



UNIVERSITY OF TRENTO

NETWORK SECURITY REPORT

Firewall

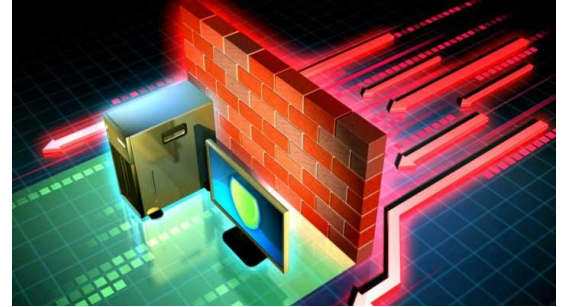
Bojan Poletan

Mirko Ioris

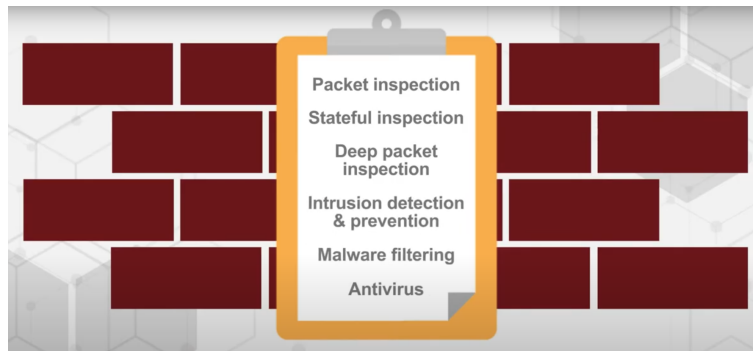
Martin Huszti

What is a Firewall?

A firewall is a **network security system**, placed between the public internet and the private network. It is usually depicted as a wall that **blocks** incoming traffic. We can **filter** this traffic by writing specific rules in the firewall, called **policies**. This way we are able to safeguard our network and let only safe packets through.



There are 5 types of firewall



Packet filtering firewall: These firewalls operate at the network layer of the OSI model. They make decisions based on network addresses, ports and protocols and are also quite fast, because there is not much logic behind the decision they make. They do not do any inspection of the traffic and do not store any state information. Packet-filtering firewalls are considered less secure than others, since they forward any traffic that flows on an approved port and could be malicious.

Circuit-level gateway. It verifies TCP handshakes to check the traffic without consuming a lot of time and resources, but it still does not check information the packets are carrying.

Stateful inspection firewall: It is also called dynamic packet filtering. It tracks the state of sessions and makes allow/deny decisions based on a session state table. It offers more security than the previous two, but it lessens the network performance.

Application-level gateway: It filters packets regarding their intended service, so the rules can be applied for certain services like HTTP. It operates on the seventh level, the application layer of the OSI model. These firewalls can be a little slower than others, since the amount of information that is being processed is bigger.

Next-generation firewalls: These kinds of firewalls offer packet and stateful inspection, malware filtering, antivirus, intrusion detection and many other features.

Physical and virtual firewalls

We can classify firewalls into two categories: physical and virtual. Both are network traffic filters but the physical one is a **hardware machine**, while the virtual one is a **software application**. The first one is usually more expensive, since you have to buy a dedicated machine, and it is an independent component in the network, like a router. It has the ability to handle **heavy traffic**. The second is used to protect **cloud services** and in virtual environments. It is cheaper than the physical one because it is just a program. It is also easy to set up, configure and maintain.

Stateless vs Stateful firewalls

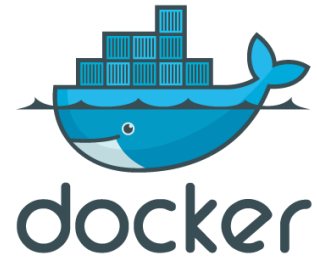
Depending on whether a firewall has the ability to remember previous events, we call them stateless or stateful firewalls. Stateful firewalls are more **powerful** and **context-sensitive**.

For example regarding password input, if we talk about a stateless method we check the password, and if it is incorrect after 5 or more times we do not do anything because we do not remember the fact that it was incorrect 5 times before.

If we talk about a stateful method, after 5 attempts, we can think that somebody is trying to hack or crack the password so we have to lock the account or notify the user about this. This is called a stateful approach.

If we go up in the application stack, things tend to be more stateful, while moving down more stateless.

Virtual Machine structure and setup



The base virtual machine is a Debian Buster, which is currently the newest version of Debian. It has a basic user, which password is **user** as well. The superuser password is **password** to be remembered easily, but the basic user has all the privileges to execute commands via sudo.

The virtual machine has a pre-installed and configured **docker engine** that we used to simulate different machines by creating various docker containers. We installed an **alpine linux image** in each machine because it is a lightweight and secure linux distribution. The whole image is **5 megabytes** at all and is perfect for the purpose of the course.

