Lab 5:

shell exercises

Linux Server Security  
 2024-2025

© Daan Pareit

## Introduction

# Lab concept

During this lab, we will make some exercises on using and understanding the shell. Most of the configuration for other (lab) topics is done at the shell, so it’s important to gain a deeper understanding of that ‘thing’ you are typing your commands in. We will also explore some SSH features you might be unfamiliar with.

# Learning goals

* Identifying the shells which are used
* Understanding bash redirection
* Using ‘find’
* Using a terminal multiplexing tool
* SSH server and client keys
* X11 forwarding over SSH
* SSH tunneling

# Practicalities and prerequisites

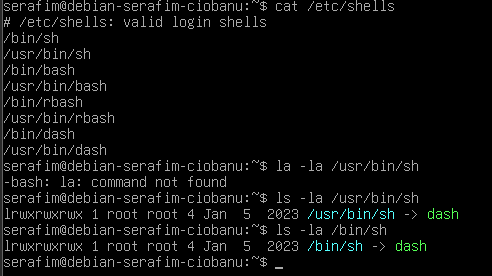
You’ll need the following:

* Your Debian and RHEL VM, as installed during previous lab(s)
* A VPN to the Howest network if not on campus

## What shell are you working on?

* Look at the file which specifies the default login shell on Debian and on RHEL (cfr lecture slides). On your freshly installed Debian/RHEL machine what are the default login shells for new users?

In both cases the interactive shells are bash. To find this out I just did cat /etc/shells, as it shows the interactive login shells.



A screenshot of a computer

Description automatically generated

* On your freshly installed Debian/RHEL machine, what is the default non-interactive shell? Wat is /bin/sh referring to?

It refers to dash on Debian, and bash on RHEL. I did ls -la /bin/sh

## Exercises on shell redirection

Every data stream has its own ‘number’ (file descriptors actually) when you are using commands on the shell. The shell identifies following for input/output by default:

0 standard input

1 standard output

2 standard error

Let’s make some exercises:

1. Delete the files tmp\*.txt, if these would already exist. To this end, execute the command  
    rm -f tmp\*.txt   
   Next, execute the following commands subsequently:
   * du /etc > tmp.txt
   * du /etc 1> tmp.txt
   * du /etc >> tmp.txt
   * du /etc 1>> tmp.txt
   * > tmp.txt du /etc
   * du /var > tmp1.txt > tmp2.txt

After every step, have a look at:

1. the output of the command
2. whether the files tmp\*.txt changed (e.g. using ls -l tmp\*.txt),
3. the content of the files tmp\*.txt (e.g. using cat tmp\*.txt or opening in a text editor).

What can you conclude after every step?

Step 1 – does the redirect of output

Step 2 – does the same thing pretty much

Step 3 – appends the output to the end of file.

Step 4 – it append the output as well.

Step 5 – this just works as a redirect.

Step 6 – all the output was gone to tmp1, but then moved towards tmp2. Or straight to tmp2.

> overwrites tmp.txt, while >> appends tmp.txt

Correct answer:

> and 1> are doing the same

1. Errors are displayed in a separate ‘channel’ to distinguish between regular standard output and error messages. Execute subsequently the following commands and have a look at the content of the files tmp\*.txt:
   * du /etc – simple command with errors
   * du /etc 1>tmp.txt – standart output to tmp.txt
   * du /etc 2>>tmp.txt – append errors to tmp.txt
   * du /etc >tmp1.txt 2>tmp2.txt – standard to tmp.txt, errors to tmp2

bash only considers the last redirection (from left to right) for the same channel (stdout 1, stderr 2 or stdin 0)

No, redirection for a command can be specified anywhere on your command line

1. Look into the following commands and their impact on redirection:
   * du /etc >tmp.txt 2>&1 – this is pretty much the normal output, as it redirects everything.
   * du /etc 2>&1 >tmp.txt – redirects only the normal output to tmp
   * du /etc &>tmp.txt – redirects both error and standard output to tmp.txt
   * du /etc 2>tmp.txt >tmp.txt – only normal output seems to be redirected.

## Exercises on ‘find’

The find command searches in all subdirectories, starting from a given directory, for files which meet certain criteria (name, size etc.). Find is a powerful tool given the right options (parameters). Being able to find the right options in the man pages is an important skill to master. The find command often generates a lot of error messages, which you preferable redirect.

1. Search for all files in the /etc directory tree which name starts with “pass”.

find /etc -type f -name "pass\*" A screenshot of a computer

Description automatically generated

1. It is also possible to combine multiple tests. How do you find all subdirectories (not regular files) whose name starts with “sh”?

find /etc -type d -name "sh\*"

1. Use the find command to obtain a list of all files in the /usr directory tree with a file size of at least 1 megabyte. When printing (one line for each file) you have to make sure that the size of the file is printed before the full path of the found files. Provide the full command line. Also make sure that you only list regular files in the list and no directories.

find /usr -type f -size +1M -exec ls -lh {} + 2>/dev/null | awk '{print $5, $9}'

finds files, with size of right amount

then executes ls -lh on the results and redirects errors to dev/null, and selects the needed options with awk.

