MCS: Identification, Authentication and Authorization 2020-21

Practical Exercises: Pluaggable Authentication Modules

April 1, 2022 Due date: no date

Changelog

- v1.0 - Initial version.

1 Introduction

The goal of these exercises is to explore the functionalities of PAM (Pluggable Authentication Modules) infrastructure present in current Linux distributions.

These exercises will also make use of several authentication methods, such as username and password, biometrics, one-time passwords and asymmetric key pairs (exploring the Portuguese Citizen Card as a way to authenticate users and to provide detailed logs).

Errors in PAM configuration files may block further access to the system. Beware! Keep at all times a session as root to be able to recover from mistakes!

2 Weak Passwords

By default, current Linux distributions do not verify the strength of the passwords in use. A weak password is a password which can be found through an intelligent search attack where a set of more likely passwords are tested (dictionary attack). The widespread availability of dictionaries with words and even well-known passwords, makes this attack very effective. Once the password is found, the security of the entire system is compromised as it can allow the execution of local exploits and access private data.

This policy can be changed so that the password modification procedure verifies the strength of the new passwords. In Linux these procedures are controlled using PAM (Pluggable Authentication Modules), which is also used to control several other authentication and access control aspects of any Linux system.

2.1 The pam_cracklib PAM module

Install `libpam-cracklib` using `apt-get`.

Take a look at the manual of `pam_cracklib` (

```
man pam_cracklib
```

) and understand the purpose of the module.

Create a test user named `beaker`[1] (with the `adduser beaker` command) and set a password for this user. Verify that you can login into its account (with the `su -l beaker` command).

The file `/etc/pam.d/common-password` stores the rules controlling what happens when users change passwords. This is a good candidate for enforcing strong passwords and is typically used for this purpose.

NOTE: as a general safety rule never close the editor where you change PAM configuration files before testing the effect of the modifications performed. Otherwise, you may not be able to revert your mistakes!

- Propose a reasonable policy for password security and enforce it. Consider password length, existence of symbols (e.g. `_,;:"’‘~^+-*/=<>[]{}()!?@#$%&\`), lower and upper case characters and (decimal) digits.
- In order to better enhance the difficulty of finding the password, enforce a policy so that common passwords found in the system dictionary are forbidden. You can use the dictionary files available at `/usr/share/dict`.

Using the `beaker` user account, test the correct enforcement of the password safety model you have implemented.

In the end, please restore all PAM-related configurations! Otherwise you may be unable to log in into the system.

[1] Beaker is a character from The Muppet Show; he is a laboratory assistant and often a Guinea-pig for dangerous experiments.

Photo extracted from `https://en.wikipedia.org/wiki/Beaker_(Muppet)`

3 Biometrics

PAM can naturally support the use of biometrics for authentication. But since this requires some kind of biometric reader for capturing biometric features, we decided to use one that is common in most laptops used by people: a camera. And, with a camera, we will explore biometric authentications using faces.

To this end we will use `howdy`, a software that implements a Linux replica of the Windows Hello™style facial authentication. It uses OpenCV and is implemented in Python. Python PAM modules can be used by means of a middleware module, `pam_python`, that makes the necessary bridge between C language PAM library calls to equivalent functions in Python. This module is distributed by the package `libpam-python`.

The instructions for installing `howdy` are provided in its GitHub repository, and are the following. First, add its repository to the list of repositories used by the laptop,

```
sudo add-apt-repository ppa:boltgolt/howdy
```

then update the list of packages,

```
sudo apt update
```

and finally install `howdy`,

```
sudo apt install howdy
```

Once installed, you can check that the installation steps included an optional facial authentication with howdy prior to the normal Linux authentication with username and password. You can check this in the file `/etc/pam.d/common-auth`, where you should have something like this:

```
auth [success=2 default=ignore]      pam_python.so /lib/security/howdy/pam.py
auth [success=1 default=ignore] pam_unix.so nullok_secure try_first_pass
auth requisite pam_deny.so
```