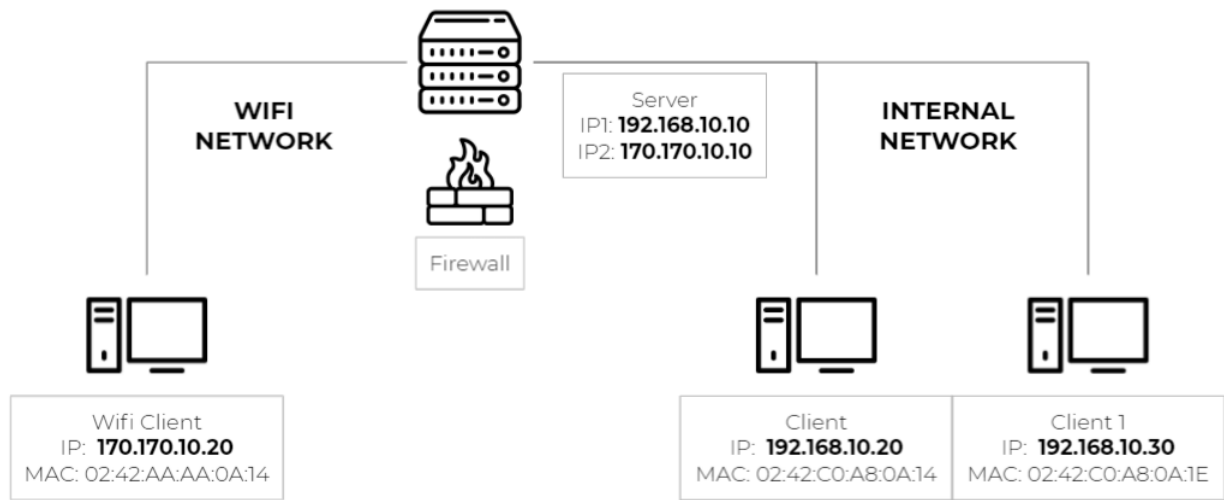
With the command **docker containers list -a** or **docker ps -a**, (a means all) you can see that there are 4 main containers, namely **client**, **client1**, **clientWifi** and **server**. The main difference between the clients and server is that only the server has the iptables installed.

Docker allows us to create **networks** as well and to connect containers to each other. There are three standard networks called *bridge*, *host* and *none*, but we created two new networks: **INTERNAL** and **WIFI NETWORK**.

```
File Edit View Search Terminal Help
root@debian:/home/user/Desktop# docker network list
NETWORK ID          NAME                DRIVER              SCOPE
a663994bf7a6        INTERNAL_NETWORK    bridge              local
7ac77f21835c        WIFI_NETWORK        bridge              local
5c8949f5f01c        bridge              bridge              local
03b7ea12de3a        host                host                local
7b38f5a93d00        none                null                local
root@debian:/home/user/Desktop#
```

Fun fact: You can see that server and client are based on the image alpine, and the other two clients are based on something else. It is because firstly we configured and installed all the necessary packages in the client image, then we cloned that one, therefore we were able to duplicate it. Actually there is no difference in the operating systems between them.

Our network topology is the following:



We have a server with two different interfaces to connect to two subnets. The WiFi network has one client while the internal network has two different machines inside. The firewall is placed under the server to show that they are on the same machine.

If you run the command **docker ps**, you can see that currently there are no running containers, so first we have to start them. To do it, give the command **docker start <container name>** and the **docker attach <container name>** to attach it on the current terminal and use the machine inside. To simplify things we created a script for running all the containers at once that also renames terminals.

This script is called **startup.sh** and located in the desktop folder of the user. Navigate to the desktop folder and type **./startup.sh** to execute it.

```
File Edit View Search Terminal Help
user@debian:~$ cd Desktop/
user@debian:~/Desktop$ ls
startup.sh
user@debian:~/Desktop$ ./startup.sh
```

Note: The predefined keyboard layout is English but you can easily change it in the settings.

Iptables - A bit of theory

As a firewall we are using **iptables**. It is an **ipv4 packet filter** (there is a version for ipv6 too) integrated into the linux kernel and part of a much wider project called **netfilter**. It has a text interface but there are also many GUIs available. Since it is implemented at kernel level it is very flexible and there is a wide availability of external modules.

Iptables is composed of **tables** that contain **chains** that contain **rules**.

By default there are three tables:

- **Filter table**: the default table and the most important
- **Nat table**: used for package alteration, for example when you want to forward a package to another port or machine
- **Mangle table**: used for package header modification

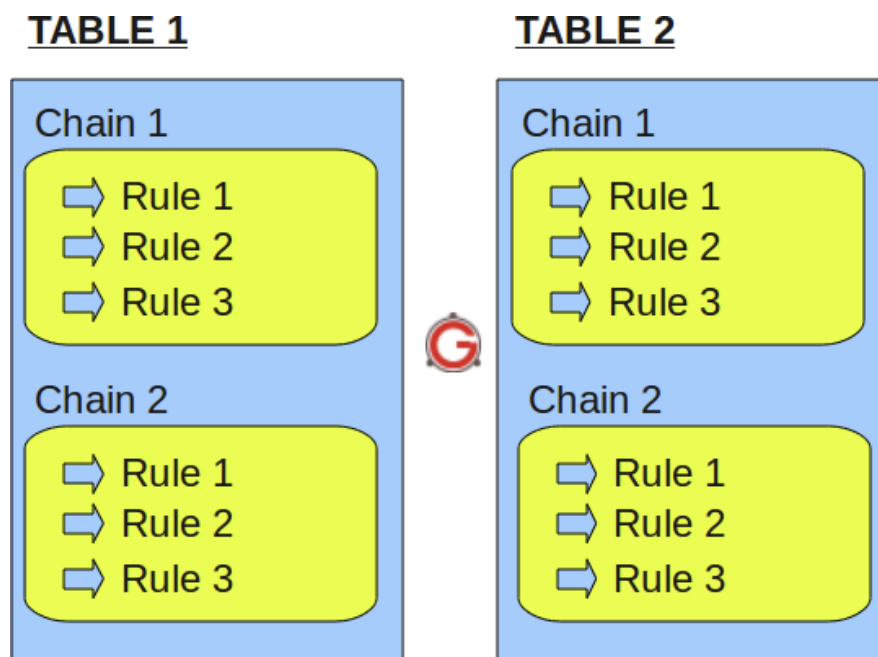
Within each table there are chains. The most common are INPUT, OUTPUT and FORWARD but the number is not limited and you can also add custom chains.

