Basics of Networking 2: Firewalls and NAT

Daniel STAN

ÉCOLE D'INGÉNIEURS EN INFORMATIQUE

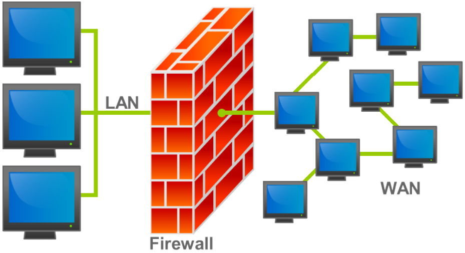
LABORATOIRE DE RECHERCHE DE L'EPITA

June 19, 2024

Firewall: Motivation

Main motivation: increase security

- Prevent **unauthorized connections**
- Rate **limit communications** (against DOS)
- Avoid **exploitation of vulnerabilities**.



NB: these do not prevent vulnerabilities in software, just **reduce the exposure**.

Goals for today

- (Briefly) analyze a machine/network surface of attack
- Write (small) firewall rules

We will illustrate all these concepts with the **linux firewall** system. More precisely: **iptables**.

Goals for today

- (Briefly) analyze a machine/network surface of attack
- Write (small) firewall rules

We will illustrate all these concepts with the **linux firewall** system. More precisely: **iptables**.

We will study connection tracking, as a **by-product** of the firewall system:

- How to track TCP sessions?
- How to use it to perform **Network Address Translation**?

You are at risk

List communications on a Linux system: `netstat -latupen`
(l=listen, u=udp, t=tcp, n=no-dns, p=processes)

```
$ sudo netstat —ltuanp
Active Internet connections (servers and established)
Proto Recv—Q Send—Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.1:9050          0.0.0.0:*               LISTEN      1807/tor
tcp        0      0 0.0.0.0:57621           0.0.0.0:*               LISTEN      1806861/spotify
tcp        0      0 192.168.170.217:45690   34.128.128.0:443        ESTABLISHED 835950/chrome ——typ
# ... 67 lines
```

Are all these communications legit, or should some of them be blocked?

Wireshark

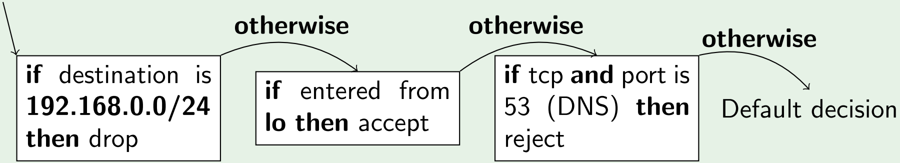
As opposed to `netstat`, `wireshark` also references communication not for our own machine, but also packets routed for others.

IPTables

`iptables` is a firewall linux system composed of

- Several **rules**, stored in a chain.
- Several **chains**, basically just a linked list, chains are stored in a table.
- Several **tables**: *filter, nat, mangle*.

Example

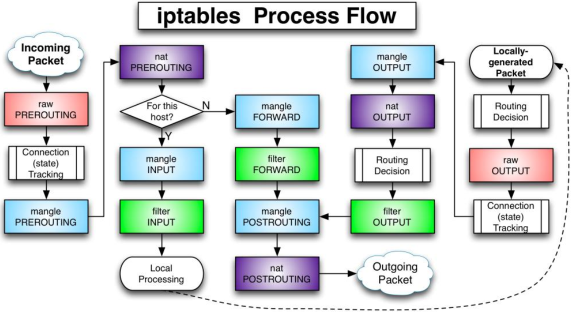


if destination is **192.168.0.0/24 then** drop
otherwise
if entered from **lo then** accept
otherwise
if tcp **and** port is 53 (DNS) **then** reject
otherwise
Default decision

Informal example of a chain made of 3 rules.

Each IP packet is processed by a chain, when some specific **network event** occurs.

Network Events: Flow of a packet in iptables



Ref: `https://westoahu.hawaii.edu/cyber/wp-content/uploads/2017/11/iptables-Flowchart.jpg`

- Locally processed = a process on the computer.
- Chains are written in UPPER CASE, tables in small case.
- We mostly use `filter` (default table).

The filter table

This is the default table, it contains three chains:

- INPUT: incoming packets
- OUTPUT: outgoing packet
- FORWARD: packets which are only passing by

The `filter` table

This is the default table, it contains three chains:

- `INPUT`: incoming packets
- `OUTPUT`: outgoing packet

Applicative firewall

- `FORWARD`: packets which are only passing by

Router's firewall

Decision: Targets

Taking a decision for a packet means to send the packet to another *special* chain:

- `ACCEPT` accepts the packet, without further reading the chain rules;
- `DROP` drops the packet, **silently**;
- `REJECT` same as `DROP` but also sends a message back (ICMP error) to the sender.