These 3 lines are interpreted as follow:

1. The `howdy` module is called. If the authentication succeeds, with will jump over the next 2 lines (as indicated by the clause `success=2`). Otherwise, it is ignored (as stated by the `default=ignore` clause) and the process continues in the next line.
2. The `pam_unix` module is called. If the authentication succeeds, with will jump over the next line (as indicated by the clause `success=1`). Otherwise, it is ignored (as stated by the `default=ignore` clause) and the process continues in the next line.
3. The `pam_deny` module is called. It always yields a failure, and since it is a requisite, a final failure decision was reached.

The `howdy` module has a timeout in the facial recognition, and it is mostly silent.

The configuration of `howdy` has two separate components: the definition of the personal profiles and the general configuration of the facial detection.

The definition of the facial profile for the current user is performed with the command

```
sudo howdy add
```

or

```
sudo howdy add -U <username>
```

for another user. For each user you can capture several snapshots, and you can label them accordingly (normal, with glasses, smiling, etc.).

The general configuration is performed with the command

```
sudo howdy config
```

It launches a `vim` editor with the configuration option for `howdy`.

Of special interest are the parameters `certainty` and `use_cnn`. A low value for certainty make the system more accurate (low False Positive Ratio), but also more difficult to use (high False Negative Ratio). With CNN (Convolutional Neural Network) instead of HOG (Histogram of the Oriented Gradient), with a value of 2 you should succeed from time to time, but not very often. With 2.8, on the other hand, it should balance well accuracy with usability. With higher values, it will basically accept any face as a good match.

Experiment these values, with both CNN and HOG. Try to use photographs or computer images as alternatives to faces. Use several accounts, each for different people, and use them to evaluate your system's accuracy and usability.

4 One-Time Passwords

One-Time Passwords (OTPs) can also be used for authenticating users. OTP systems can have different flavours; the oldest one computes uses a list of OTPs that users must print and carry with them. Other system use an application executed in a personal device for computing OTPs. In this guide we experiment the former.

4.1 Printed OTPs

As expected, PAM supports printed OTPs by means of a library and a definition in the configuration files. In Linux systems, it is required to install the packages `libpam-otpw` and `otpw-bin`.

If not available, get the source code file `otpw-1.5.tar.gz` from `https://www.cl.cam.ac.uk/~mgk25/download`, unpack it (with

```
tar xzvf otpw-1.5.tar.gz
```

install the package `libpam0g-dev`, run the

```
make
```

command to generate the necessary binaries and, finally, run

```
sudo make install
```

to install those binaries on the right directories. The `pam_otpw.so` library will be installed in the correct directory (`/lib/x86_64-linux-gnu/security`).

The next step is to add the `pam_otpw.so` module to the appropriate place in the PAM authentications stack. For the purpose of this laboratory guide, add it to the `common-auth` file, as an alternative authentication method relatively to the UNIX one:

```
auth    [success=2 default=ignore]      pam_unix.so nullok_secure
auth    [success=1 default=ignore]      pam_otpw.so
auth    requisite                       pam_deny.so
```

Also, comment `howdy` to experiment only with `pam_otpw` and do not forget to adjust the success jump values.

You should take in consideration that the order is important! Place `pam_otpw.so` after `pam_unix.so`. Note that with this configuration you can choose among two types of authentication: first, with Unix, second, with OTP; if the first fails, OTP will be attempted. If they both fail, the authentication will fail.

NOTE: do not close the editor! Just write the contents of file being edited

After configuring PAM for using `pam_otpw.so` users can choose to use OTP by running the `otpw-gen` command. This will generate a file named `\textasciitilde/.otpw` containing some metadata, as well as a list of one-way hashes for the purpose of verifying the response to challenges.