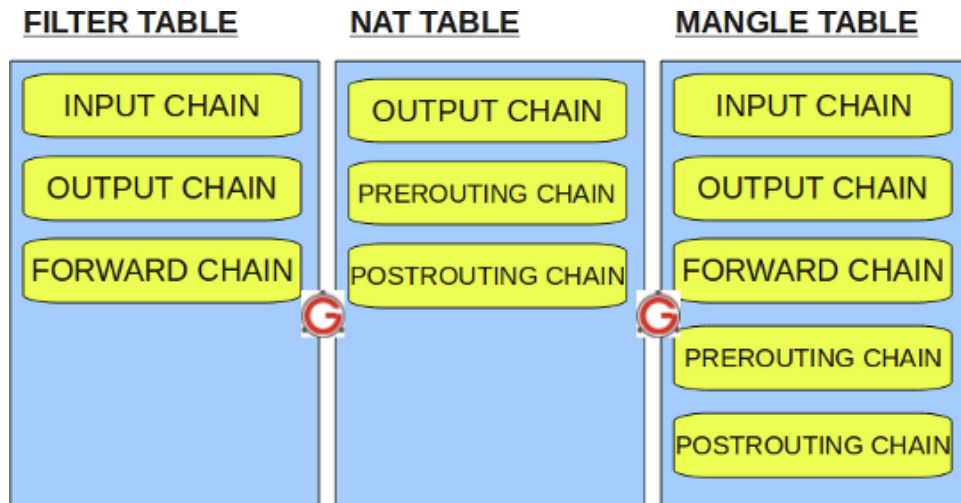
A rule is a written condition composed of several parameters like:

- Source/destination address
- Source/destination port
- Protocol
- And much more

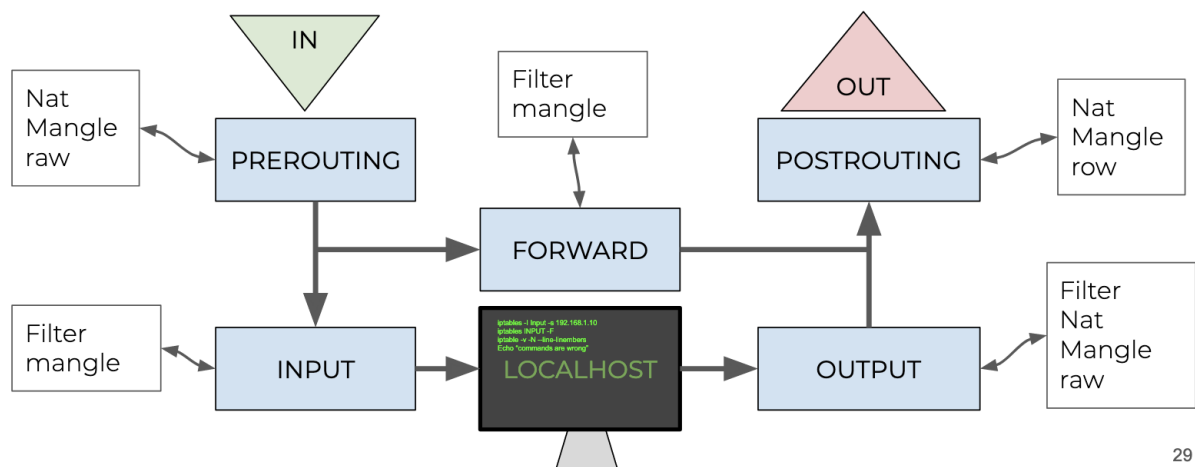
If a packet matches a certain rule, it is processed according to that.

A graphical representation is the following:





And this is how all components are connected together:



29


How it works:

1. The package that enters in a chain will be matched against the first rule



permission	protocol	src	dest	port
ACCEPT	TCP	any	192.168.1.25	80
ACCEPT	TCP	any	192.168.1.25	443
ACCEPT	UDP	any	192.168.1.10	53
REJECT	UDP	any	any	

- If it doesn't match the condition it will be matched against the next rule and so on



permission	protocol	src	dest	port
ACCEPT	IP	any	192.168.1.25	80
ACCEPT	IP	any	192.168.1.25	443
ACCEPT	UDP	any	192.168.1.10	53
REJECT	UDP	any	0.0.0.0/0	

until the end.

- Each table has a **default policy** and it is applied in cases where a packet does not match any rule in the table and reaches the end. We must process these packets as well and usually a good practice is to DROP them.

Iptables commands are composed of many parts, the following picture shows a summary:

	Table name	command	chain	parameter	target/rule
iptables		-A			
		-D		-p	
		-I		-s	
	-t filter	-R	INPUT	-d	-j ACCEPT
	-t nat	-L	FORWARD	-i	-j DROP
	-t mangle	-F	OUTPUT	-o	-j REJECT
	-t raw	-Z	PREROUTING	--sport	...
		-N	POSTROUTING	--dport	
		-X		...	
		-P			

You can use the various columns to compose the command you need. How?

- Chose one table (if not specified the tab *filter* will be used)
- Chose one command
- Chose one chain
- Scose one or more parameters
- Chose a target

This is an example:

```
iptables -t filter -A INPUT -P -J ACCEPT
```

An important command is **iptables -L** which shows us the rule list. We can add many parameters to see different details:

```
iptables -L -v -n --line-numbers
```

- **-L** *to see the rule list*
- **-v** *a verbose output for interface names, rule options, pkg and byte counters*
- **-n** *IP addresses and ports will be printed in numerical format*
- **--line-numbers** *to display the line numbers of each rule*

We can divide rules in two categories, inserting and deleting.

