Lab 6:

systemd

Linux Server Security  
 2024-2025

© Daan Pareit

## Introduction

# Lab concept

During this lab, we will explore systemd.

# Learning goals

* Controlling existing systemd service units
* Understanding systemd target units
* Creating a new systemd service unit

# Practicalities and prerequisites

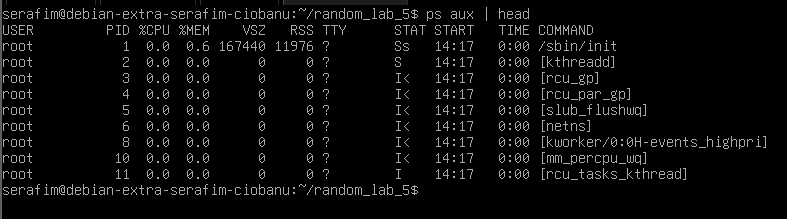
You’ll need the following:

* Your RHEL and debian-extra VM, as installed during previous lab(s)

## Simple exercises to get to know systemd service units

è Do the following on debian-extra: ps aux | head . What is the command of the process with PID 1?

/sbin/init

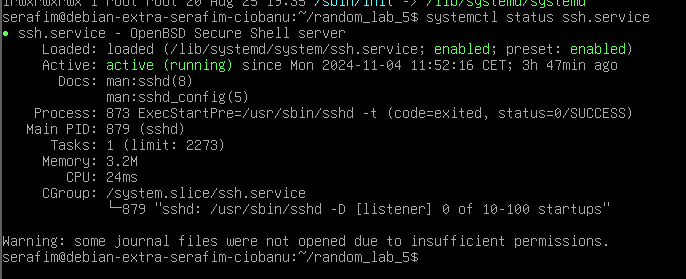


è Check the properties of that file, you’ll see it’s a symbolic link. Where does it link to? That’s the first process that is run by your linux kernel.

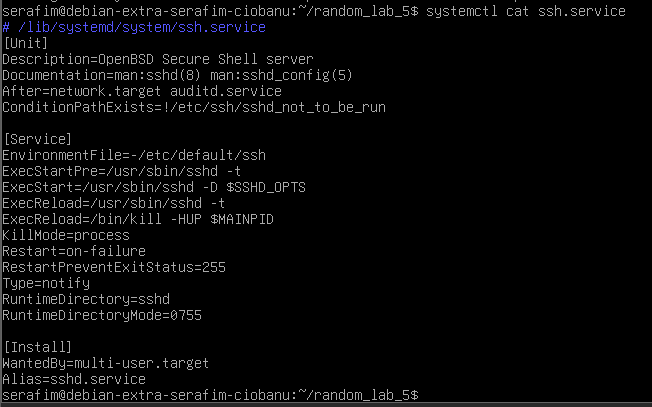
/lib/systemd/systemd



è Use systemctl status ssh.service to show you the status of the ssh server

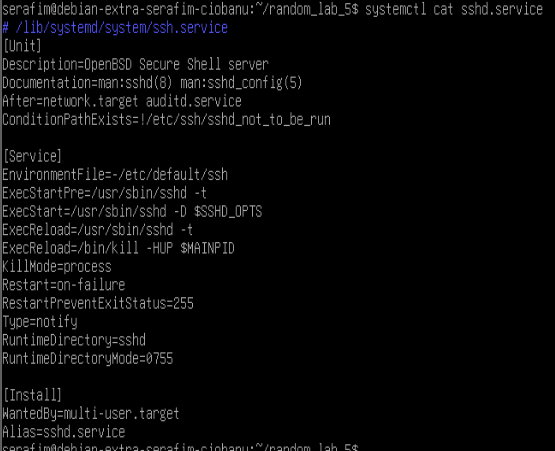


è Use systemctl cat ssh.service to show you the content of the unit configuration file.



è Based on that config file, you’ll see that this service is also known by another name. Which one? Try doing a systemctl status or systemctl cat of that service name and you’ll see you get the information of ssh.service .