Rule of thumb: if you don't want to disclose your internal network, just drop packets, but rejecting is better for debugging purposes.

Decision: Targets

Taking a decision for a packet means to send the packet to another *special* chain:

- `ACCEPT` accepts the packet, without further reading the chain rules;
- `DROP` drops the packet, **silently**;
- `REJECT` same as `DROP` but also sends a message back (ICMP error) to the sender.

Rule of thumb: if you don't want to disclose your internal network, just drop packets, but rejecting is better for debugging purposes.
For example, to reject all routed packets through our machine:

```
# –A = appends (queue of the list), –j = jump
iptables –A FORWARD –j REJECT
```

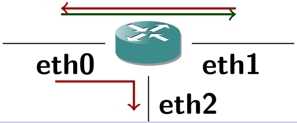
Conditions: Interfaces

Plenty of conditions: `https://linux.die.net/man/8/iptables`

Network Interfaces

`-i` (input) followed by an interface name specifies the interface used by the packet to enter the system, while `-o` (output) specifies the planned interface for outputting the packet.

```
# accept all packets routed from eth0 AND to eth1
iptables —A FORWARD —i eth0 —o eth1 —j ACCEPT
# reject all the rest
iptables —A FORWARD —j REJECT
```



Conditions: Interafces

Plenty of conditions: `https://linux.die.net/man/8/iptables`

Conditions: IP Address or Subnet

`--src` specifies the source IP address (or subnet) while `--dst` specifies the destination address (or subnet).

```
# throws away packets
#   ∗from eth0 AND
#   ∗with source IP in 192.168.0.0/16
iptables –A FORWARD ––src 192.168.0.0/24\
                    –i eth0 –j DROP
```

This takes the conjunction (AND). One can also take negations with `!` keyword.

stateless-conditions3

Conditions: Protocol

Transport Protocol

-p specifies a transport protocol (above IP), that is to say for example tcp, udp or icmp (ping). (one could also use the protocol number instead of its name).
If tcp or udp, one can specify ports with --dport or --sport.

```
# Let IP packets of type TCP pass by
iptables -I FORWARD -p tcp -j ACCEPT
# Authorize a SSH (22/tcp) server on this computer
iptables -I INPUT -p tcp --dport 22 -j ACCEPT
```

(optional) Custom Chains

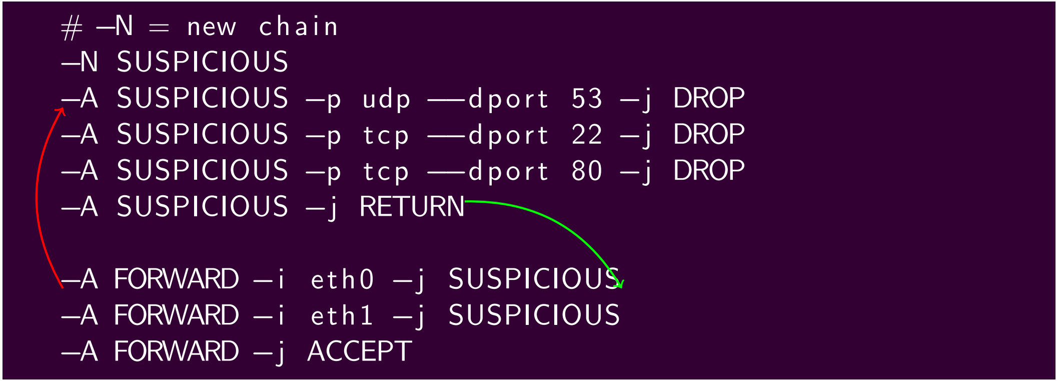
Chains are some kind of functions in a programming language, hence the `RETURN` target. Example of a custom chain that be re-used:

```
# -N = new chain
-N SUSPICIOUS
-A SUSPICIOUS -p udp --dport 53 -j DROP
-A SUSPICIOUS -p tcp --dport 22 -j DROP
-A SUSPICIOUS -p tcp --dport 80 -j DROP
-A SUSPICIOUS -j RETURN

-A FORWARD -i eth0 -j SUSPICIOUS
-A FORWARD -i eth1 -j SUSPICIOUS
-A FORWARD -j ACCEPT
```