Inserting rules:

- **Iptables -A [chain]** *//append a new rule in the chain*
- **Iptables -I [chain] [position]** *//add a new rule in the specified position*
- **Iptables -P [chain] [rule]** *//applies a new rule in the chain as default policy*
- **Iptables -N [chain]** *//create a custom chain*

Deleting rules:

- **Iptables -F [chain]** *//deletes all rules in the chain or in all chains*
- **Iptables -R [chain] [position] [rule]** *//rewrite a rule in a specific position*
- **Iptables -D [chain] [position]** *//delete the selected rule in the chain*
- **Iptables -X [custom chain]** *//delete a custom chain*

Some examples:

```
iptables -A INPUT -p tcp --dport 8080 -j ACCEPT
```

Appends a rule in the input chain to accept tcp traffic on destination port 8080.

```
iptables -I INPUT 2 -p tcp --dport 8081 -j ACCEPT
```

Inserts a rule in the second position in the input chain to accept tcp traffic on port 8081.

```
iptables -D INPUT 1
```

Deletes the rule in the first position.

Exercise 1

Reject all the incoming requests of any type

Let's do some exercises with iptables. The basic scenario is that we are a company and we do not want to receive any icmp messages from our network.

A way is to block all incoming traffic. For that we will change the table policy from accept to deny. If we try to ping the server from the client before applying the policy everything works fine. All the packets arrived successfully.

Now we will change the default policy with the following command:

iptables -P INPUT DROP

All the packets intercepted by the input chain are now dropped by default. We can test it by pinging the server again and see that all the packets are lost.

server - 192.10.168.10 / 170.170.10.10	client - 192.168.10.20
<pre>File Edit View Search Terminal Help Chain INPUT (policy ACCEPT) target prot opt source destination Chain FORWARD (policy ACCEPT) target prot opt source destination Chain OUTPUT (policy ACCEPT) target prot opt source destination / # iptables -P INPUT DROP / # iptables -L Chain INPUT (policy DROP) target prot opt source destination Chain FORWARD (policy ACCEPT) target prot opt source destination Chain OUTPUT (policy ACCEPT) target prot opt source destination / #</pre>	<pre>File Edit View Search Terminal Help / # ping 192.168.10.10 -c 1 PING 192.168.10.10 (192.168.10.10): 56 data bytes 64 bytes from 192.168.10.10: seq=0 ttl=64 time=0.110 ms --- 192.168.10.10 ping statistics --- 1 packets transmitted, 1 packets received, 0% packet loss round-trip min/avg/max = 0.110/0.110/0.110 ms / # ping 192.168.10.10 -c 1 PING 192.168.10.10 (192.168.10.10): 56 data bytes ^C --- 192.168.10.10 ping statistics --- 1 packets transmitted, 0 packets received, 100% packet loss / #</pre>

Blocking all the incoming packets is too extreme, we will try now to block just icmp messages.

Exercise 2

Add a policy to **REJECT** traffic from icmp port in the server

First let's restore the default accept policy:

iptables -P INPUT ACCEPT

And now add a rule to reject the incoming traffic only on icmp port:

iptables -A INPUT -p icmp -j REJECT

```
server
File Edit View Search Terminal Help
Chain INPUT (policy ACCEPT)
/ # iptables -A INPUT -p icmp -j REJECT
/ # iptables -L
Chain INPUT (policy ACCEPT)
target      prot opt source                destination          reject-with icmp-port-unreachable
REJECT      icmp -- anywhere              anywhere
Chain FORWARD (policy ACCEPT)
target      prot opt source                destination
Chain OUTPUT (policy ACCEPT)
target      prot opt source                destination
/ #
```

Exercise 2.1


Now try to write a rule to accept ICMP packages.

Is it working?

If yes: well done.

If not: can you guess why?

This is your current situation:



permission	protocol	src	dest	port
REJECT	icmp	any	any	
ACCEPT	icmp	any	any	

If the packet matches the first condition, it will be processed and will be stopped in the chain, so it means that it will never reach the second rule you inserted.



permission	protocol	src	dest	port
REJECT	icmp	any	any	
ACCEPT	icmp	any	any	

Iptables - parameters

As said earlier in every rule we can insert a set of parameters. For example:

parameter	description	specified
-p	protocol	tcp, udp, icmp, /etc/protocols, all
-s	src addr	network name, hostname, subnet (192.168.1.0/24 or 192.168.1.0/255.255.255.0), ip address
-d	Dest addr	
-i	Input interface	interface name (eth0)
-o	Output interface	
--sport	Src port	service name, port number, port range(start:end)
--dport	Dest port	

Let's see them in detail:

- **-s [!][source]** // specify the source IP/mask/hostnames
 - 192.168.1.100 or 192.168.1.0/24 or 192.168.1.0/255.255.255.0
 - www.hostname.com
 - All or 0.0.0.0/0
- **-d [!][destination]** // specify the destination IP/mask, ex. -d 192.168.1.100
- **-p [!][protocol]** // specify the protocol to which we want to apply the rule among tcp, udp, icmp or all in /etc/protocols
- **-i [!][interface]** // specify the input interface (eth+ will include all eth interface but will exclude others)
- **-o [!][interface]** // specify the output interface
- **--sport [port:port]** // source port/port range (max range)

- **--dport [port:port]** *// destination port/port range*
 - `--dport 22` or `--dport 20:40`
- **--tcp-flags [FLAG]** *// TCP flags like: SYN, ACK, FIN, RST, URG, PSH, ALL, NONE*
- **--syn** *// packages with only SYN active (new connections)*
- **--tcp-option [option]**

Question: How can you drop all tcp packages with SYN flag on arriving on one of 1024 dedicated ports?

