1. Министерство науки и высшего образования Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. **Институт кибербезопасности и защиты информации**

**ЛАБОРАТОРНАЯ РАБОТА № 6**

1. «Аутентификация в ОС Linux»
2. по дисциплине «Безопасность операционных систем»
3. Выполнил
4. студент гр. 4851003/70801 Гасанов Э.А.

<*подпись*>

1. Преподаватель
2. ассистент Ахтямов Д.Р.

<*подпись*>

1. Санкт-Петербург
2. 2020

**Цель**

Изучение подсистемы аутентификации в ОС Linux.

**Задачи**

1. Изучить принципы работы подключаемых модулей аутентификации в ОС Linux.
2. Реализовать собственный модуль аутентификации и использовать его для входа в систему.
3. Добавить аутентификацию пользователя с помощью разработанного модуля в собственное или стороннее приложение, использую интерфейс PAM.

**Ход работы**

Модули PAM накладываются друг на друга в конфигурационном файле /etc/pam.d/… Это реализует гибкость, заложенную в PAM – каждый модуль исполняет своё одно задание.

Поэтому стеком в PAM называется список модулей, последовательно заданных в конфигурационном файле. На последних строках рисунка видна последовательная запись модулей.

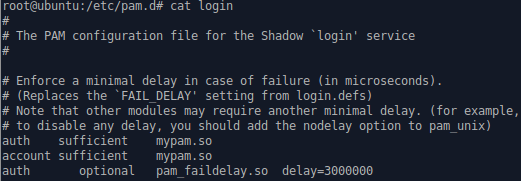


Рисунок 1 – Стек модулей PAM.

Для того чтобы записать в нужный конфигурационный файл (/etc/pam.d/…) собственный модуль необходимо записать его в стандартном виде:

|  |
| --- |
| Тип модуля Контрольные флаги Путь к модулю Аргументы |

**Типы модулей PAM**

Модули PAM совершают не только аутентификацию, но и аккаунтинг, и сеансовый менеджмент, и смену токенов аутентификации. Поэтому существует группировка по типам модулей:

* Аккаунт менеджмент определяет имеет ли пользователь доступ к системе.
* Аутентификация устанавливает действительно ли пользователь тот, за кого он себя выдаёт. Тут же решается вопрос о принадлежности членства к группе пользователей и установка полномочий.
* Пароль менеджмент – возможность смены аутентифицирующего токена (пароля).
* Сеанс. Выполняет задачи, выполняемые, когда пользователь запускает или завершает сеанс, например, отображение последнего входа в систему или монтирование каталогов.

**Контрольные флаги**

* *required*

Должен выполниться, иначе последующие модули будут запущены, но стек вернет ошибку.

* *requisite*

Должен выполниться, иначе ни один другой модуль не запуститься и сразу же вернется ошибка первого провалившегося модуля.

* *sufficient*

Если модуль с этим флагом выполняется успешно, то последующие модули не выполняются и возвращается PAM\_SUCCESS. Но если проваливается, то стек выполняется дальше.

* *optional*

Не влияет на успех или провал.

**Путь к модулю**

Здесь необходимо выставить путь к разделяемой библиотеке. В данной лабораторной работе мы использовали эту директорию, куда сохраняли свою разделяемую библиотеку:

|  |
| --- |
| /lib/x86\_64-linux-gnu/security/ |

**Аргументы**

Необязательны, поэтому назовём некоторые из них

* *debug*

Отправляет отладочную информацию с помощью syslog ()

* *no\_warn*

Не отсылать предупреждения

**Собственный модуль аутентификации для входа в систему**

Далее был создан собственный модуль для аутентификации в систему. Для того чтобы теперь аутентифицироваться, необходимо ответить на вопрос, который спрашивает сумму текущего часа с днём месяца. Для вывода этого вопроса был использован механизм conversation. Проверочное значение также вычисляется в модуле. Оба значения (введенное пользователем и проверочное) записывается в массив с помощью функции snprintf и проверяется побайтово в strcmp. При полном совпадении возвращается PAM\_SUCCESS, а по определению модуль с флагом sufficient при первом PAM\_SUCCESS больше не исполняет стек с модулями и позволяет пользователю войти.

