Lab 7:

Linux firewalling

Linux Server Security  
 2024-2025

© Daan Pareit

## Introduction

# Lab concept

During this lab, we will explore different firewalling techniques on linux machines.

# Learning goals

* Using/understanding iptables and netfilter
* Using/understanding nft and nftables
* Using/understanding firewall-cmd and firewalld
* Using config files for TCP wrappers
* Using systemd config files for firewalling for specific services

# Practicalities and prerequisites

You’ll need the following:

* The fully configured network with all your VMs as installed during previous lab(s)
* Hint: make snapshots of your VMs once your network is correctly configured, before starting the lab

## Configuring netfilter on RHEL using iptables

Let’s explore the iptables command on RHEL, neglecting RHELs firewalld for now (it has a separate table and ruleset which you’ll use later in this assignment):

🡺 With iptables we have predefined tables. Use iptables -L to show you the rules of the default table. What are its chains, policies and what rules does it contain? What is the name of this default table (see slides or man page)?

A screen shot of a computer

Description automatically generated

This one is called “filter” table.

Chains : INPUT, FORWARD, OUTPUT

Does not seem like it has default rules set.

It has (or can have) rules that do forward through the host, gotten by the host, or sent by the host.

🡺 Now, use iptables with appropriate arguments to show you the rules of the ‘raw’ table. What are its chains, policies and what rules does it contain?

sudo iptables -L -t raw

A screen shot of a computer code

Description automatically generated

Prerouting – changing packages as soon as they come in.

Output – change before packages are sent out

🡺 Now, via the iptables command with appropriate arguments (see slides or man page), create a rule (in the default table) to log *all* *incoming ICMP* packets

A screen shot of a computer

Description automatically generated

sudo iptables -A INPUT -p icmp -j LOG

-A = append to a chain

-p = for protocol

-j = what to do if the packet matches it

🡺 Check your rule with iptables -L. What is mentioned as source and destination?

Anywhere and anywhere, since we did not specify anything extra.

🡺 Check the log via journalctl (cfr previous lesson on systemd) to verify log entries are dynamically being added while e.g. running a ping from debian to RHEL or from your laptop to RHEL.

A computer screen shot of a program

Description automatically generated

🡺 To easily filter in the log file, it is better to provide some kind of tagging in your logging and choose the log level you want. Delete your previous rule and recreate it again but by using extra arguments to (1) tag your iptables logs with a prefix of your choice and (2) specify it to have log level ‘info’. Have a look at the man page of ***iptables-extensions*** on how to do this. Check your logs to verify it works.

sudo iptables -A INPUT -p icmp -j LOG --log-prefix "IMCP INCOMING -- " --log-level info

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

Kind of strange but I guess it is fine.

🡺 Start pinging from debian to RHEL. Now, on RHEL, create with iptables an additional rule to drop all *incoming* *ICMP* packets (and no other packets). Your ping should no longer work.

A screen shot of a computer

Description automatically generated

sudo iptables -A INPUT -p icmp -j DROP

A black screen with white text

Description automatically generated

I mean it stopped working hehe.

🡺 Flush the filter table in iptables and block again all incoming ICMP on RHEL. However, now use REJECT as rule target, instead of DROP. Do you notice a difference?

Sudo iptables -F

A screen shot of a computer

Description automatically generated

sudo iptables -A INPUT -p icmp -j REJECT

A screenshot of a computer

Description automatically generated

Now it says that the port is unreachable, so an actual reason why doesn’t it work.

🡺 Try now pinging from RHEL to debian (instead of debian to RHEL). Does this still work? Why (not)? What is debian doing with these pings? Check with a sudo tcpdump -i ens33 icmp on debian.

A screen shot of a computer

Description automatically generated

The problem is that we reject the ICMP protocol, which in this case is also our response from Debian. So technically we cant get an answer.

However pinging from/to Debian-extra actually works.

🡺 Now try to be more specific in your rule. Flush the filter table again. Now only reject the *incoming ping request messages* on RHEL. What machine can still ping what machine (try combinations of your laptop, RHEL, debian, debian-extra)?