(optional) Custom Chains

Chains are some kind of functions in a programming language, hence the `RETURN` target.
Example of a custom chain that be re-used:

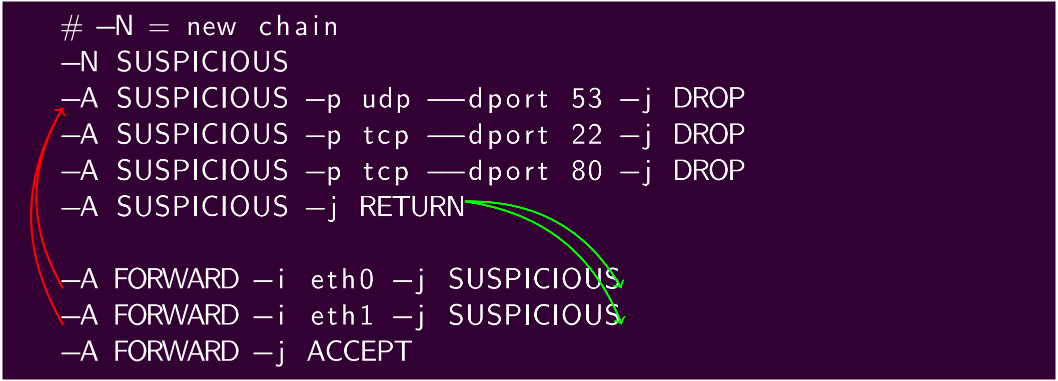


```
# -N = new chain
-N SUSPICIOUS
-A SUSPICIOUS -p udp --dport 53 -j DROP
-A SUSPICIOUS -p tcp --dport 22 -j DROP
-A SUSPICIOUS -p tcp --dport 80 -j DROP
-A SUSPICIOUS -j RETURN

-A FORWARD -i eth0 -j SUSPICIOUS
-A FORWARD -i eth1 -j SUSPICIOUS
-A FORWARD -j ACCEPT
```

(optional) Custom Chains

Chains are some kind of functions in a programming language, hence the `RETURN` target.
Example of a custom chain that be re-used:

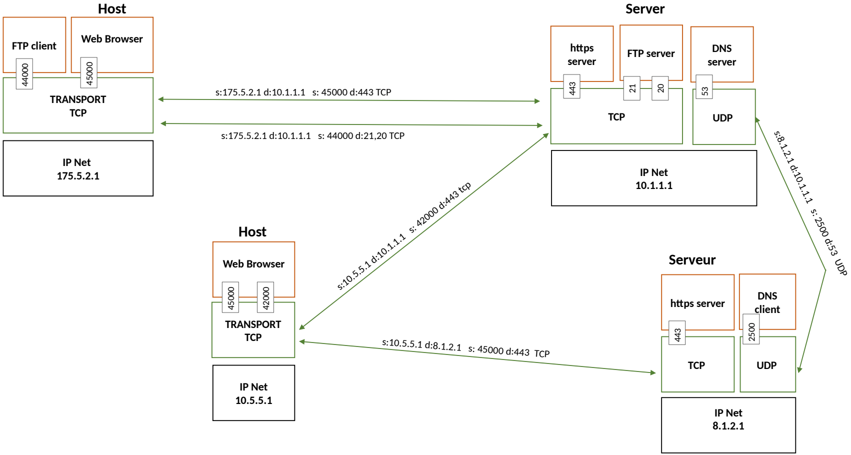


```
# —N = new chain
—N SUSPICIOUS
—A SUSPICIOUS —p udp ——dport 53 —j DROP
—A SUSPICIOUS —p tcp ——dport 22 —j DROP
—A SUSPICIOUS —p tcp ——dport 80 —j DROP
—A SUSPICIOUS —j RETURN

—A FORWARD —i eth0 —j SUSPICIOUS
—A FORWARD —i eth1 —j SUSPICIOUS
—A FORWARD —j ACCEPT
```

Stateful Firewalls

Recall: TCP sessions, made of several IP packets



Stateful vs Stateless

So far, we inspected IP packets individually, but they may be part of a TCP session, so it might be best to filter them together. This require the firewall to have some **memory** of what previous packets passed by:

- Stateless: filters packets **individually**;
- Stateful: some **memory**.

States with iptables

Condition: TCP/UDP state

With `-m state`, one can specify the state of the by `--state` and a set of possible connection states.