Solution:

```
iptables -I INPUT -p tcp --syn --dport 0:1024 -j DROP
```

Iptables - Logging

We can use the target **LOG** in a rule to activate kernel logging of matching packets. The log is a **non-terminating target** and must be followed by another rule to **handle packets**, usually a reject or drop rule. It can have some parameters like the following:

- **--log-level [level]** *// specifies the level of logging, there are 7 different levels (debug, info, notice, warning, err, crit, alert, emerg), the default is level 4*
- **--log-prefix "prefix"** *// to insert a message before the log, up to 29 letters long*
- **--log-tcp-options** *// to log different tcp options from packet headers*

Exercise 3

Reject tcp and udp streams coming from client 192.168.10.20 with ICMP port unreachable messages but LOG all these packets.

Hint: Create a new chain for logging and send all packets there. Then log and drop them.

We can use **netcat** to send tcp and udp packets to the server, before and after applying the rule and note the difference.

On the server write:

```
nc -l -p 3030 (to listen tcp packets)
nc -u -l -p 3030 (to listen udp packets)
```

On the client write:

nc 192.168.10.10 3030 (to send tcp packets)

nc -u 192.168.10.10 3030 (for udp packets)

And then the message you want to send.

To reject packets with a specific message we can use REJECT with an option. The syntax is the following:

REJECT --reject-with [option]

- host-unreachable
- **icmp-port-unreachable** *//we are interested in this*
- icmp-proto-unreachable
- icmp-net-prohibited
- icmp-host-prohibited
- icmp-admin-prohibited

First of all we should create a new chain for logging:

iptables -N LOGCHAIN

Then we can send packets there:

iptables -A INPUT -p udp -s 192.168.10.20 -j LOGCHAIN

iptables -A INPUT -p tcp -s 192.168.10.20 -j LOGCHAIN

We use LOG to log them:

iptables -A LOGCHAIN -p udp -j LOG --log-prefix "UDP streams logged packets: "

iptables -A LOGCHAIN -p tcp -j LOG --log-prefix "TCP streams logged packets: "

And finally we reject them

iptables -A LOGCHAIN -j REJECT --reject-with icmp-port-unreachable

Using netcat again will show us that the connection is closed after the first message sent because we receive a response that the port is unreachable.

Unfortunately we are not able to view the log file because, for security reasons, iptables does not save it inside docker containers. This avoids a possible DDos attack on the host. To see what the log file looks like we will do another exercise.

Exercise 4

Block the site www.ciao.it from Debian and log the request.

To use iptables in the Debian terminal we have to fix the path by typing this command:

```
PATH=/sbin/:$PATH
```

Add a new LOG chain and block the site:

```
iptables -N LOGGING
```

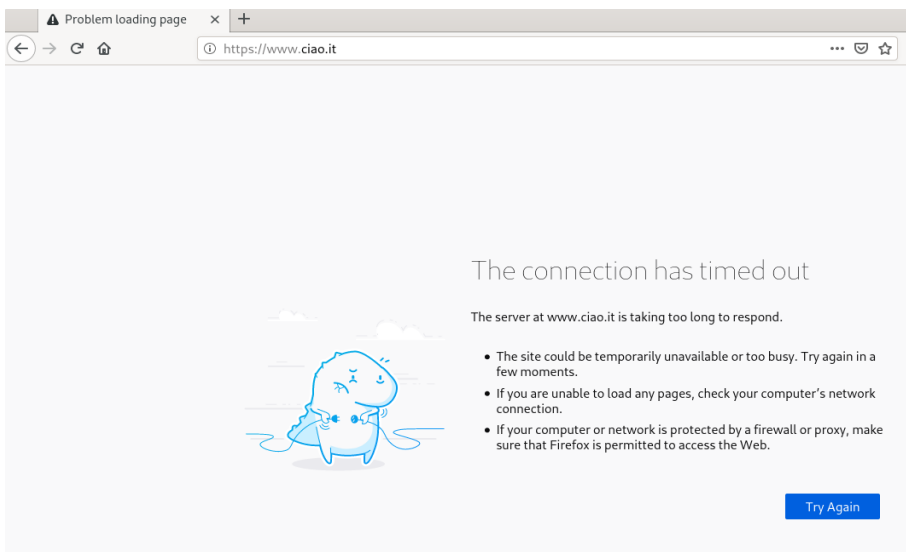
```
iptables -A OUTPUT -p tcp -d www.ciao.it -j LOGGING
```

```
iptables -A LOGGING -d www.ciao.it -j LOG --log-prefix "Site ciao blocked:"
```

```
iptables -A LOGGING -d www.ciao.it -j DROP
```

```
File Edit View Search Terminal Help
EC=0x00 TTL=64 ID=12625 DF PROTO=TCP SPT=56642 DPT=443 WINDOW=62780 RES=0x00 ACK PSH URGP=0
[ 313.609342] Site ciao blocked: IN= OUT=enp0s3 SRC=10.0.2.15 DST=143.204.11.56 LEN=134 TOS=0x00 PR
EC=0x00 TTL=64 ID=12626 DF PROTO=TCP SPT=56642 DPT=443 WINDOW=62780 RES=0x00 ACK PSH URGP=0
[ 315.272410] Site ciao blocked: IN= OUT=enp0s3 SRC=10.0.2.15 DST=143.204.11.56 LEN=134 TOS=0x00 PR
EC=0x00 TTL=64 ID=12627 DF PROTO=TCP SPT=56642 DPT=443 WINDOW=62780 RES=0x00 ACK PSH URGP=0
[ 318.662764] Site ciao blocked: IN= OUT=enp0s3 SRC=10.0.2.15 DST=143.204.11.56 LEN=134 TOS=0x00 PR
EC=0x00 TTL=64 ID=12628 DF PROTO=TCP SPT=56642 DPT=443 WINDOW=62780 RES=0x00 ACK PSH URGP=0
[ 325.315617] Site ciao blocked: IN= OUT=enp0s3 SRC=10.0.2.15 DST=143.204.11.56 LEN=134 TOS=0x00 PR
EC=0x00 TTL=64 ID=12629 DF PROTO=TCP SPT=56642 DPT=443 WINDOW=62780 RES=0x00 ACK PSH URGP=0
[ 332.268940] Site ciao blocked: IN= OUT=enp0s3 SRC=10.0.2.15 DST=143.204.11.56 LEN=173 TOS=0x00 PR
EC=0x00 TTL=64 ID=12630 DF PROTO=TCP SPT=56642 DPT=443 WINDOW=62780 RES=0x00 ACK PSH URGP=0
[ 338.620757] Site ciao blocked: IN= OUT=enp0s3 SRC=10.0.2.15 DST=143.204.11.56 LEN=173 TOS=0x00 PR
EC=0x00 TTL=64 ID=12631 DF PROTO=TCP SPT=56642 DPT=443 WINDOW=62780 RES=0x00 ACK PSH URGP=0
[ 340.267757] Site ciao blocked: IN= OUT=enp0s3 SRC=10.0.2.15 DST=143.204.11.56 LEN=212 TOS=0x00 PR
EC=0x00 TTL=64 ID=12632 DF PROTO=TCP SPT=56642 DPT=443 WINDOW=62780 RES=0x00 ACK PSH URGP=0
[ 340.267868] Site ciao blocked: IN= OUT=enp0s3 SRC=10.0.2.15 DST=143.204.11.56 LEN=212 TOS=0x00 PR
EC=0x00 TTL=64 ID=12633 DF PROTO=TCP SPT=56642 DPT=443 WINDOW=62780 RES=0x00 ACK PSH URGP=0
[ 340.267876] Site ciao blocked: IN= OUT=enp0s3 SRC=10.0.2.15 DST=143.204.11.56 LEN=236 TOS=0x00 PR
EC=0x00 TTL=64 ID=12634 DF PROTO=TCP SPT=56642 DPT=443 WINDOW=62780 RES=0x00 ACK PSH URGP=0
[ 340.267920] Site ciao blocked: IN= OUT=enp0s3 SRC=10.0.2.15 DST=143.204.11.56 LEN=236 TOS=0x00 PR
EC=0x00 TTL=64 ID=12635 DF PROTO=TCP SPT=56642 DPT=443 WINDOW=62780 RES=0x00 ACK PSH FIN URGP=0
[ 365.999274] Site ciao blocked: IN= OUT=enp0s3 SRC=10.0.2.15 DST=143.204.11.56 LEN=236 TOS=0x00 PR
EC=0x00 TTL=64 ID=12636 DF PROTO=TCP SPT=56642 DPT=443 WINDOW=62780 RES=0x00 ACK PSH FIN URGP=0
root@debian:/home/user/Desktop#
```

