

# Network Security

## Assignment 4, Wednesday 13 May 2020, version 1.2

**Handing in your answers:** Submission via Brightspace (<https://brightspace.ru.nl>)

**Deadline:** Wednesday 20 May 2020, 23:59:59 (midnight)

Please turn in all your work in plain text files (program source files are also plain text). If you prefer a document with formatting for whatever reason, like including images, use the PDF format to turn in your work (most editors allow you to export to PDF). Note that it's okay to include images separately and then refer to them from within the text files.

**Teaching assistants.** Please email *all* of us if you have a question.

- Pol Van Aubel <[pol.vanaubel@cs.ru.nl](mailto:pol.vanaubel@cs.ru.nl)>
- Daan Sprenkels <[d.sprenkels@cs.ru.nl](mailto:d.sprenkels@cs.ru.nl)>

**Tooling** For exercises 1 and 2, you will be using the following tools:

- iptables: <http://netfilter.org/projects/iptables/index.html>
- sshuttle: <https://github.com/sshuttle/sshuttle>

Do not compile these programs from source. iptables is likely installed by default. Use your Linux distribution's package manager to install the package sshuttle (e.g. Ubuntu/Debian: `apt install sshuttle`, Arch Linux: `pacman -S sshuttle`, Fedora: `yum install sshuttle`, Gentoo: `emerge sshuttle`). It seems that a couple of weeks ago, sshuttle was [removed](#) from the `kali-rolling` repositories. So on Kali, use `sudo pip3 install sshuttle` to install sshuttle.

Many commands in this exercise need to be run with root rights. This is denoted by a prefix `#`. When a command should be run without root rights, it will be prefixed with `$`. Do not include the prefix when typing the command.

You will not need to use some VM network that you needed for assignments 2 and 3. The only thing needed a linux machine with root rights, and you can use the Kali VM for this. Exercises 3 and 4 are purely theoretical, and no tools are required to solve these.

## Firewall configuration

1. In this exercise you will use `iptables` to create **two firewall configurations**: one for a **client** machine, one for a **masquerading server**. You are encouraged to test your configuration on your own (virtual) machine. You can use the commands `iptables-save` and `iptables-restore` to save and restore iptables rules to and from a file, respectively. Usage: `# iptables-save >filename` stores the firewall configuration in the file `filename`. `# iptables-restore <filename` restores the firewall configuration from `filename`. It might be a good idea to run `iptables-save` on the firewall configuration you have before starting this exercise.

If you get in unrecoverable trouble, you can completely reset the firewall configuration by running the following commands:

```
# iptables -F
# iptables -X
# iptables -t nat -F
# iptables -t nat -X
# iptables -t mangle -F
# iptables -t mangle -X
# iptables -P INPUT ACCEPT
# iptables -P FORWARD ACCEPT
# iptables -P OUTPUT ACCEPT
```

This will flush (-F) all built-in chains and delete (-X) all user-defined chains in the standard tables, and set the default policy (-P) to accept.

Create a folder called `exercise1` to hold the answers to this exercise.

Note that under some Linux distributions (most notably Ubuntu), you **may have to add a rule allowing traffic from localhost to localhost** in order to allow some local processes to communicate.

For this exercise, use the iptables manpages (`man iptables` and `man iptables-extensions`<sup>1</sup>), the netfilter documentation on <http://www.netfilter.org/documentation/index.html#documentation-howto> (especially the Packet Filtering HOWTO), the lecture slides, and any other sources you want.

(a) Build a client firewall that does the following:

- Allow all outgoing traffic.
- Deny all incoming traffic, except
  - traffic that belongs to an established connection, and
  - incoming SSH traffic (filter on transport protocol and port).
- Allow **all ICMP** traffic, **except ICMP** redirects.<sup>2</sup>

If you use tutorials or examples, please make sure you understand what the rules do.

Write your firewall configuration, preferably dumped by `# iptables-save`, to `exercise1a.fw`

---

<sup>1</sup>To get a list of all iptables-related `man` pages, use `man --apropos iptables` to find them.

<sup>2</sup>There are other ways to handle ICMP redirects securely. E.g. on Linux, there is the `/proc/sys/net/ipv4/conf/*/secure_redirects` directive, which tells the kernel to ignore all ICMP redirects that don't redirect to an already known gateway.

- (b) If you felt exercise 1a was easy, try your hand at this one. Otherwise, skip to exercise 2 and come back if you have time left over.