Possible state types: `NEW`, `ESTABLISHED`, `RELATED`, `INVALID`, `UNTRACKED`.

```
iptables -A FORWARD -m state \
            --state RELATED,ESTABLISHED -j ACCEPT

# This second expensive rule will be inspected
# only for new connections, and not for every packet
iptables -A FORWARD -m state --state NEW \
          -j EXPENSIVE_ANALYSIS
```

How a firewall records a TCP session?

On linux, `conntrack -L` displays the memory of the firewall, that is to say the connection tracking mechanism:

```
$ conntrack -L
#...
```

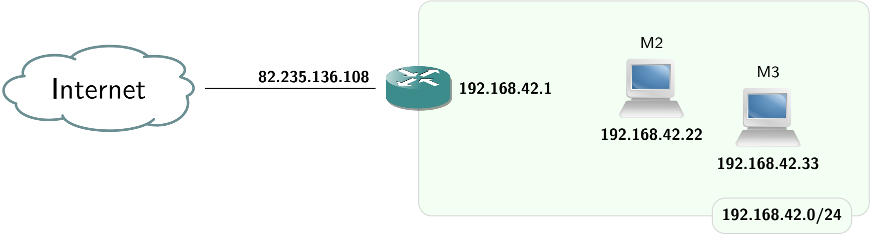
A connection is **uniquely identified** by 4 informations: (Source,Destination)×(IP, Port).
Recall: when connecting to a server, the destination port is known (80 for http, 443 for https, etc), but the source port is **chosen at random** by the OS.

Network Address Translation (NAT)

The quadruple (Source,Destination)×(IP, Port) is stored **twice** in `conntrack`: once for the **input** network interface and another for the **output** interface.
→ **NAT**: The firewall can *rewrite* IP addresses/ports, in a consistent way.

NAT: example and usage

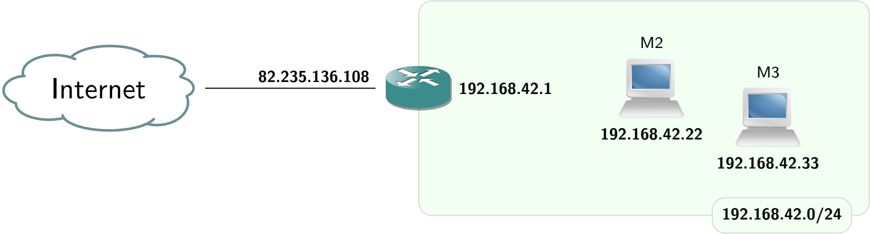
- Private addresses: **192.168.42.0/24**, cannot be used on the internet as they are **not unique**.
- Only a single public IP **82.235.136.108**.



Say M2 connects to http://**104.26.7.225**,

NAT: example and usage

- Private addresses: **192.168.42.0/24**, cannot be used on the internet as they are **not unique**.
- Only a single public IP **82.235.136.108**.

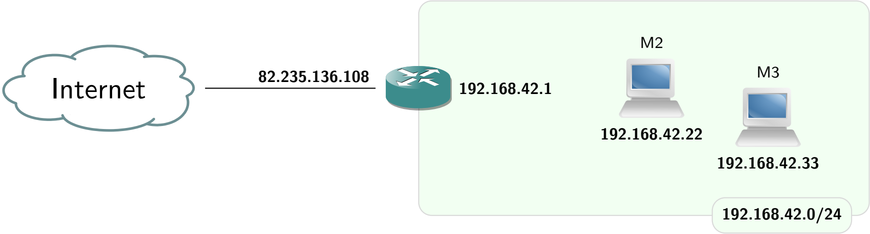


Say M2 connects to http://**104.26.7.225**,

- Query packet
  - On the local network: (**192.168.42.22**,**104.26.7.225**,15077,80)

NAT: example and usage

- Private addresses: **192.168.42.0/24**, cannot be used on the internet as they are **not unique**.
- Only a single public IP **82.235.136.108**.

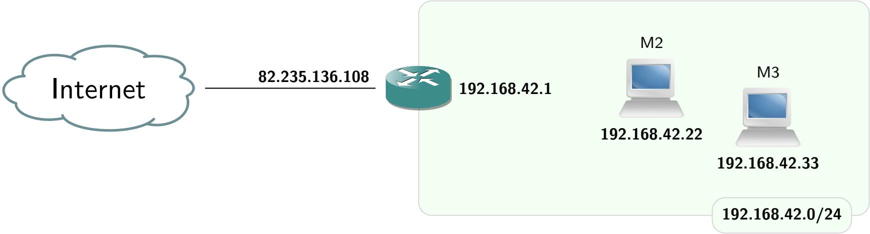


Say M2 connects to http://**104.26.7.225**,

- Query packet
  - On the local network: (**192.168.42.22**,**104.26.7.225**,15077,80)
  - On the internet, after NAT: (**82.235.136.108**,**104.26.7.225**,15077,80)

NAT: example and usage

- Private addresses: **192.168.42.0/24**, cannot be used on the internet as they are **not unique**.
- Only a single public IP **82.235.136.108**.