A computer screen shot of white text

Description automatically generated

sudo iptables -A INPUT -p icmp --icmp-type echo-request -j REJECT

A screenshot of a computer screen

Description automatically generated

Debian, Debian extra, laptop – cannot get to my RHEL

RHEL – can get to everyone.

🡺 Flush the iptables again and add yet another argument: rejecting again the *incoming ping request messages* on RHEL, but only those originating from debian.

A computer screen shot of a computer program

Description automatically generated

sudo iptables -A INPUT -p icmp --icmp-type echo-request -j REJECT -s 192.168.11.10

🡺 Blocking certain IP addresses (or complete subnets), as you did above, is a common aspect of firewalling. Now let’s bundle IP addresses you want to block in a separate user-defined chain. First, flush the tables again. Then, make a new user-defined chain with the name DENYLIST in the filter table on debian. Next, create a similar rule as above (for *incoming ping request messages* on RHEL), but now jump to this new DENYLIST. Then create a rule in the DENYLIST to reject if the packet originates from debians IP address.

sudo iptables -N DENYLIST

sudo iptables -A DENYLIST -s 192.168.11.10 -j REJECT

sudo iptables -A INPUT -p icmp --icmp-type echo-request

A screen shot of a computer

Description automatically generated

A small issue was detected.

A screen shot of a computer program

Description automatically generated

sudo iptables -A INPUT -p icmp --icmp-type echo-request -j DENYLIST

🡺 What happens with your chain if you flush the filter table? Is the DENYLIST chain gone? How to delete your user-defined chain? Look in the iptables/nft man page.

The chain is still there, however, it does not have the values.

sudo iptables -X DENYLIST

## Exploring netfilter on RHEL using nft

All the above was done with the legacy (but still very relevant!) iptables command in the xtables framework. Now let’s have a look at nftables framework.

🡺 When using the ‘iptables’ command, which binary is actually being used? Use ‘which’ and follow symlinks to find out.

/usr/sbin/xtables-nft-multi

A screen shot of a computer screen

Description automatically generated

🡺 Consult the xtables-nft man page. You’ll see it uses the nftables API to communicate with netfilter in the kernel. Execute the lsmod command and you’ll see the nf\_tables kernel module is loaded.

A screen shot of a computer code

Description automatically generated

🡺 Use nft to show you the currently available firewall tables with the ‘nft list tables’ command. What tables do you see? What is the name and address family shown by nft for the table in which you previously configured some rules using iptables and which you see with “iptables -L”?

A black background with white text

Description automatically generated

A screen shot of a computer

Description automatically generated

Looks like IP FILTER is the one we been like working with.

FILTER is the table name

And address family is IP

🡺 Via nft, inspect this specific table using nft list table <address family> <table name> and verify it has indeed the same chains as you saw with iptables -L . What command do you use in nft to inspect this specific table?

Well I use sudo nft list table ip table.

A screen shot of a computer

Description automatically generated

Does not look like it has all the things being present in the iptables.

Nevermind, it appears if you actually populate it with Information and rules.

🡺 In nft, how do you see the DENYLIST is a user-defined chain (or “regular chain”) and not a “base chain”?

The **absence of type, hook, priority, and policy**

A screen shot of a computer

Description automatically generated

You could manipulate all these rules or perform all the above steps with nft instead of the legacy iptables on RHEL. However, we wanted you to play around with the legacy iptables first, as it is still very common. Instead of doing the same thing over with nft on RHEL, we’ll do some other exercises with nft on debian now.

## Configuring netfilter on Debian using nft

Let’s use nft on debian, as the legacy iptables command is no longer available by default since Debian 11.

🡺 Use nft to show you the currently available firewall tables with the ‘nft list tables’ command. What tables do you see?

A screenshot of a computer

Description automatically generated

Does not look like I get any

We now want to create our own table, with its chains and rules which will *apply only to packets which are being routed by debian* from one network to another (“flowing through debian”), not the packets which are sent to/by debian host itself.

🡺 Now, create a new table “myfirewalltable” via nft which will apply to IPv4 and IPv6 network traffic (what address family to use)?