Try to visit the site. You will no longer be able to access it because you are dropping your own request with this rule. The logs are visible in the `/var/log/messages` file or they can be easily accessed by typing **dmesg** (as sudo).



In the first picture we can see the saved logs with our custom message and in the second we are trying to reach the site but we get a connection timeout error.

Exercise 5 - Stateful firewalls

We want to allow only established connection with the server and allow new connections only for telnet on port 23.

With iptables we are able to configure **stateful firewalls** and add a **connection state** information to the rule to better identify packets. There are four states:

- **ESTABLISHED** *//pkg related to an already existing connection*
- **NEW** *//pkg related to a new connection*
- **RELATED** *//pkg related to a new connection but linked to an existing one*
- **INVALID** *//pkg related to unknown connection*

To use it we specify the parameter:

-m state --state [state]

The state is an explicit match and must be called with **-m**. Along with it there are many other matches and we will see some of them in the next exercises.

To allow only enstablished connection with the server we can use the following rule:

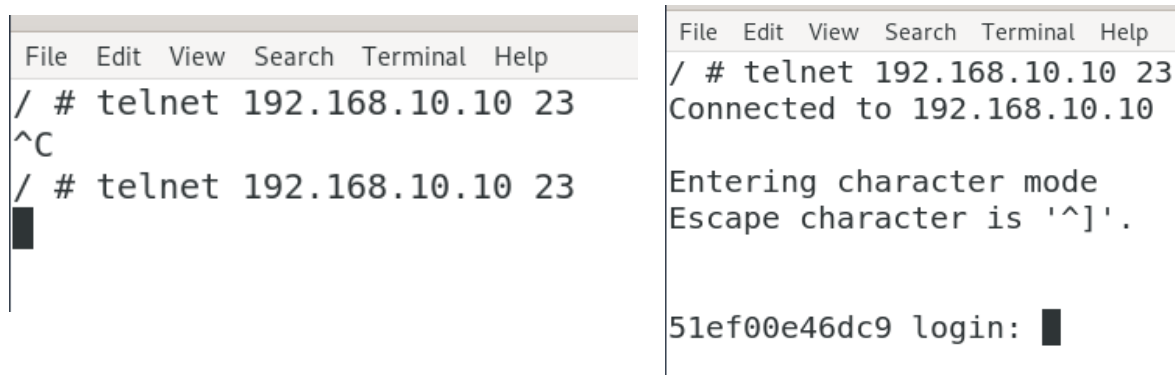
iptables -I INPUT 1 -m state --state ESTABLISHED,RELATED -j ACCEPT

iptables -P INPUT DROP

To allow only telnet connections on port 23, append this rule after the previous one:

iptables -A INPUT -p tcp --dport 23 -m state --state NEW -j ACCEPT

We can monitor that the rule works by using telnet from the client 192.168.10.20



In the left picture we can see that new connections are blocked and we cannot start a new one with telnet. In the right picture we have allowed it and can connect without any problems.