Build a firewall for a masquerading server / router with two network cards: `eth0` with IP address 198.51.100.42 and `eth1` with address 10.0.0.1/24. (Feel free to use other addresses, for example the addresses relevant to the Kali VM. But if you do, document them in your solution!) `eth1` is the internal card, the internal network should be obvious from its address. `eth0` is the external card, which is the link to the Internet. The firewall should do the following:

- Masquerade traffic coming from the local network going to the Internet.
- Allow outgoing traffic to the Internet that's forwarded from the local network.
- Block all outgoing traffic from the machine itself to the Internet, except for ICMP and traffic that belongs to an established connection.
- Allow all incoming ICMP traffic.
- Allow all outgoing traffic from the machine itself to the local network.
- Block all incoming traffic from the Internet, except traffic that belongs to an established connection.
- Accept incoming TCP connections on port 80 (let's say that's a webinterface) and port 22 from the Internet.
- Forward TCP and UDP traffic on port 2222 and 8080 to some other host in the local network. Feel free to pick that host yourself.

Since you likely cannot test this, simply give it your best shot. This is the type of configuration you'd use if you use a Linux machine as your home router.

Write your firewall configuration, preferably dumped by `# iptables-save`, to `exercise1b.fw`

## sshuttle

2. In this exercise you will use `sshuttle` to set up a secure tunnel to the lilo login server of the Faculty of Science, and then inspect and analyze the resulting iptables rules. For this, you will need to use your Faculty of Science account. If you do not have one, and do not have access to an alternative SSH server on which `sshuttle` works, send us an e-mail as soon as possible.

Use the manpage of `sshuttle` (`man sshuttle`) to figure out the command to route everything through the VPN. The remote host to use is `<username>@lilo.science.ru.nl`, where `<username>` is your Science login name. Write the command you use to `exercise2`. Remember to run `sshuttle` as root.

If the command consistently dies with `c : fatal: server died with error code 255`, you may need to add an exclusion rule for `lilo.science.ru.nl` (i.e. add `--exclude lilo.science.ru.nl` to your command).

Now, view the resulting iptables configuration using either `# iptables -t <table> -L` for each table listed in the manual page, or use `# iptables-save`.

- (a) Write the rules to `exercise2`, and explain what they do and how these rules work to route all the traffic through the VPN. Try e.g. looking for the port number you see in a listing of listening ports (hint: you can use `netstat` to do this).

Feel free to play with other configurations (e.g. routing only certain networks through the VPN, or using exceptions) and explain what the different firewall rules for these configurations do as well.

## Routing tables

3. Create a folder called `exercise3` to hold the answers for this exercise.

Although the lecture focusing on VPNs has not yet been given, there are some aspects which we can already cover in assignments. As you should have seen in exercise 2, `sshuttle` uses `iptables` to direct its traffic into the VPN process. There exists another type of VPN software that provides a virtual network interface to route traffic into. `OpenVPN` and `wireguard` are both of this type.

A machine's network stack decides what interface to put outgoing traffic into based on routes. Therefore, when using such a VPN, the routing table contains additional routes to route traffic over the VPN.

However, this is not as straightforward as simply adding a single route. Therefore, we will examine this approach in this exercise. Bear in mind that when a route has a `via` entry, this traffic is sent to that gateway over the specified interface, `dev`, for routing. If a route does not have a `via` entry, the traffic is sent directly to the target over the specified interface. A routing table therefore usually contains at least two routes: one that specifies how to reach the gateway, and one that specifies how to reach the rest of the internet through that gateway. The latter one is the `default route`.

For example, my IP address is `145.116.128.31/22`. When I'm not connected to my VPN, my routing table looks something like this:

```
$ ip route show
default via 145.116.128.1 dev wlp3s0
145.116.128.0/22 dev wlp3s0 proto kernel scope link src 145.116.128.31
```

For all the following questions, keep in mind that if routes overlap, the `more specific route` (i.e. one that applies to a smaller network, a smaller number of hosts) `overrides more generic routes`. So a route with a netmask of `/24` overrides a route covering the same hosts with a netmask of `/8`. The default route is effectively a route of `0.0.0.0/0`.

Let's say that my VPN runs on a machine with IP address `198.51.100.42`. When I connect to my VPN, a new interface (`tap0`) is created, and the routing table is changed (I have slightly altered the output for clarity):

```
$ ip route show
1. 0.0.0.0/1 via 10.50.9.1 dev tap0
2. 128.0.0.0/1 via 10.50.9.1 dev tap0
3. 10.50.9.0/24 dev tap0 proto kernel scope link src 10.50.9.60


4. 10.0.0.0/8 via 145.116.128.1 dev wlp3s0
5. 172.16.0.0/12 via 145.116.128.1 dev wlp3s0
6. 192.168.0.0/16 via 145.116.128.1 dev wlp3s0

7. default via 145.116.128.1 dev wlp3s0
8. 131.174.117.20 via 145.116.128.1 dev wlp3s0
9. 145.116.128.0/22 dev wlp3s0 proto kernel scope link src 145.116.128.31
10. 198.51.100.42 via 145.116.128.1 dev wlp3s0
```