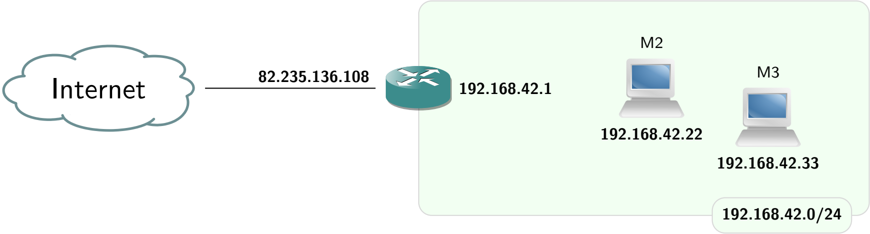


Say M2 connects to http://**104.26.7.225**,

- Answer from the HTTP server:
  - On the internet: (**104.26.7.225**,**82.235.136.108**,80,15077)

NAT: example and usage

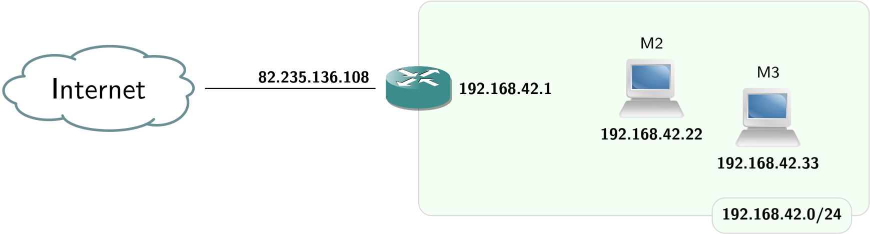
- Private addresses: **192.168.42.0/24**, cannot be used on the internet as they are **not unique**.
- Only a single public IP **82.235.136.108**.



Say M2 connects to http://**104.26.7.225**,

- Answer from the HTTP server:
  - On the internet: (**104.26.7.225**,**82.235.136.108**,80,15077)
  - On the local network, after NAT: (**104.26.7.225**,**192.168.42.22**,80,15077)

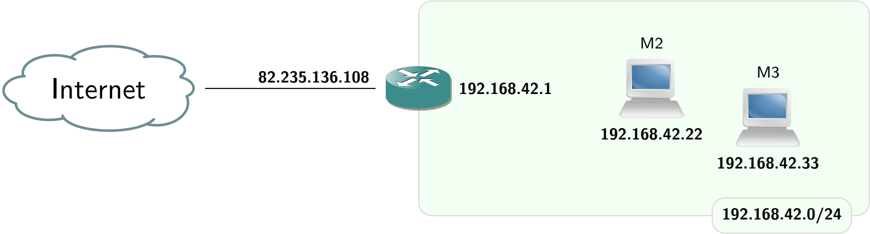
NAT: example and usage



Say M2 and M3 both connect to http://**104.26.7.225**:

- Query packets
  - On the local network (M2): (**192.168.42.22**,**104.26.7.225**,929,80)
  - (M3): (**192.168.42.33**,**104.26.7.225**,7207,80)

NAT: example and usage



Say M2 and M3 both connect to http://**104.26.7.225**:

- Query packets
  - On the local network (M2): (**192.168.42.22**,**104.26.7.225**,929,80)
  - (M3): (**192.168.42.33**,**104.26.7.225**,7207,80)
  - On the internet, after NAT: (**82.235.136.108**,**104.26.7.225**,929,80)
  - (**82.235.136.108**,**104.26.7.225**,7207,80)

NAT: example and usage

Say M2 and M3 both connect to http://**104.26.7.225**:

- Query packets
  - On the local network (M2): (**192.168.42.22**,**104.26.7.225**,929,80)
  - (M3): (**192.168.42.33**,**104.26.7.225**,7207,80)
  - On the internet, after NAT: (**82.235.136.108**,**104.26.7.225**,929,80)
  - (**82.235.136.108**,**104.26.7.225**,7207,80)
- Answers from the HTTP server:
  - On the internet: (**104.26.7.225**,**82.235.136.108**,80,929)
  - (**104.26.7.225**,**82.235.136.108**,80,7207)