Exercise 6 - Limit module

The client1 (192.168.10.30) tries to DDOS the server with icmp packages! Try to mitigate the problem by limiting the icmp requests that can arrive at maximum 3 packets in a minute.

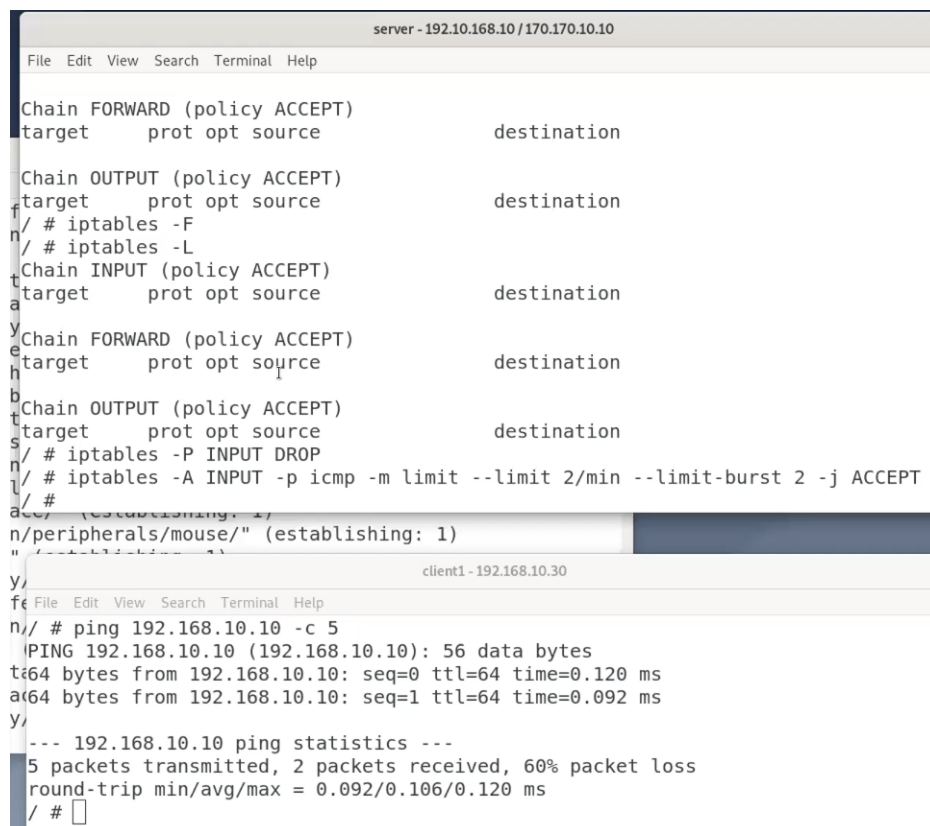
For this exercise we will use the **LIMIT MODULE**. We can call it with **-m limit**.

In order to understand how the limit module works, we'll use an analogy. Suppose, you're given a bucket containing 5 tokens. Whenever a packet comes in, you should throw out a token. Also, you can't allow a packet if the bucket is empty. In addition, you can add back tokens for example at the rate of 3 in an hour (or 1 in 20 minutes). Technically, the number of tokens is the "**limit-burst**" value, and the rate at which you can refill it is the "**limit**" value. We can use these two values as options for our rule.

As long as the traffic is within the given limits, packets will be accepted. However, as soon as the flow of packets exceeds this limit, the traffic passes through this rule over to the next one. Thus, you should set a default policy of DROP on the INPUT chain for this to work.

iptables -P INPUT DROP

iptables -A INPUT -p icmp -s 192.168.10.30 -m limit --limit 2/min --limit-burst 2 -j ACCEPT



```
server - 192.168.10 / 170.170.10.10
File Edit View Search Terminal Help

Chain FORWARD (policy ACCEPT)
target     prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination
n/ # iptables -F
/ # iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source                destination
Chain FORWARD (policy ACCEPT)
target     prot opt source                destination
Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination
n/ # iptables -P INPUT DROP
/ # iptables -A INPUT -p icmp -m limit --limit 2/min --limit-burst 2 -j ACCEPT
/ #
a/ (establishing: 1)
n/peripherals/mouse/" (establishing: 1)
" (establishing: 1)
client1 - 192.168.10.30
File Edit View Search Terminal Help
n/ # ping 192.168.10.10 -c 5
(PING 192.168.10.10 (192.168.10.10): 56 data bytes
t:64 bytes from 192.168.10.10: seq=0 ttl=64 time=0.120 ms
a:64 bytes from 192.168.10.10: seq=1 ttl=64 time=0.092 ms
y/
--- 192.168.10.10 ping statistics ---
5 packets transmitted, 2 packets received, 60% packet loss
round-trip min/avg/max = 0.092/0.106/0.120 ms
/ #
```