INET

sudo nft add table inet myfirewalltable

A black background with white text

Description automatically generated

🡺 Then, in that table, create a chain “myforwardchain” via nft, which you link with the correct hook. Which one of the 6 hooks do you need to use to *apply only to packets which are being routed by debian*? Type and priority can be set to ‘filter’.

sudo nft add chain inet myfirewalltable myforwardchain { type filter hook forward priority 0 \; }

iptables is so much better

A screenshot of a computer program

Description automatically generated

🡺 Now, via the nft command, create a rule in that chain to log all *ICMP echo request packets* which are being forwarded by debian.

sudo nft add rule inet myfirewalltable myforwardchain ip protocol icmp log

A black screen with white text

Description automatically generated

🡺 Check the log via journalctl -ef to verify log entries are dynamically being added while e.g. running a ping from RHEL to debian-extra.

A computer screen with white text

Description automatically generated

At least this works

🡺 Rember you can use nft list table <address family> <table name> to inspect your table but note that you can also use nft list ruleset to see all rules in all tables at once.

A screen shot of a computer program

Description automatically generated

🡺 Start pinging RHEL to debian-extra. Now, on debian VM, create with nft a rule to reject all those *ICMP echo request packets* that are forwarded on the debian VM. Your RHEL-to-debian-extra ping should no longer work.

A screen shot of a computer

Description automatically generated

sudo nft add rule inet myfirewalltable myforwardchain ip protocol icmp reject

A screen shot of a computer screen

Description automatically generated

Yes, we killed it.

🡺 Note that you could also implement logging and rejecting in one rule. Flush the chain and add logging and rejecting of *ICMP echo request packets* that are forwarded on the debian VM via one rule.

A screenshot of a computer program

Description automatically generated

🡺 Now, try pinging from RHEL to debian this time (instead of to debian-extra). Does this still work? Why (not)?

Nothing works unfortunately, probably because it is about forwarding that we did restrict, and not the INPUT traffic.

Next, we also want to block *incoming* *ICMP echo request packets* for the Debian host itself, but only for specific IP addresses.

🡺 Create a base chain “myinputchain” for incoming traffic in “myfirewalltable”.

sudo nft add chain inet myfirewalltable myinputchain { type filter hook input priority 0 \; }

🡺 Create a regular chain “denylist” in “myfirewalltable”

sudo nft add chain inet myfirewalltable denylist

🡺 Now, create a rule in “myinputchain” to make *ICMP echo request packets* jump to denylist

sudo nft add rule inet myfirewalltable myinputchain ip protocol icmp icmp type echo-request jump denylist

🡺 Then, create a rule in “denylist” to reject packets from source address 192.168.11.20 (rhel).

sudo nft add rule inet myfirewalltable denylist ip saddr 192.168.11.20 reject

A screenshot of a computer program

Description automatically generated

Time to save our current config. That’s done by output redirection (cfr lecture/lab on shells) of the nft list ruleset command to a file. You can then load your config with nft -f <yourfile> .

sudo nft list ruleset > nft\_rules\_firewall

🡺 Try saving your current config, delete the “myfirewalltable” table, and load your file to verify you can save/restore a nftables config file.

sudo nft list ruleset > nft\_rules\_firewall

sudo nft delete table inet myfirewalltable

A black background with white text

Description automatically generated

sudo nft -f nft\_rules\_firewall

A computer screen shot of white text

Description automatically generated

🡺 How can we automatically load an nftables config file at boot time? Well, as many things today, that’s via systemd service unit. There’s a nftables.service systemd unit on your system. Have a look at its systemd configuration file (cfr lecture/lab on systemd). What file is loaded at boot time when this service is started?

A computer screen shot of a computer program

Description automatically generated

/usr/sbin/nft -f /etc/nftables.conf ( /etc/nftables.conf !!!)

