Lab 9:

Pluggable Authentication Modules

Linux Server Security  
 2024-2025

© Daan Pareit

## Introduction

# Lab concept

During this lab, we will play with Linux’ Pluggable Authentication Modules (PAM).

# Learning goals

* Locating the PAM modules
* Reading/changing PAM configuration files
* Adding 2-factor authentication (2FA) via PAM

# Practicalities and prerequisites

You’ll need the following:

* Your Debian VM and the pfSense VM as installed during previous lab(s)
* A smartphone for 2FA (although it’s also possible without smartphone)

## PAM USAGE

* Where are the PAM modules located on your machine (Debian/RHEL)? As *pam\_unix.so* is probably the most important module that is (almost) always included in your distro, use *find* to look for it.

Debian

A screen shot of a computer

Description automatically generated

RHEL

A black background with white text

Description automatically generated

* As PAM modules are just dynamic libraries (shared objects), we can use *ldd* (List Dynamic Dependencies) to check whether a binary supports PAM or not. Use man ldd to inform yourself about the *ldd* tool. Invoke *ldd* on su, sshd, … Find the location of these binaries and provide these binaries as argument to *ldd* .  
  Is PAM supported by *su, sshd, …* ?

A screenshot of a computer program

Description automatically generated

Seems like both of them binaries support PAM.

ldd /usr/sbin/sshd | grep pam

## PAM for 2 factor authentication with Google Authenticator

2FA (2 factor authentication) provides a more secure way for authentication as a verification code is also needed during authentication, besides your password. This verification code is a one-time-password (OTP) which changes over time (TOTP) or after each usage (HOTP). When your password is stolen, thieves still cannot login to your service without a valid verification code. 2FA can be enabled on Gmail, Dropbox, Facebook, etc.

We will enable 2FA (2 factor authentication) for SSH connections to our DEBIAN machine and need a specific PAM to be installed/configured for this.

# The Network Time Protocol (NTP)

The 2FA method we’ll use, will be time based (TOTP). We need to ensure a correct date and time on our server (which is sometimes a problem when using VMs).

* Luckily, there’s already a service taking care of synchronizing time using NTP (Network Time Protocol). That’s done via -you guessed it- systemd. Check this with:

systemctl status systemd-timesyncd.service

A screenshot of a computer program

Description automatically generated

# Install the Google Authenticator PAM module

You need to install the **PAM module** which provides the Google 2FA service.

* We will use the following library: <https://github.com/google/google-authenticator-libpam> . This is conveniently available via the package manager :  
  apt install libpam-google-authenticator
* Verify that this installs a new **PAM module**. Where is it located?

A screen shot of a computer code

Description automatically generated

# Enable the Google Authenticator PAM module for SSH

Now the PAM module of Google Authenticator is installed, we can enable it in the services where we want the extra 2FA by changing their PAM configuration files. We choose to enable this extra 2FA security when logging in over SSH.

* Now, you will need to change the **PAM configuration file** for the Secure Shell Service. Which file would that be?

/etc/pam.d/sshd – this looks like the configuration file for the whole thing.

* In this configuration file, add the following entry below the other entries with the ‘auth’ function type (which are here in Debian combined via an “@include common-auth”):  
  auth required pam\_google\_authenticator.so

A screen shot of a computer screen

Description automatically generated

* Restart the sshd service for the changes to be impacted.  
  Note that connected ssh users will not lose their connection. Only new SSH connections are impacted.

Sudo systemctl restart sshd.service

* Try starting another ssh session from your laptop to debian; it should no longer work

A screenshot of a computer

Description automatically generated

Indeed.

# Configure the Google Authenticator

The reason SSH connections no longer work, is because you enabled the Google Authenticator module to be used for SSH (by adding it in its PAM configuration file), but you didn’t yet configure the Google Authenticator itself for the user(s).

In the next step, we will create the 2FA secret key for each user. To add the secret to your smartphone, the most convenient way is to scan a QR code (instead of typing the secret key in your smartphone).

* Install the google authenticator app on your smartphone:

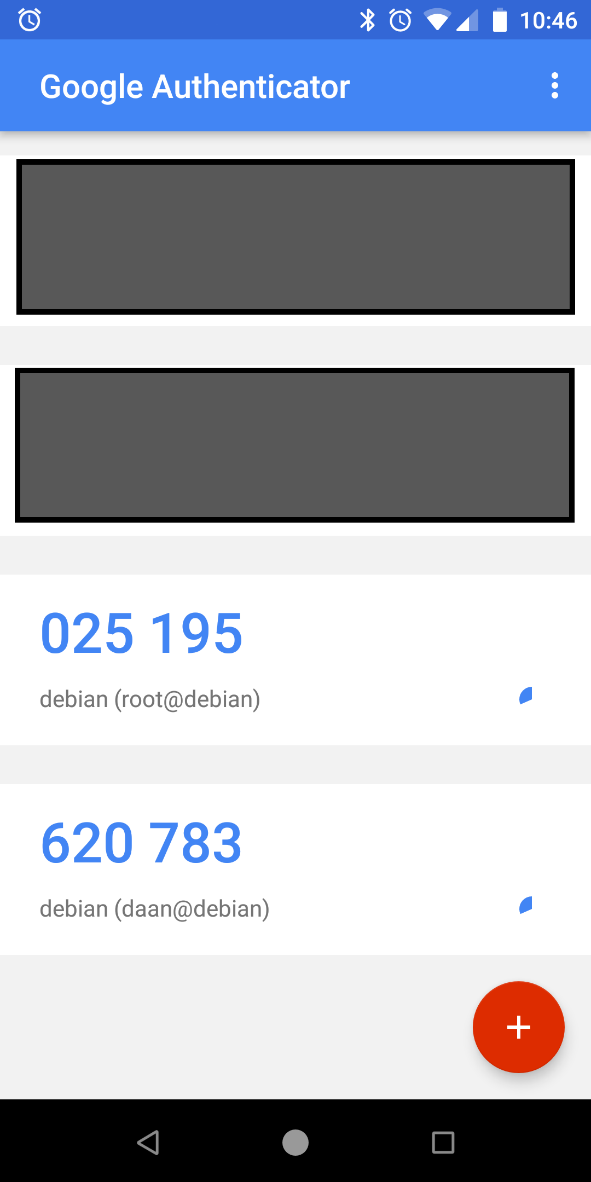
|  |  |
| --- | --- |
|  | <https://play.google.com/store/apps/details?id=com.google.android.apps.authenticator2> |
|  | <https://itunes.apple.com/nl/app/google-authenticator/id388497605> |

* You still have an ssh connection open for your user account for which you will create the 2FA secret key. If not, you need to login via the local tty console (in the VMware software).
* Now, create the 2FA secret key:
  + Run google-authenticator   
    google-authenticator --qr-mode=utf8
  + Anwer ‘Yes’ to the question for time-based tokens.  
    You will obtain something like the output below:



Note: If you are doing this in a local tty instead of over ssh, your QR code will not be shown properly.

* + Answer ‘-1’ to skip a code from app or scan the QR code with the google authenticator app on your phone already to verify codes are generated as expected.
  + Anwer ‘yes’ to update the .google\_authenticator file
  + Anwer ‘no’ to restrict login every 30s (‘yes’ would be better in production for security, but choose ‘no’ in order for us to test it without having to wait)
  + Anwer ‘yes’ to allow more acceptable tokens (in case of larger time skew, in real life ‘no’ would be better for security)
  + Anwer ‘no’ for rate-limiting (again, in order to test it without having to wait, in real life ‘yes’ would be better)
* If you didn’t do so already, now scan the QR with the google authenticator app (or type the shown “secret key” below the QR code). The secret will be added in your app, e.g.:



* Verify that a *.google-authenticator* file has been created in your home directory. Have a look at it and verify that it contains your secret key and emergency scratch codes. The ‘emergency scratch codes’ should be saved somewhere offline, in case you lose your smartphone.

A computer screen shot of a computer program

Description automatically generated

* Try logging in via ssh; it will still not work although we just configured our 2FA secret key. Let’s solve this in the next step.

# Configure the SSH server to fully use PAM

We installed the Google Authenticator PAM module, enabled it for the sshd daemon by including it in its PAM configuration file and we created the QR code (the secret) for the specific user. We now still have to tell the SSH server it needs to use PAM and accept challenge/responses (in general, not only specifically for Google Authenticator).

* Recap: what is the config file for our SSH server again (regardless of PAM)?

Do **sudo systemctl status sshd.service** and **man sshd\_config**, then find the directory - **/etc/ssh/sshd\_config**

* Have a look at this config file and look for the ‘usePAM’ option. It should already be set to ‘yes’.

A screenshot of a computer screen

Description automatically generated

* However, the ‘KbdInteractiveAuthentication’ should be changed to ‘yes’ to allow OTP.

A black screen with blue text

Description automatically generated

* Restart the ssh service

Sudo systemctl restart sshd.service

* Try logging in via ssh, now you will be prompted for your verification code
* Verify that you now are able to correctly login via ssh.

A screenshot of a computer program

Description automatically generated

YEEEEEPY.

# Some final tweaks

* The verification code is not shown as you type. Look at the README of the repository <https://github.com/google/google-authenticator-libpam> . How do you enable visualizing the verification code you type? Implement this.

echo\_verification\_code

A screen shot of a computer

Description automatically generated

* Now try logging in using one of your emergency scratch codes as verification code, instead of using the Google Authenticator app. That should work.

A computer screen with white text

Description automatically generated

* Now login again, using the same emergency scratch code. Does this work? Have a look at your .google-authenticator file. What do you notice?

A screen shot of a computer

Description automatically generated

The used one is gone

A screenshot of a computer screen

Description automatically generated

You cant use it, of course

* Try to login locally at a tty console of your VM. Notice that no verification code is asked for here. Why is this

A screenshot of a computer

Description automatically generated

It works like this, because we enabled PAM and 2FA for SSH service specifically, hence the normal authentication methods were left untouched.