A screenshot of a computer

Description automatically generated

1. Use find to obtain a list of all files in your home directory which were changed during the last 2 weeks. When printing the results (one line per file) you need to first print the time of the file’s last change and then the full path of the file. Only search for regular files, not directories.

find ~ -type f -mtime -14 -exec stat --format='%y %n' {} + 2>/dev/null

1. Show all files of the /etc folder that have “rwx”-rights for the current user.

find /etc -type f -perm -u=rwx 2>/dev/null

## tmux

To avoid your active processes from being killed when closing an SSH session deliberately (e.g. leaving the office) or unintendedly (e.g. connection problem) when working remotely at a server, we’ll explore the usage of a terminal multiplexer such as ‘tmux’.

* Start a ping to localhost in one ssh session (and redirect it to a file). Put the ping to the background (ctrl+ Z and then *bg*) . Check with jobs, ls -lh en tail -f. Exit ssh session and check again. Is it still running?

Ping 1.1.1.1 > ping\_output.txt

A computer screen with white and green text

Description automatically generated



Not running of course

* Now kill any existing ping process (pkill ping) and start the ping again on the foreground and close your ssh window. Login again. Is the ping still running?

Nope, it does not.

* Install the tmux package

Sudo apt-get install tmux

* Now start the ping in a tmux session. Detach from the tmux and close the (ssh) session. Login again and resume the tmux session.

Tmux new -s serafim\_ping (to create)

Ping 1.1.1.1

Ctrl+b d – to detach

Tmux ls – to check the working ones.

A screen shot of a computer program

Description automatically generated

Tmux attach -t serafim\_ping – to attach to the session existing.

And yes it still works.

A screen shot of a computer

Description automatically generated

## SSH server and client keys

There’s a server available at Howest, known as ‘gallie.snwb.howest.be’ (or 172.21.24.7 if DNS resolve doesn’t work) at which you can login with your Howest credentials.

Note: authentication is done via LDAP and if your “firstname.lastname” is longer than 20 characters, it will be abbreviated: the 20th and following characters are replaced with character “1”. E.g. “firstname.alongerlastname” would be replaced by “firstname.alongerl1”

🡺 Login to that server with your Howest credentials over SSH:

ssh firstname.lastname@gallie.snwb.howest.be

or ssh firstname.lastname@172.21.24.7

The first time you connect, you are presented with a key fingerprint **to identify the server**. Accept that.

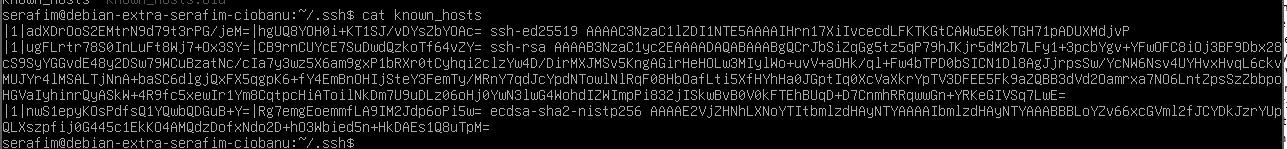
A computer screen shot of a black screen

Description automatically generated

🡺 Immediately exit the server. Where is this server key saved for future connections on your debian-extra (cfr slides)? Have a look at the contents of that file and write it down.

But why Debian extra?)

The hosts we connect are stored in /.ssh/known\_hosts.



🡺 Now login to the server again, you’ll notice the server key is no long

er presented as you’ve connected before. Where is this SSH server key located on the server (cfr slides)? Print it contents and verify it is the same (public) key as is now stored on your debian-extra.

A black screen with white text

Description automatically generated

As far as I can see – they are the same. So that I know the public key of the server.

🡺 To login to the server, we now want to use a key pair to **identify your user** account instead of having to type your Howest password every time. Therefore, on your debian-extra, generate an ssh public/private key pair for your debian-extra user with ssh-keygen . You don’t need to provide a passphrase here. Where is your private key stored on debian-extra? And your public key?

Stored in ~/.ssh/ directory. Both.