Run the command in order to create the above referred file. You will need a password prefix. Choose any string and remember it! The prefix will be required for authentication purposes. A table will also be printed to the generator's standard output; this table provides the responses to the several challenges initiated by the PAM OTPW module. In a real exploitation scenario, this table would be printed on a sheet of paper or on a card. In this exercise you can save it on a file by redirecting the output of the generator.

```
otpw-gen > otps_to_print
```

To test if the system is working, execute `sudo bash` command to initiate a superuser shell session in the current console. Provide an empty password for the first password prompt. You should be presented with a new password prompt carrying a number as challenge. The correct answer is composed by the password prefix and the corresponding entry in the previously referred table (including or not the space). If multiple numbers are provided, you must provide the answer to all challenges. Spaces are ignored.

Test now with a user for which this system was not set up: `beaker`. For that, execute the command

```
su - beaker
```

In this case, you will see that the PAM OTPW module will not ask for an OTP, since it does not have a way to check it!

4.2 Google Authenticator

This PAM module works together with the Google Authenticator App, which you should add to a mobile terminal. This mechanism uses a shared key (not a password) for computing OTPs. In detail, it implements HOTP (HMAC-based One Time Password algorithm), which computes OTPs from the shared key and loosly synchronized counter, and TOTP, a HOTP extension that uses the current time as the challenge.

First, install `libpam-google-authenticator` using `apt`:

```
apt install libpam-google-authenticator
```

This installation keeps your PAM orchestration files unchanged, you must change them manually. Now you should have access to the `google-authenticator` application, which you can use to create a login random secret for the current account.

4.2.1 Time-based One-time Password (TOTP)

Running the `google-authenticator` application, it asks whether you want to use TOTP (a negative answer falls back to HOTP). Select TOPT for this first experiment.

Once selected the authentication method, the application will present you 3 ways for adding the randomly generated secret key to your mobile terminal:

- A Google link (which should not be used, as it exposes the key to Google);
- A QR code, that you can scan with the application in your mobile terminal to import the key; or
- The key itself, that you can enter manually in the mobile terminal.

Once configured, you can use the code presented by the mobile application to check if the setup is working; notice that for TOTP the codes presented are time limited.

The setup completes with several other steps.

First, the user is presented with a set of 5 emergency scratch codes. These are codes that you should print and use in case you loose the secret key. You can only use each of them once.

Second, you configure whether you can use the same TOTP password several times during its validity period. Allowing it increases usability, but decreases robustness against eavesdroppers.

Third, you are asked if you want to allow to compensate time skews between the computer where you are configuring this method and the mobile terminal that generates the OTPs. Since usually mobile terminal will have a network connection, and thus will be able to keep their clock synchronized, you can assume that time skews will be negligible.

Forth, you can limit the number of wrong input credentials to 3 per each 30 seconds. This alternative is the safest one, but do not select it to facilitate the experiments.

This configurations correspond to running the command as follows:

```
google-authenticator -t -u -W -d
```

Now, its time to add this OTP method to the PAM configuration files. Manually edit the `common-auth` file, and add this other authentication method to the ones already existing:

```
auth    [success=2 default=ignore]      pam_unix.so nullok_secure
auth    [success=1 default=ignore]      pam_google_authenticator nullok
auth    requisite                       pam_deny.so
```

Also, keep `howdy` commented and comment `pam_otpw`.

At this moment you should have your mobile application configured to login in this host using TOPT. Try it, using exactly the same commands used for experimenting the OTPW module.

4.2.2 HMAC-based One-time Password (HOTP)

Run again the command `google-authenticator` to set up an OTP mechanism based on a loosely coupled counter:

```
google-authenticator -c
```

Import the new configuration for your mobile application, it should overwrite the exist profile for your account. With this new configuration, the user is supposed to request a new code each time it needs to.

