_file_path;

  unsigned int ctrl;
  int retval;
  const char *name, *p;

  struct pam_conv *conv;
  struct pam_message *pmsg[3],msg[3];
  struct pam_response *response;

  if (argc != 1)
    return PAM_AUTH_ERR;
  arg_file_name = (char *) strtok(argv[0], "=");
  arg_file_path = (char *) strtok(0, "=");
  if ((arg_file_name == 0) || (arg_file_path == 0))
    return PAM_AUTH_ERR;

  retval = pam_get_user(pamh, &name, "login: ");
  retval = pam_get_item(pamh, PAM_CONV, (const void **) &conv) ;

  pmsg[0] = &msg[0];
  msg[0].msg_style = PAM_PROMPT_ECHO_OFF;
  msg[0].msg = "Key:";

  retval = conv->conv(1, (const struct pam_message **) pmsg, &response,
conv->appdata_ptr);
  if (find_key(response->resp, arg_file_path)) return PAM_SUCCESS;

  return PAM_AUTH_ERR;
}


PAM_EXTERN int pam_sm_setcred(pam_handle_t * pamh, int flags, int
argc, const char **argv)
{
  unsigned int ctrl;
  int retval;
```

```
  retval = PAM_SUCCESS;

  return retval;
}

int find_key(char *key, char *file_name)
{
  FILE *f;
  int b_found = 0;
  char fk[18];

  if ((f = fopen(file_name, "r")) == 0) return 0;
  while (!b_found && fgets(fk, 17, f)) b_found = !strcmp(strcat(key, "\n"), fk);
  fclose(f);
  return b_found;
}

#ifdef PAM_STATIC
struct pam_module _pam_unix_auth_modstruct = {
  "pam_keys_test",
  pam_sm_authenticate,
  pam_sm_setcred,
  NULL,
  NULL,
  NULL,
  NULL,
};
#endif
```

Макроопределение `#define PAM_SM_AUTH` задает тип модуля (auth).

В модуле такого типа существуют две стандартные функции, которые необходимо определить: `pam_sm_setcred` и `pam_sm_authenticate`. Эти функции будут вызваны библиотекой PAM при обращении к модулю. Их объявления стандартны, а реализации определяют специфику работы данного пользовательского модуля. Первая предназначена для установки пользовательских групп и в нашем примере не используется, а вторая возвращает значение, которое принимается как успешное или неудачное прохождение авторизации данного модуля. Решение об успехе всей цепочки библиотека libpam принимает сама.

Данный пример модуля запрашивает у пользователя имя и некоторый ключ, а затем проверяет его наличие в специальном файле, имя которого передается модулю в качестве параметра. Для упрощения кода никакого шифрования ключей не производится. Таким образом, мы осуществляем авторизацию пользователя на основе некоего подобия паролей, хранящихся в файле на диске.

Компилируется модуль командой

```
cc -o pam_keys_test.so -shared pam_keys_test.c
```

Затем его необходимо скопировать в каталог модулей (обычно `/lib/security`) и дописать соответствующую строчку в конфигурационный файл соответствующего сервиса. Я добавлял ее в файл `/etc/pam.d/login`:

```
auth required pam_keys_test.so pfile=/tmp/keys
```

В файл `/tmp/keys`, переданный модулю в качестве параметра, небходимо дописать какие-нибудь ключи, по одному в каждой строке.

Если теперь попробовать войти с любой консоли или через telnet, то мы увидим, что после имени пользователя появился запрос на ключ. Вводим в качестве ключа один из тех, которые мы напечатали в файле `/tmp/keys` (в исходном тексте видно, что был использован специальный флаг, выключающий echo), затем вводим обычный пароль и попадаем в систему.

Я рекомендую читателям поэкспериментировать с параметрами настройки модуля, чтобы лучше понять назначение различных типов участия модуля в очереди.

Гораздо более подробное описание другого примера модуля, а также пример использования модуля PAM в приложени, можно найти в [7].

6 JAAS как пример реализации PAM.

Ярким примером использования PAM служит система JAAS — Java Authentication and Authorization Service. Ее подробное описание может быть найдено в [2]. В настоящее время практически любая проверка пользователя в Java осуществляется с помощью JAAS.

Необходимо четко понимать разницу между значениями слов Authentication и Authorization.

- *Authentication* — идентификация пользователя, то есть гарантированно верное определение, кто выполняет код,
- *Authorization* — проверка, имеет ли данный пользователь права на выполнение данного действия.

JAAS позволяет производить и то, и другое.

Как и в случае с Linux-PAM, я не стану переписывать [2], а лишь поясню общую идею использования JAAS. Последовательность действий может быть такой:

1. Создаем собственный модуль PAM, наследуя его класс от **LoginModule.** Логика работы нашего модуля описывается в переопределяемых методах, таких, как **login, logout.**
2. Создаем собственный principal, который будет одним из представлений объекта. В простейшем случае это имя объекта. Класс для нашего principal необходимо наследовать от **Principal.**

3. Создаем собственный callback handler, который будет запрашивать необходимую информацию у пользователя. В простейшем случае это может быть имя пользователя и пароль в консоли, в более сложном случае — вывод графического диалогового окна или запрос к базе данных. В классе, наследованном от **CallbackHandler**, необходимо переопределить метод **handle**, в который передается массив из объектов класса **Callback**. Эти объекты служат для передачи в модуль полученных от пользователя имени, пароля и другой информации.

4. Создаем конфигурационный файл для нашего приложения, перечисляем в нем используемые модули, их участие в очереди и передаваемые параметры. Ниже приведено содержимое конфигурационного файла из примера Authentication Tutorial:

```
Sample {
   sample.module.SampleLoginModule required debug=true;
};
```

5. В основном приложении, где и необходимо пройти authentication, осуществляем следующие действия:

   (a) Создаем объект класса LoginContext:
   ```
   LoginContext lc = new LoginContext(‘‘Sample’’,
       new MyCallbackHandler());
   ```
   Первый параметр — имя контекста в конфигурационном файле.

   (b) Входим в систему:
   ```
   lc.login();
   ```

   (c) Выполняем необходимые действия, в том числе можем получить `Subject` — полное представление вошедшего в систему пользователя (subject содержит в себе все необходимые principals и cridentials), выполнить действия от имени других пользователей.

   (d) Выходим и системы:
   ```
   lc.logout();
   ```

Главное отличие JAAS от Linux-PAM и PAM в Solaris связано со спецификой языка Java: вместо динамических библиотек в качестве модулей здесь используются классы Java.

Прекрасно разобранный пример для первого знакомства с JAAS можно найти в Authentication Tutorial, расположенном по адресу

`http://java.sun.com/j2se/1.4.2/docs/guide/security/jaas/tutorials/GeneralAcnOnly.html`.

7 Новые возможности PAM.

Из новых возможностей, которые были недоступны в полной мере до появления PAM, в первую очередь хочу отметить предоставляемый системой PAM в Solaris *password-mapping.* Как уже было сказано в разделе 3, очень часто возникает проблема запоминания нескольких паролей к разным системам авторизации для одного пользователя. Конечно, можно поставить один пароль на все сервисы, но это значительно ослабит безопасность системы.

Разработчики PAM предложили компромиссное решение: выбирается один сильный пароль, с помощью которого зашифровываются остальные. У пользователя спрашивают, как правило, только первый пароль, а остальные получаются в результате расшифровки. Если расшифрованный пароль, подставленный системой автоматически, по каким-то причинам не подошел, модуль может запросить пароль для себя у пользователя, или может завершиться с ошибкой. Все это настраивается с помощью параметров модулей в файле конфигурации PAM.

Вторая очень полезная возможность PAM — легкое добавление авторизации на основе SmartCard. Это новая технология хранения паролей на специальных пластиковых картах.

1. На карту можно записать достаточно длинный пароль, поэтому их использование решает проблему «слабых» паролей.

2. Кроме того, для использования чужой smart-карты помимо самой карты нужен еще и PIN-код, который владелец карты никому не должен сообщать.

3. В такие карты может быть встроена система шифрования и шифрования, поэтому голый пароль не будет доступен операционной системе и программам злоумышленника, который хотел бы его заполучить.

Теперь для использования smart-карты достаточно написать собственный модуль (или использовать готовый) и встроить его в PAM.

8 Выводы.

Как и любая другая система, PAM имеет свои плюсы и минусы. Я выделил бы следующие преимущества PAM перед другими системами авторизации:

- Функциональность PAM находится во включаемых модулях, поэтому система легко переконфигурируется под новые нужды системным администратором.

- Возможность добавлять и удалять модули позволяет использовать новые средства авторизации, не затрагивая всю систему обеспечения безопасности операционной системы.

- Наличие общепринятого стандарта на PAM позволяет, поняв один раз логику его работы, быстро изучать конкретные его реализации.
- За счет организации списка модулей PAM упрощает прохождение авторизации в нескольких службах, в том числе сохраняя зашифрованные пароли, и значительно расширяет возможности по созданию гибкой и наиболее приемлемой для конкретных условий и пользователей системы прохождения авторизации.

Как и следовало ожидать, все минусы связаны с ослаблением безопасности:

- Система PAM достаточно сложна, поэтому трудно гарантировать, что нет ошибок в конкретной реализации libpam.
- Если присмотреться внимательнее, задача построения действительно безопасного и в том же время удобного стека модулей также очень сложна, поэтому трудно гарантировать, что администратор не допустил ошибок при конфигурировании libpam.
- Использование password-mapping, хотя и является удобным для пользователя средством, все же значительно ослабляет безопасность. Действительно, если каким-то образом первичный пароль оказался недостаточно надежным и был узнан или подобран, остальные пароли сразу станут известны злоумышленнику.
- Реализация PAM часто бывает интегрирована с другой достаточно большой и запутанной системой, что вызывает новые ошибки. Например, авторизация через JAAS во многих web-серверах содержит неоправданно много ошибок и недоработок.

В связи с вышесказанным необходимо сформулировать общие рекомендации по использованию или неиспользованию PAM. Если система, которую Вы проектируете, обладает значительной сложностью, Вам и без того придется поплатиться в некоторой мере ее надежностью, поэтому некоторые слабости в безопасности, вносимые PAM, не станут серьезной проблемой, зато сильно сократят время на разработку и упростят поддержание системы в дальнейшем.

С другой стороны, если Вы пишете на каком-то простом языке, например, на C, относительно небольшую и обозримую по сложности систему, и уделяете значительное внимание ее безопасности, использование PAM, по моему мнению, не оправдано, потому что их сложность испортит красоту простоты системы и заметно увеличит риск ослабления ее безопасности.

9 Acknowledgments.

В первую очередь хочется поблагодарить Rajaram за идею PAM, а также разработчиков из Sun Microsystems, Inc, создавших PAM для Sun Solaris, особенно Vipin Samar (`vipin@eng.sun.com`) и Charlie Lai (`charlie@eng.sun.com`).

Хочу также выразить благодарность Станиславу Иевлеву (inger@altlinux.ru), занимающемуся обеспечением безопасности в операционных системах компании ALT Linux (www.altlinux.ru), который в статье [7] очень доступно преподносит начальные сведения о Linux-PAM.

Особая благодарность разработчикам операционной системы OpenBSD, Theo de Raadt (deraadt@cvs.openbsd.org) и Александру Юрченко (grange@rt.mipt.ru), любезно отвечавшим на мои вопросы, касающиеся обеспечения безопасности в данной операционной системе.

Список литературы

1. BSD Authentication system — `man auth_open`
2. JAAS Reference Guide — `http://java.sun.com/j2se/1.4.2/docs/guide/security/jaas/JAASRefGuide.html`
3. Linux-PAM — `http://www.kernel.org/pub/linux/libs/pam/`
4. Pluggable Authentication Modules — `http://java.sun.com/security/jaas/doc/pam.html`
5. Trusted Computer Base — `www.openwall.com/tcb/`
6. X/Open Single Sign-On Service (XSSO) — Pluggable Authentication — `http://www.opengroup.org/pubs/catalog/p702.htm`
7. Начала PAM — `http://www.citforum.ru/operating_systems/articles/pam.shtml`
8. Реализация PAM в FreeBSD — `http://www.freebsd.org/doc/ru_RU.KOI8-R/articles/pam/index.html`