NAT: example and usage

Say M2 and M3 both connect to http://**104.26.7.225**:

- Query packets
  - On the local network (M2): (**192.168.42.22**,**104.26.7.225**,929,80)
  - (M3): (**192.168.42.33**,**104.26.7.225**,7207,80)
  - On the internet, after NAT: (**82.235.136.108**,**104.26.7.225**,929,80)
  - (**82.235.136.108**,**104.26.7.225**,7207,80)
- Answers from the HTTP server:
  - On the internet: (**104.26.7.225**,**82.235.136.108**,80,929)
  - (**104.26.7.225**,**82.235.136.108**,80,7207)
  - On the local net, after NAT: (**104.26.7.225**,**192.168.42.22**,80,929)
  - (**104.26.7.225**,**192.168.42.33**,80,7207)

NAT: example and usage

Say M2 and M3 both connect to http://**104.26.7.225**:

- Query packets
  - On the local network (M2): (**192.168.42.22**,**104.26.7.225**,929,80)
  - (M3): (**192.168.42.33**,**104.26.7.225**,7207,80)
  - On the internet, after NAT: (**82.235.136.108**,**104.26.7.225**,929,80)
  - (**82.235.136.108**,**104.26.7.225**,7207,80)
- Answers from the HTTP server:
  - On the internet: (**104.26.7.225**,**82.235.136.108**,80,929)
  - (**104.26.7.225**,**82.235.136.108**,80,7207)
  - On the local net, after NAT: (**104.26.7.225**,**192.168.42.22**,80,929)
  - (**104.26.7.225**,**192.168.42.33**,80,7207)

Q: What if the same port is picked at random (*bad luck*)?

NAT: example and usage

Say M2 and M3 both connect to http://**104.26.7.225**:

- Query packets
  - On the local network (M2): (**192.168.42.22**,**104.26.7.225**,929,80)
  - (M3): (**192.168.42.33**,**104.26.7.225**,7207,80)
  - On the internet, after NAT: (**82.235.136.108**,**104.26.7.225**,929,80)
  - (**82.235.136.108**,**104.26.7.225**,7207,80)
- Answers from the HTTP server:
  - On the internet: (**104.26.7.225**,**82.235.136.108**,80,929)
  - (**104.26.7.225**,**82.235.136.108**,80,7207)
  - On the local net, after NAT: (**104.26.7.225**,**192.168.42.22**,80,929)
  - (**104.26.7.225**,**192.168.42.33**,80,7207)

Q: What if the same port is picked at random (*bad luck*)?

→ NAT can also **rewrite ports**

Linux: the `nat` table

Option `-t nat`, this table has four chains:

- `PREROUTING`, `INPUT`
- `POSTROUTING`,`OUTPUT`

Linux: the `nat` table

Option `-t nat`, this table has four chains:

- `PREROUTING`, `INPUT`
- `POSTROUTING`,`OUTPUT`

Three new targets:

- `SNAT`: rewrites the source IP together with `--to-source`
- `DNAT`: rewrites the destination IP of a session, together with `--to-destination`

Exercise/Question: what rule was used on the previous example?

Linux: the nat table

Option `-t nat`, this table has four chains:

- PREROUTING, INPUT
- POSTROUTING,OUTPUT

Three new targets:

- SNAT: rewrites the source IP together with `--to-source`
- DNAT: rewrites the destination IP of a session, together with `--to-destination`

Exercise/Question: what rule was used on the previous example?

```
iptables -t nat -A POSTROUTING --src 192.168.42.0/24 \
         -j SNAT --to-source 82.235.136.109
```

(optional) MASQUERADE target

`MASQUERADE` is a shortcut for `SNAT` where the source address is **automatically computed** using the routes.

Example

Assume *R1* has three IPs **192.168.42.1**, **82.235.136.108** and **10.0.0.1**.

It has the following routes:

```
default via 82.235.136.1 dev eth0 src 82.235.136.108
10.0.0.0/24              dev eth1 src 10.0.0.1
192.168.42.0/24          dev eth2 src 192.168.42.1
```

Then `-j MASQUERADE` is equivalent to:

```
-A POSTROUTING -o eth0 -j SNAT --to-source 82.235.136.108
```

(optional) MASQUERADE target

`MASQUERADE` is a shortcut for `SNAT` where the source address is **automatically computed** using the routes.

Example

Assume *R1* has three IPs **192.168.42.1**, **82.235.136.108** and **10.0.0.1**.