Each time you request a new code, the local counter is incremented. And, on the authenticator side, the same happens each time the PAM module is called. However, a desynchronisation can occur if (i) the user requests codes without need and (ii) the authenticator increments the counter upon a malicious logging attempt. The authenticator supports a configurable amount of clients' desynchronisation, by looking ahead for a correct response. The look-ahead amount is adjustable, being 3 by default. Thus, if the user requests more than 3 OTPs without using them, their counter will become desynchronised and they need to resort to the scrap codes. To thwart malicious increments on the authenticator side, you should use the following module option: `no_increment_hotp`.

Experiment the system with a correct synchronization. Then, request more than 3 OTPs without using them and try again; it should fail. Now make only attempts with fake OTPs. After a few, you should be able to authenticate again with the correct OTPs (the counters got close enough, and now became synchronized). Add the option above referred to the PAM module:

```
auth    [success=2 default=ignore]      pam_unix.so nullok_secure
auth    [success=1 default=ignore]      pam_google_authenticator nullok no_increment_hotp
auth    requisite                       pam_deny.so
```

Try again to use fake OTPs and confirm that you did not succeed in getting the counters desynchronised.

5 Authentication using Smartcards

An example source code in C of a PAM module implementing local authentication with the Portuguese Citizen Card (CC) can be found at: `https://code.ua.pt/projects/ccpam`. We can get the source code using `git` and the following command:

```
git clone https://code.ua.pt/git/ccpam
```

Before you can compile the module, you should install the PAM library development files (`libpam0g-dev`), the C language PKCS#11 development files (`libopencryptoki-dev`) and the OpenSSL library development files (`libssl-dev`) using `apt`:

```
apt install libpam0g-dev libopencryptoki-dev libssl-dev
```

Also install the Portuguese Citizen Card middleware[2].

Compile the module by executing

```
make
```

and install it with

```
sudo make install
```

Edit the `common-auth` PAM configuration file and a line the line using `pam_PTEIDCC.so` as follows:

```
auth     [success=2 default=ignore]      pam_unix.so nullok_secure
auth     [success=1 default=ignore]      pam_PTEIDCC.so /etc/CC/keys
auth     requisite                       pam_deny.so
```

as the second rule in the authentication stack. Keep all the other methods other than the UNIX one commented.

Using the `addCCuser` tool to bind a given CC to the test user (the default file is `/etc/CC/keys`).

Verify that you can use the CC to authenticate the test user.

Modify the library so that the actual name and BI of the user are logged to the `auth.log` file. You will have to consult the CC SDK documentation in order to determine the relevant PTEID API functions to use.

The following code may be helpful to log messages to `syslog`:

```c
static void pam_cc_syslog( int priority, const char *format, ... )
{
    va_list args;
    va_start( args, format );

    openlog( "pam_PTEIDCC", LOG_CONS | LOG_PID, LOG_AUTHPRIV );
    vsyslog( priority, format, args );
    closelog();
    vfprintf( stderr, format, args );
}
...
pam_cc_syslog( LOG_ALERT,"Name: %s Civil ID: %u\n", name, civil_id );
```

Listing 1: Helper Function to write a string to syslog

[2] `https://www.autenticacao.gov.pt/cc-aplicacao`

Once again, verify that this information is indeed present in the system authentication log file, `/var/log/auth.log`.

6 Final remarks

Use the several PAM modules experimented in this guide to make several kinds of two-factor authentication.

At the end, leave the configuration file as it was after installing `howdy`, but you may want to comment the line using it. That line should be removed upon uninstalling the `howdy` package.

Note: see if all works well with the final configuration before leaving the **NOTE: see if all works well with the final configuration before closing the editor!**

7 References

- Linux PAM web site: `http://www.linux-pam.org`
- Linux PAM Howdy GitHub web site: `https://github.com/boltgolt/howdy`
- Linux PAM OTPW web site: `https://www.cl.cam.ac.uk/~mgk25/otpw.html`
- Linux PAM Google Authenticator GitHub web site: `https://github.com/google/google-authenticator-libpam`
- Portuguese Citizen Card web site: `https://www.cartaodecidadao.pt`