Для того чтобы скомпилировать модуль необходимо использовать команды:

|  |
| --- |
| gcc -fPIC -fno-stack-protector -c src/mypam.c  sudo ld -x --shared -o /lib/x86\_64-linux-gnu/security/mypam.so mypam.o |

Далее в директории /etc/pam.d/login добавляем строчки с определением собственного модуля в соответствии с правилами записи в конфигурационный файл. Они должны располагаться на вершине стека. Мы пишем имя разделяемой библиотеки без пути, так как она уже находится в /lib/x86\_64-linux-gnu/security/



Рисунок 2 – Модуль на вершине стека.

Проверим доступ:

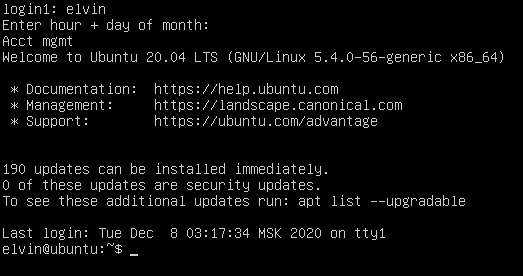


Рисунок 3 – Пароль для входа 3(время в часах в формате 24) +8(число месяца).

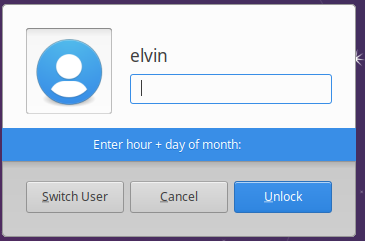


Рисунок 4 – С графическим интерфейсом – аналогично.

**Добавление аутентификации пользователя с помощью созданного модуля в собственное приложение**

Добавим эту же разделяемую библиотеку по пути /etc/pam.d/common-auth:

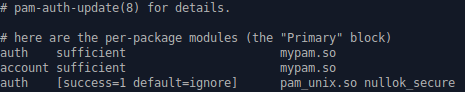


Рисунок 5 – Добавление библиотеки common-auth.

Далее создадим приложение, где с помощью интерфейсных функций PAM (pam\_start, pam\_authenticate, pam\_acct\_mgmt, pam\_end) будем обращаться к уже созданному модулю.

Опишем создаваемое приложение.

Оно рассчитывает месячные выплаты (m) и суммарную выплату (s) по кредиту. О кредите известно, что он составляет n рублей, берется на y лет, под p процентов. Месячные выплаты находятся по формуле:

|  |
| --- |
| m = (n \* p \* (1 + p)y) / (12 \* ((1 + p)y – 1)) |

где p выражается в долях единицы, а не процентах.

Суммарная выплата представляет собой выплаты за все месяцы каждого года:

|  |
| --- |
| s = (m \* 12) \* y |

“Обернём” это функциями интерфейса PAM и скомпилируем:

|  |
| --- |
| g++ -o pam\_credit src/credit.c -lpam -lpam\_misc -lm |

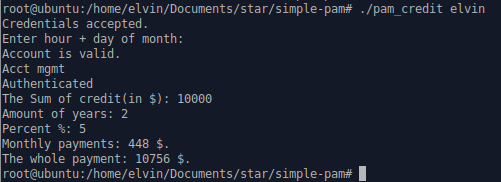


Рисунок 6 – Аутентификация перед использованием банковского приложения для подсчёта кредита. Пароль для входа 23=15(время в часах) + 8 (день месяца).

**Вывод**

В данной лабораторной работе были изучены теоретические основы PAM, такие как разбиение модулей на группы, стекирвание, флаги. А также способ применения модулей в системе для изменения порядка аутентификации и способ применения модулей уже в собственных приложениях с помощью интерфейса.

**Листинг**

**Модуль**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include<time.h>

#include <security/pam\_appl.h>

#include <security/pam\_modules.h>

#define PAM\_SM\_AUTH

#define MAX\_V 30

/\* expected hook \*/

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) {

printf("Acct mgmt\n");

return PAM\_SUCCESS;

}

/\* expected hook, this is where custom stuff happens \*/

PAM\_EXTERN int pam\_sm\_authenticate(pam\_handle\_t \* pamh, int flags

,int argc, const char \*\*argv)

{

unsigned int ctrl;

int retval;

const char \*name, \*p;

char \*right;

/\*special variables\*/

long x1,x2,x3,x4,y;

time\_t mytime;

struct tm \*mytm;

/\*готовимся к аутентификации\*/

mytime=time(0);

mytm=localtime(&mytime);

srandom(mytime);

x1=random()%MAX\_V;

x2=random()%MAX\_V;

x3=random()%MAX\_V;

x4=random()%MAX\_V;

//завели несколько случайных величин, а заодно узнали и время.