Other relevant information is in the DHCP leases I got:

```
$ dhcpcd --dumplease wlp3s0
dhcp_server_identifier=131.174.117.20
domain_name_servers=131.174.117.20
ip_address=145.116.128.31
network_number=145.116.128.0
routers=145.116.128.1
subnet_cidr=22
subnet_mask=255.255.252.0
```

```
$ dhcpcd --dumplease tap0
dhcp_server_identifier=10.50.9.1
ip_address=10.50.9.60
network_number=10.50.9.0
subnet_cidr=24
subnet_mask=255.255.255.0
```



Internally the VPN uses the network 10.50.9.0/24, as can be seen in the DHCP lease for tap0. The DHCP protocol requires periodic communication with the DHCP server to keep the address lease active.

In these questions, when asked “where traffic goes”, please answer with the gateway IP address and the interface. If traffic is not sent to a gateway, answer with the direct IP address.

- (a) Look at routes 1, 2, and 7. Where does traffic not matched by any of the other routes go, and why?

Route 9 is one of the two original routes, also present when the VPN is not active. Routes 4–6 are always added by my VPN setup script. Note the IP ranges used, and try to imagine the usage scenario for a VPN.

- (b) Explain what these routes accomplish. What traffic do they match, where does that traffic go, and why?
- (c) Look at the DHCP lease for the tap0 interface. Explain why route 3 is necessary, in light of what routes 4–6 accomplish.
- (d) Look at route 8 and the DHCP lease for the wlp3s0 interface. Why is route 8 necessary? What happens if it is not present?
- (e) Route 10 is, from a functionality point of view, the most important route present. Explain what it does, and what would happen if it was *not* present.
- (f) Try to explain what happens when I connect e.g. to an SSH server running on the same machine as the VPN server. Does that traffic get tunneled or not? Explain why.

## Network address translation

4. Create a folder called **exercise4** for your answers.

Take a look at RFC 5508, “NAT Behavioral Requirements for ICMP” (<http://tools.ietf.org/rfc/rfc5508.txt>). Read sections 2 (especially the part on “ICMP Message Classification”), 3, and 4.

Answer the following 4 questions not from a security, but from a **technical point of view**.

- (a) What does a NAT do with **inbound** ICMP Error messages which *do* belong to an existing NAT session, and why?
- (b) **Why** does a NAT **drop inbound** ICMP Error messages which *do not* belong to an existing NAT session?
- (c) What does a NAT do with **outbound** ICMP Error messages which *do* belong to an existing NAT session, and why?
- (d) **Why** does a NAT drop **outbound** ICMP Error messages which *do not* belong to an existing NAT session?

Now read section 10, looking at 9 for the requirements.

- (e) Try to **summarize the security considerations of NAT for ICMP**. Explain the security concerns, and explain how they are mitigated.
- (f) Suppose an incompetent network administrator decides that ICMP is evil and just **blocks all ICMP** packets from entering and leaving the network. Give two examples of things that would break.

Section 9 mentions that a **NAT should not support ICMP Source Quench messages**. ICMP Source Quench is an ICMP message type which was proposed for congestion control in the early days of networking. A congested router would send a Source Quench message to hosts sending a lot of traffic in order to get them to back off. ICMP Source Quench messages were explicitly deprecated in RFC 6633, “Deprecation of ICMP Source Quench Messages” (<https://tools.ietf.org/html/rfc6633>).

- (g) Why is ICMP Source Quench **not an effective method** against distributed denial of service attacks against a router?
- (h) Think of a way in which you could have used ICMP Source Quench to perform a denial of service attack **against honest communicating hosts in a network**.

NAT has been used for years now to mitigate the shortage of IPv4 addresses in the world. While a lot of institutions are lagging behind in supporting IPv6<sup>3</sup>, mainstream Dutch ISPs are now gradually rolling out support for IPv6.

- (i) Why has NAT become **obsolete in the realm of IPv6?**

A couple of weeks ago, the Dutch ISP “KPN” enabled IPv6 on some of their retail “Experiabox” gateways. They **forgot to add new default firewall rules** when they enabled this.

- (j) **Why is this very dangerous** from a security perspective? What kind of firewall configuration is appropriate here? (Hint: What implicit protection did NAT provide?)

---

<sup>3</sup>Yes, looking at you, Radboud’s central IT department.

5. Place the files and directories `exercise1`, `exercise2`, `exercise3` and `exercise4`, and all their contents, in a folder called `netsec-assignment4-STUDENTNUMBER1-STUDENTNUMBER2`. Replace `STUDENTNUMBER1` and `STUDENTNUMBER2` by your respective student numbers, and accommodate for extra / fewer student numbers. Make a `tar.gz` archive of the whole `netsec-assignment4-STUDENTNUMBER1-STUDENTNUMBER2` directory and submit this archive in Brightspace.