It has the following routes:

```
default via 82.235.136.1 dev eth0 src 82.235.136.108
10.0.0.0/24              dev eth1 src 10.0.0.1
192.168.42.0/24          dev eth2 src 192.168.42.1
```

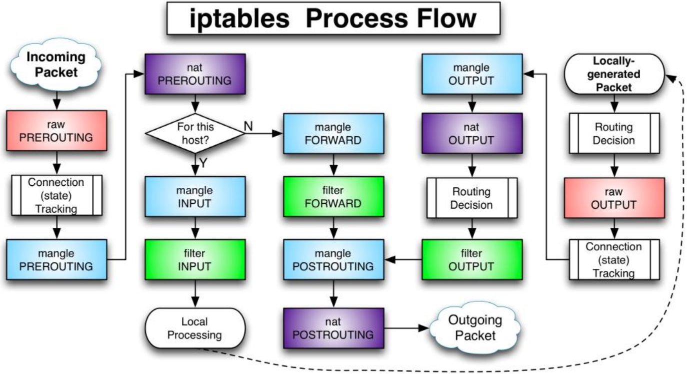
Then `-j MASQUERADE` is equivalent to:

```
-A POSTROUTING -o eth0 -j SNAT --to-source 82.235.136.108
-A POSTROUTING -o eth1 -j SNAT --to-source 10.0.0.1
-A POSTROUTING -o eth2 -j SNAT --to-source 192.168.42.1
```

Recall: Network Flow

- SNAT is to be used **after** routing decision: valid only in `POSTROUTING` and `OUTPUT`;
- DNAT is to be used **before** routing decision: valid only in `PREROUTING` and `INPUT`.

NB: this also means that `-i` cannot be used for `SNAT` and `-o` cannot be used for `DNAT`.



DNAT: MITM or caching

`DNAT`: used to intercept communications. For example, one could intercept some HTTP connection to serve it with a local server:

```
# Let's make 192.168.42.2 believe he's talking to ftp.debian.org:80
# while in fact, he's talking to me
-A PREROUTING --src 192.168.42.2 --dst 199.232.170.132 \
              -p tcp --dport 80 \
              -j DNAT --to-destination 10.0.0.1:80
```

Useful technique to introduce a **local mirrors**, **captive portals** on a Wifi hotspot, or to perform **Man-in-the-middle** attacks.
NB: doesn't work well with **authenticated communications**, for example `https`.

Exercise: NAT and conntrack

Analyze the following `conntrack -l` output, and give the different used NAT rules:

dev1	src1	dst1	dev2	src2	dst2
eth0	192.168.0.3:7331	146.75.118.132:80	**eth2**	82.235.136.108:7331	146.75.118.132:80
eth2	172.67.71.150:4242	82.235.136.108:2222	**eth0**	172.67.71.150:4242	192.168.0.2:22
eth2	172.67.71.150:4242	82.235.136.108:3333	**eth0**	172.67.71.150:4242	192.168.0.3:22

Exercise: NAT and conntrack

Analyze the following `conntrack -l` output, and give the different used NAT rules:

dev1	src1	dst1	dev2	src2	dst2
eth0	192.168.0.3:7331	146.75.118.132:80	**eth2**	82.235.136.108:7331	146.75.118.132:80
eth2	172.67.71.150:4242	82.235.136.108:2222	**eth0**	172.67.71.150:4242	192.168.0.2:22
eth2	172.67.71.150:4242	82.235.136.108:3333	**eth0**	172.67.71.150:4242	192.168.0.3:22

- SNAT

Exercise: NAT and conntrack

Analyze the following `conntrack -l` output, and give the different used NAT rules:

dev1	src1	dst1	dev2	src2	dst2
eth0	192.168.0.3:7331	146.75.118.132:80	**eth2**	82.235.136.108:7331	146.75.118.132:80
eth2	172.67.71.150:4242	82.235.136.108:2222	**eth0**	172.67.71.150:4242	192.168.0.2:22
eth2	172.67.71.150:4242	82.235.136.108:3333	**eth0**	172.67.71.150:4242	192.168.0.3:22

- SNAT

  ```
  --src 192.168.0.3 -j SNAT $PUB
  ```
- DNAT

Exercise: NAT and conntrack

Analyze the following `conntrack -l` output, and give the different used NAT rules:

dev1	src1	dst1	dev2	src2	dst2
eth0	192.168.0.3:7331	146.75.118.132:80	**eth2**	82.235.136.108:7331	146.75.118.132:80
eth2	172.67.71.150:4242	82.235.136.108:2222	**eth0**	172.67.71.150:4242	192.168.0.2:22
eth2	172.67.71.150:4242	82.235.136.108:3333	**eth0**	172.67.71.150:4242	192.168.0.3:22