Exercise 7 - Time module

Drop all the packets that are coming in the upcoming minute

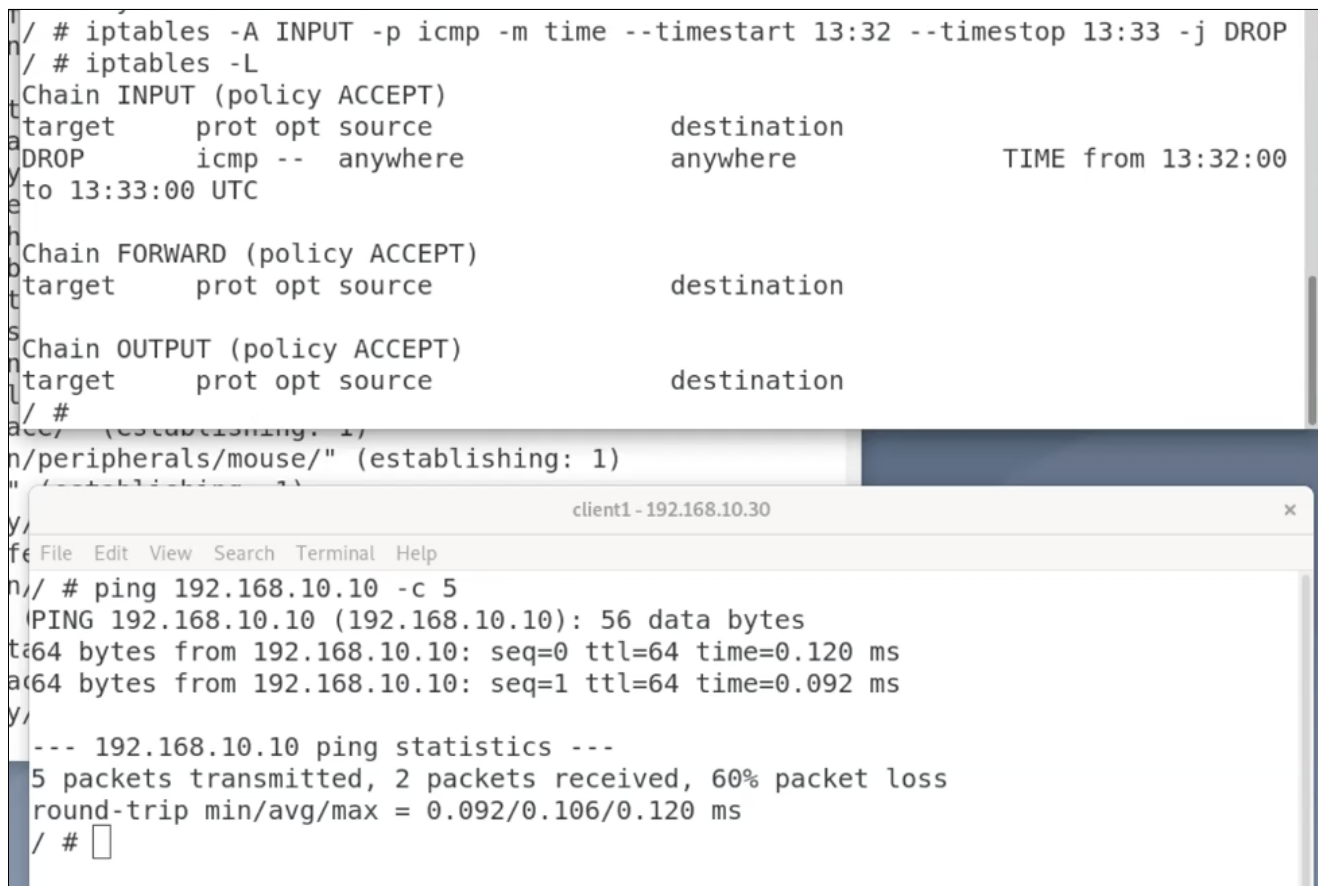
Iptables contains a time module as well, usable by typing **-m time**.

The function of this module is to block or accept packets within a given time. You can set the date, hours, month days even weekdays. It is quite useful for example if you want to block content from your workplace for a certain period of time, like a system update to prevent interruptions, but it can be also used, to allow websites at lunch time for the employees.

The input time is in UTC, so you have to set the parameter regarding that.

The following rule will DROP all the packets for a few minutes:

iptables -A INPUT -p icmp -m time --timestart 13:32 --timestop 13:33 -j DROP



```
/ # iptables -A INPUT -p icmp -m time --timestart 13:32 --timestop 13:33 -j DROP
/ # iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source                               destination      TIME from 13:32:00
to 13:33:00 UTC
DROP      icmp --  anywhere                               anywhere
Chain FORWARD (policy ACCEPT)
target     prot opt source                               destination
Chain OUTPUT (policy ACCEPT)
target     prot opt source                               destination
/ #
n/peripherals/mouse/" (establishing: 1)
client1 - 192.168.10.30
File Edit View Search Terminal Help
n/ # ping 192.168.10.10 -c 5
PING 192.168.10.10 (192.168.10.10): 56 data bytes
64 bytes from 192.168.10.10: seq=0 ttl=64 time=0.120 ms
64 bytes from 192.168.10.10: seq=1 ttl=64 time=0.092 ms
--- 192.168.10.10 ping statistics ---
5 packets transmitted, 2 packets received, 60% packet loss
round-trip min/avg/max = 0.092/0.106/0.120 ms
/ #
```

Exercise 8 - Last exercise

Write a rule that forwards all traffic from port 80 to port 8080 on the server.

The solution is the following:

```
iptables -t nat -A PREROUTING -p tcp -dport 80 -j REDIRECT --to-ports 8080
```

As you can see we used the table **nat**, the chain **prerouting** and the target **redirect** with a specific option to be able to do it.

This is the end of this short presentation, but it has all the basics you need to get started with iptables. Some commands are more advanced than others and there are many options and parameters that we haven't covered here, because the full list is huge. By reading this report we hope that you have learned something new and figured out how to set up a simple firewall. There are an infinite amount of combinations with rules and parameters and you will be able to do everything with a little practice.

Please, consider that iptables is not immune to bugs, check it out by yourself here:

<https://bugzilla.netfilter.org/>

Thank you!