/\* получим имя пользователя \*/

// Получаем имя пользователя

// Вся мудрость PAM в том, что приглашение "login: " появится если имяеще не известно,

// иначе мы сразу получим ответ, сгенерированный предыдущими модулями.

retval = pam\_get\_user(pamh, &name, "login1: ");

/\*получим пароль используя диалог\*/

{

struct pam\_conv \*conv;

struct pam\_message \*pmsg[3],msg[3];

struct pam\_response \*response;

retval = pam\_get\_item( pamh, PAM\_CONV, (const void \*\*) &conv ) ;

// Сами мы не знаем как будет осущестляться диалог, это забота программы(в нашем случае этим займется login). Мы

// лишь только укажем параметры, вид приглашения и более того, можемзадать сразу несколько приглашений, если надо

// получить сразу несколько ответов

pmsg[0] = &msg[0];

msg[0].msg\_style = PAM\_PROMPT\_ECHO\_OFF;

msg[0].msg=malloc(100);

//snprintf(msg[0].msg,60,"Second Password:%d:%d:%d:%d:",x1,x2,x3,x4);

snprintf(msg[0].msg,99,"Enter hour + day of month:");

retval = conv->conv(1, ( const struct pam\_message \*\* ) pmsg

, &response, conv->appdata\_ptr);

// Нам дали указатель на диалоговую функцию. ╢е и запускаем.

/\*просчитаем правильный ответ\*/

//y=2\*x1\*mytm->tm\_mday+x3\*mytm->tm\_hour;

y=mytm->tm\_mday+mytm->tm\_hour;

right=malloc(100);

snprintf(right,20,"%ld",y);

/\*memset(&msg[0],0,60);

pmsg[0] = &msg[0];

msg[0].msg\_style = PAM\_PROMPT\_ECHO\_OFF;

msg[0].msg=malloc(100);

snprintf(msg[0].msg,60,"\nAnswer:%s",right);

retval = conv->conv(1, ( const struct pam\_message \*\* ) pmsg

, &response, conv->appdata\_ptr);\*/

// Сравним с ответом пользователя. Ответ формируется диалоговой функцией в специальном формате.

if (!(strcmp(right,response->resp))){

//if ((strcmp(right,response->resp))){

return PAM\_SUCCESS;

}else{

return PAM\_AUTH\_ERR;

}

}/\*диалог\*/

return PAM\_SUCCESS;

// Нашим результатом будет да или нет. Как прервать программу разберется основной модуль PAM.

}

**Приложение с модулем**

#include <security/pam\_appl.h>

#include <security/pam\_misc.h>

#include <stdio.h>

#include<math.h>

const struct pam\_conv conv = {

misc\_conv,

NULL

};

void credit(void);

int main(int argc, char \*argv[]) {

pam\_handle\_t\* pamh = NULL;

int retval;

const char\* user = "nobody";

if(argc != 2) {

printf("Usage: app [username]\n");

exit(1);

}

user = argv[1];

retval = pam\_start("check\_user", user, &conv, &pamh);

// Are the credentials correct?

if (retval == PAM\_SUCCESS) {

printf("Credentials accepted.\n");

retval = pam\_authenticate(pamh, 0);

}

// Can the accound be used at this time?

if (retval == PAM\_SUCCESS) {

printf("Account is valid.\n");

retval = pam\_acct\_mgmt(pamh, 0);

}

// Did everything work?

if (retval == PAM\_SUCCESS) {

printf("Authenticated\n");

} else {

printf("Not Authenticated\n");

}

credit();

// close PAM (end session)

if (pam\_end(pamh, retval) != PAM\_SUCCESS) {

pamh = NULL;

printf("check\_user: failed to release authenticator\n");

exit(1);

}

return retval == PAM\_SUCCESS ? 0 : 1;

}

void credit(void)

{

int n,y;

float p,m,s;

printf("The Sum of credit(in $): ");

scanf("%d", &n);

printf("Amount of years: ");

scanf("%d", &y);

printf("Percent %%: ");

scanf("%f", &p);

p = p / 100;

m = (n \* p \* pow((1+p),y)) / (12 \* (pow(1+p,y) - 1));

s = m \* 12 \* y;

printf("Monthly payments: %.0f $.\nThe whole payment: %.0f $.\n", m, s);

}