Sshd.service.



è Based on the config file, what file would prevent de ssh service from starting if it would exist?

/etc/ssh/sshd\_not\_to\_be\_run

è Check the RHEL VM. By what name(s) is the ssh service known there? And is the same file which prevents the ssh daemon from running also configured there?

It is probably known by sshd.service as far as I can see.

A screenshot of a computer

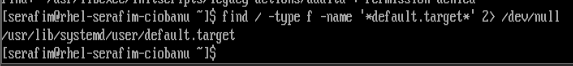
Description automatically generated

## Simple exercises to get to know systemd target units

è On debian-extra again, where is ‘default.target’ located? As rehearsal exercise, use find to figure this out.

/usr/lib/systemd/user/default.target

Find / -type f -name ‘\*default.target\*’ 2>/dev/null



è On your tty console (NOT via ssh), start the ‘rescue.target’ via systemctl start and type your root password to activate it. Then check the status of the ssh service. Is it still active?

Yeah it is still active.

A screenshot of a computer program

Description automatically generated

è Exit the rescue mode, using exit, as shown in your tty console. Start rescue mode again, but now via systemctl isolate. Check the status of the ssh service. Is it still active? Now check the message on your local tty screen. Also try going to other tty consoles (ctrl+alt+f2 or f3 or f4 or ...). What do you notice?

A screenshot of a computer program

Description automatically generated

Not active anymore

The other tty’s do not even work for now, or maybe I am just doing something wrong.

è Check the overall system status with systemctl status without any arguments. What state is our overall debian system in?

A screenshot of a computer

Description automatically generated

Maintenance.

è Exit the rescue mode again, using exit, as shown in your tty console. Check ssh service status and tty consoles again. What state is our debian system in again now?

A screenshot of a computer screen

Description automatically generated

Working!

A screenshot of a computer program

Description automatically generated

Running state is the normal one.

And other ttys work!

è Inspect the rescue.target config file. What man page is it referring to?

A screenshot of a computer

Description automatically generated

Systemd.special

è Check that man page and check the differences between emergency.target and rescue.target. Try going into emergency mode via systemctl isolate

Emergency mode is where you’ll be dropped into after booting if you e.g. have a broken RAID0 array (see ‘Datacenter Virtualization’ course).

As far as I can understand, the overall emergency mode is like the safe mode from Windows 7 era, where you only load the files for the system, gain root privileges, and other stuff like that.

Regarding the rescue – it is something intermediate.

## Create your own service

We want to make a systemd service unit from a python web server. This way, we’ll create a simple web server without the need for actual web server software such as Apache or Nginx.

è Create a /opt/exciting-webapp directory on your debian-extra machine.

è Create an ‘index.html’ file here with some simple yet valid HTML content.

A screen shot of a computer

Description automatically generated

è Now within this directory, start python3 -m http.server

è Verify, that you can surf from your laptop to the IP address of debian-extra on port 8000 and you’ll be served with your web page.

A screenshot of a computer

Description automatically generated

è Now stop the python3 process

è Create a new systemd unit configuration file for a service named ‘exciting’. Put this in the appropriate directory as the other system configuration files as you identified before.

è In this configuration file, make sure that the service would execute python3 -m http.server in the /opt/exciting-webapp directory. Hint: To set a working directory you’ll need to use “WorkingDirectory=” in the unit configuration file.

WorkingDirectory=/opt/exciting-webapp

A computer screen with green and white text

Description automatically generated

HOUSTON WE GOT A MISSPELLING! excitting and not exciting.

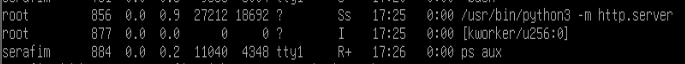
è You will have to reload the systemd manager configuration to account for the new service. How do you do this via systemctl?