A screenshot of a computer

Description automatically generated

🡺 Now copy your public key to the server using:

ssh-copy-id [firstname.lastname@gallie.snwb.howest.be](mailto:firstname.lastname@gallie.snwb.howest.be)

A screenshot of a computer program

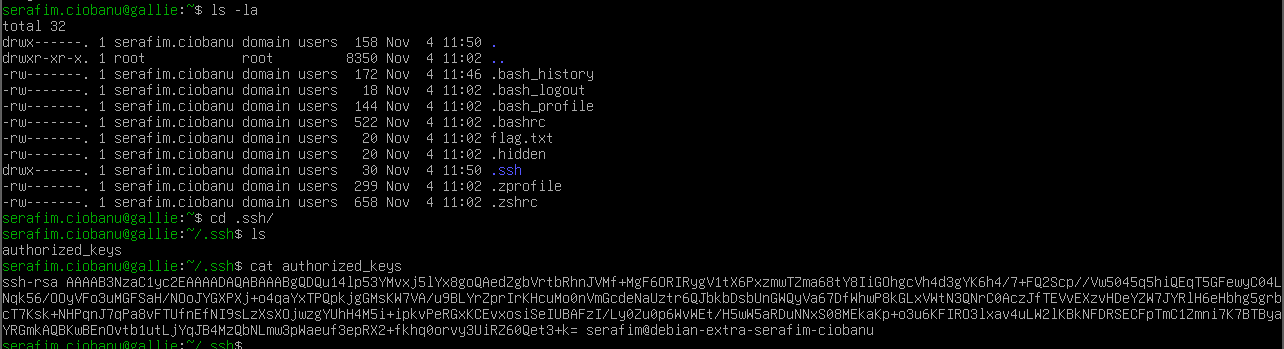
Description automatically generated

🡺 Try ssh’ing to the server again. You should now be logged in without any password 😮 but by the automatic exchange of messages based on the keys you’ve created and your public key which is available at the server.

A screenshot of a computer

Description automatically generated

Authorized\_keys is the filename and yeah it looks like the one I need.



🡺 Where is your public key stored on the server (cfr slides)? Have a look at that file and verify that it is indeed the same key as the public key you generated and stored on your debian-extra.

A screenshot of a computer

Description automatically generated

## SSH tunneling

Preparation phase:

1. On debian-extra go to (or create) a folder that we will share over a SSH tunnel.
2. Create a file with the name **lab.txt** and put your firstname and lastname in it (for example: “Daan Pareit”).
3. Run the built-in HTTP server of python with python3 -m http.server in the same folder that contains the **lab.txt.**
4. Verify that you can locally surf to your VM to view your lab file: <http://192.168.22.42/lab.txt>

don’t forget to specify port 8000

A screenshot of a computer

Description automatically generated

Our SSH server (gallie.snwb.howest.be) is the server you will use to make your your lab.txt available at a **port number** you are assigned, see below. (Note: the server is only available over VPN when at home.) The goal is that if one is surfing to [http://gallie.snwb.howest.be:<portnumber>/lab.txt](http://gallie.snwb.howest.be:%3cportnumber%3e/lab.txt), he/she ends up accessing *your lab.txt file* on *your virtual machine* 😱! In other words, this way you are using SSH port forwarding so that everyone in the (Howest) network can access your lab.txt, as long as your tunnel is open. And that without making any port forwarding changes in pfSense, VMware or your home router!

1. Go to **Leho**. There will be a **pdf** file with your name in it but more importantly a number starting with 8 or a 9 (for example 8001). This is a port number reserved for you! Take note of this **(port) number**.

8014

1. ssh to gallie.snwb.howest.be *with the correct SSH tunneling option* (cfr slides and/or man page) to make the above described goal happen. (SSH’ing to the server should be passwordless if you perform the first exercises, else use your Howest credentials.)

ssh -R 8014:localhost:8000 [serafim.ciobanu@gallie.snwb.howest.be](mailto:serafim.ciobanu@gallie.snwb.howest.be)

This is like to make the outdoor traffic (remote) coming to gallie at 8014 to go to my machine with port 8000 where I serve the file. Also made sure to constantly run the python server with tmux.

1. Now, use a web browser on your laptop to surf to gallie.snwb.howest.be:*<your-port>*/lab.txt and you will be served with the file which resides on your VM on your laptop, not a file on our server 😱😱😱

A screenshot of a computer

Description automatically generated

## X11 forwarding over SSH

Mostly, you do not want a desktop environment on your server. It slows down the boot process, consumes disk space and memory and can interfere with your command line settings is you’re not aware (e.g. the network-manager service ). However, it can often be convenient to use a graphical (configuration) tool for your server. To this end, it can be useful to install a basic Xserver on your server, but without a desktop environment. You can then visualize graphical applications of your server on a remote machine instead.

🡺Install a package with some simple graphical applications such as *xclock* or *xcalc* via the *x11-apps* package. Or install a webbrowser such as *chromium* or *firefox-esr* . This will also install the x11-common package for the xclient.

🡺On your laptop, install an Xserver for the SSH client you use to login remotely to Debian. You can e.g. use Xming (https://sourceforge.net/projects/xming/) as xserver for putty or use MobaXterm (https://mobaxterm.mobatek.net/) with builtin xserver.

🡺Login with X forwarding over ssh to Debian. On a linux machine you would use the ssh -X option. When using putty, you need to check the X11 forwarding check box in the settings. MobaXterm automatically uses X forwarding

🡺 Now run your graphical application xclock, xeyes, chromium or whatever you installed. You should see its forwarded graphical output being visualized to the xserver on your laptop, rather than on the server itself.