- SNAT

  ```
  --src 192.168.0.3 -j SNAT $PUB
  ```

- DNAT

  ```
  --dst $PUB -p tcp --dport 2222 -j DNAT --to-destination 192.168.0.2:22
  ```

Exercise: NAT and conntrack

Analyze the following `conntrack -l` output, and give the different used NAT rules:

dev1	src1	dst1	dev2	src2	dst2
eth0	192.168.0.3:7331	146.75.118.132:80	**eth2**	82.235.136.108:7331	146.75.118.132:80
eth2	172.67.71.150:4242	82.235.136.108:2222	**eth0**	172.67.71.150:4242	192.168.0.2:22
eth2	172.67.71.150:4242	82.235.136.108:3333	**eth0**	172.67.71.150:4242	192.168.0.3:22

- SNAT

  ```
  --src 192.168.0.3 -j SNAT $PUB
  ```

- DNAT

  ```
  --dst $PUB -p tcp --dport 2222 -j DNAT --to-destination 192.168.0.2:22
  ```

- DNAT

Exercise: NAT and conntrack

Analyze the following `conntrack -l` output, and give the different used NAT rules:

dev1	src1	dst1	dev2	src2	dst2
eth0	192.168.0.3:7331	146.75.118.132:80	**eth2**	82.235.136.108:7331	146.75.118.132:80
eth2	172.67.71.150:4242	82.235.136.108:2222	**eth0**	172.67.71.150:4242	192.168.0.2:22
eth2	172.67.71.150:4242	82.235.136.108:3333	**eth0**	172.67.71.150:4242	192.168.0.3:22

- SNAT

  ```
  --src 192.168.0.3 -j SNAT $PUB
  ```
- DNAT

  ```
  --dst $PUB -p tcp --dport 2222 -j DNAT --to-destination 192.168.0.2:22
  ```
- DNAT

  ```
  --dst $PUB -p tcp --dport 3333 -j DNAT --to-destination 192.168.0.3:22
  ```

NB: typical setup with multiple SSH servers behind NAT, by using different ports on the public IP.

NAT Summary

- SNAT: heavily used to overcome the **IP shortage problem**:
  - Every customer gets only **a single** public IP for all its devices, the router (box) performs SNAT
  - Carrier-Grade NAT (CGNAT): same but at the ISP level.
- DNAT:
  - MITM attacks;
  - Captive Portals;
  - Host servers behind your ISP modem;

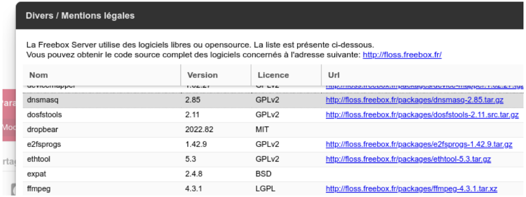
Recall: NAT requires tracking all connections, this can be become **very expensive** in a large network.

Epilogue

In this class and in lab sessions, you have seen:

- How to setup a local network with `ip a`, `ip r`
- Distribute IP addresses (DHCP) and provide DNS services with `dnsmasq`
- How to share a single IP address with `iptables`

This is exactly how most ISP modem work, on GNU/Linux, for example `http://floss.freebox.fr`:



Divers / Mentions légales

La Freebox Server utilise des logiciels libres ou opensource. La liste est présente ci-dessous.
Vous pouvez obtenir le code source complet des logiciels concernés à l'adresse suivante: http://floss.freebox.fr/

Nom	Version	Licence	Url
devicemapper	1.02.27	GPLv2	http://floss.freebox.fr/packages/device-mapper.1.02.27.tgz
dnsmasq	2.85	GPLv2	http://floss.freebox.fr/packages/dnsmasq-2.85.tar.gz
dosfstools	2.11	GPLv2	http://floss.freebox.fr/packages/dosfstools-2.11.src.tar.gz
dropbear	2022.82	MIT	
e2fsprogs	1.42.9	GPLv2	http://floss.freebox.fr/packages/e2fsprogs-1.42.9.tar.gz
ethtool	5.3	GPLv2	http://floss.freebox.fr/packages/ethtool-5.3.tar.gz
expat	2.4.8	BSD	
ffmpeg	4.3.1	LGPL	http://floss.freebox.fr/packages/ffmpeg-4.3.1.tar.xz