🡺 Copy your config to that file and enable the service for it to be activated at boot time.

sudo cp nft\_rules\_firewall /etc/nftables.conf

A screenshot of a computer screen

Description automatically generated

It is still there after reboot

🡺 The nftables package was installed by default on our debian. Let’s have a look what was included in the nftables package with dpkg -L nftables. You should by now recognize the first files as required for the systemd service, next the actual nft binary (the user level command you were using intensively) but then apparently a lot of example files as well. Have a look at e.g. workstation.nft which provides a good firewall baseline for a workstation.

/usr/share/doc/nftables/examples/workstation.nft

A screenshot of a computer screen

Description automatically generated

A screen shot of a computer

Description automatically generated

## Configure netfilter on RHEL using firewalld

Back to RHEL now. RHEL uses the firewalld daemon with the firewall-cmd user tool to provide a dynamically managed firewall with support for different network/firewall zones to define the trust level of network connections or interfaces. This approach makes it more manageable than defining nftables rules yourself, but firewalld does use nftables underneath (hence the importance to understand nftables).

🡺 Use the nft tool to inspect the different existing tables. What table(s) are being used by firewalld to create its rules in?

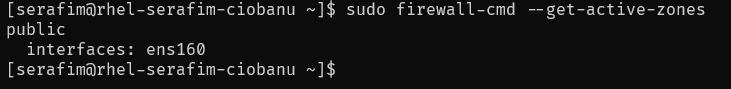
A black background with white text

Description automatically generated

Firewalld, obviously. Family INET

🡺 Use firewall-cmd to find out which are the active ‘zones’. What command do you use?

sudo firewall-cmd --get-active-zones



🡺 Have a look with the nft list ruleset command at the chains and rules that are created by firewalld.

A computer screen shot of white text

Description automatically generated

🡺 Use firewall-cmd to get information about the zone that is active on your interface. Get information about the ssh service in firewalld as well.

sudo firewall-cmd --zone=public --list-all

A computer screen with white text

Description automatically generated

🡺 The firewall-cmd provides a ‘panic’ mode. In this mode, all incoming and outgoing packets are dropped, active connections will expire. Enable this only if there are serious problems and well, if you’re panicking 😊. For example, if the machine is getting hacked in. Enable and disable this mode once. Have a look at what is being changed in your nftables by firewall-cmd.

Hint: save your nft ruleset once before and after panic mode and compare both files with the ‘diff’ command to find differences.)

sudo firewall-cmd --panic-on

sudo firewall-cmd --panic-f

diff panic-on panic-f

A screen shot of a computer

Description automatically generated

🡺 On RHEL, how is firewalld automatically started? Via a systemd service unit again, of course. It is known as firewalld.service . Have a look at its status. Is it active/enabled? Recap: have a look at [Install] section, it is ‘WantedBy’. Where is a symbolic link thus created when enabled?

A screenshot of a computer

Description automatically generated

It is enabled and active

🡺 On debian, we had a nftables.service. Is that service also available on RHEL? Can it coexist with firewalld? Have a look at the systemd configuration file of firewalld.service.

I do not think it is possible, since it has the nftables service as conflicts.

/etc/system/system/multi-user.target.wants/firewalld.service

## Additional application access control using TCP wrappers

Complementary to network firewalling with nftables (or any tool using these such as firewalld), we can use TCP wrappers to limit access on the application layer.

🡺 Via TCP wrappers on Debian, limit ssh access by only allowing 192.168.11.1 . This way only your laptop can SSH to Debian, and you won’t be able to SSH from e.g. RHEL.

A screenshot of a computer screen

Description automatically generated

Sshd: 192.168.11.1 >> /etc/hosts.allow

Sshd: ALL >> /etc/hosts.deny

🡺 The same method will not work in RHEL. Why?

Hint: use ldd

Probably it lacks some things that are being used. Since remember the fact that the services are even defined differently as systemd and everything.

## Additional application access control using systemd config files

🡺 Instead of using TCP wrappers, let’s then use IP access lists in systemd unit configuration files to limit ssh access to RHEL by only allowing 192.168.11.1 . Change the sshd.service in RHEL to make that happen. (The same is also possible in debian of course.)

A screen shot of a computer program

Description automatically generated

Horrible lab