Sudo systemctl daemon-reload

è Start your service and verify if your python web server is active.

sudo systemctl start exciting

sudo systemctl enable exciting



è Check the systemd journal for log entries of this specific unit.

sudo journalctl -xeu exciting

è Change your systemd config file for your service so it starts automatically after every reboot (whenever multi-user target is reached). Check the slides about dependencies (and/or compare them with another service config file). Verify it works after rebooting.  
WantedBy=multi-user.target

è Now we want to ensure that this service cannot be started manually via ‘systemctl start exciting.service’ nor that it is started via dependencies of other services/targets (without deleting/changing config files). What systemctl command do you have to execute to obtain this? And what/how will systemctl do to accomplish this?

Sudo systemctl mask exciting.service

When you mask a service with systemctl mask, systemd creates a symbolic link from the service file to /dev/null. This action prevents systemctl (or any dependencies) from locating the actual service file, making it impossible to start the service manually or automatically.

What a good explanation

It will create this path /etc/systemd/system/exciting.service and point to /dev/null

è Now undo the previous changes via a systemctl command so your service can be started again.

Sudo systemctl unmask exciting.service

è Another thing you can do, now you made your program into a service, is to automatically restart if the process would stop for one reason or another. Add the “Restart=always”in the [Service] section of your config file.

è Reload your systemd, search the python3 process, kill it and verify your python3 web server is recreated.

è Analyze your services with systemd-analyze security . The exposure rate indicates the amount of sandboxing of a service (the lower, the better). What is the predicate/exposure of your exciting service?



è Try adding ‘PrivateTmp=yes’ in your [Service] section and reload your systemd. That provides a separate folder for temporary files in your /tmp directory for your exciting service, instead of using the shared /tmp. Does the exposure rate change?

You do sudo systemctl daemon-reload and then check. It changes.



è With systemd-analyze security exciting , you see different possible improvements. Try e.g. adding the following and verify the change in exposure rate:

PrivateTmp=yes

NoNewPrivileges=true

ProtectSystem=strict

CapabilityBoundingSet=CAP\_NET\_BIND\_SERVICE CAP\_DAC\_READ\_SEARCH

RestrictNamespaces=uts ipc pid user cgroup

ProtectKernelTunables=yes

ProtectKernelModules=yes

ProtectControlGroups=yes

PrivateDevices=yes

RestrictSUIDSGID=true

A screen shot of a computer

Description automatically generated

Yeepy

## Now, let’s create yet another service

We want again to make a systemd service unit for a simple web server, now based on the netcat tool.

è On leho, you’ll find a shell script (what shell will execute this? Look at the shebang), named ‘mysimpleserver.sh’ and a file ‘index.http’ with hard coded HTTP response in text format. Download these files.

It is the default sh, most probably bash.

è Create a /opt/awesome-webapp diraectory on your debian-extra machine.

è Put both files into that directory. Hint: use scp to do this fast and easily. (SFTP is better)

è Make the script executable and test if running the script indeed enables a web server via netcat on the specified port within the script.

A screenshot of a computer

Description automatically generated

è Create a systemd service named ‘awesome’ to put the script in a service and verify it works.

A screen shot of a computer

Description automatically generated

A computer screen shot of a black screen

Description automatically generated

Both work and server (even after reboot)

## Create your own target

We want to create a custom target which combines both services we created in the steps above.

è Create a systemd target ‘myextraservices’

sudo nano /etc/systemd/system/myextraservices.target

A screen shot of a computer program

Description automatically generated

sudo mkdir /etc/systemd/system/myextraservices.target.wants

è Ensure that you link the ‘exciting’ and the ‘awesome’ service to this target without explicitly mentioning these in the config file of ‘myextraservices’. How do you do this?

sudo ln -s /etc/systemd/system/exciting.service /etc/systemd/system/myextraservices.target.wants/exciting.service

sudo ln -s /etc/systemd/system/awesome.service /etc/systemd/system/myextraservices.target.wants/awesome.service

è Verify if it works by first stopping ‘exciting’ and ‘awesome’ and then starting the ‘myextraservices’ target.

Sudo systemctl stop <both>

sudo systemctl start myextraservices.target