Firewall

Laboratory for the class "Information Systems Security" (02TYMUV)
Politecnico di Torino – AA 2024/25
Prof. Antonio Lioy

prepared by:
Flavio Ciravegna (flavio.ciravegna@polito.it)
Andrea Atzeni (andrea.atzeni@polito.it)

v. 3.7 (29/11/2024)

Contents

1 Purpose of the laboratory

The purpose of this laboratory is to experiment with the configuration and use of basic elements of firewalls, that provide filtering functionality both at the network level and/or at the application level.

The exercises use the following software:

- Netfilter / IP Tables – is used to set up, maintain, and inspect the tables of IP packet filter rules;
- Dynamic port forwarding SSH – allows to configure the SSH server as a circuit-level gateway (use `man sshd` for further details).
- ModProxy – the `mod_proxy` module allows to configure an Apache web server to act as an application gateway for the web applications.
- HTTPtunnel – creates an HTTP tunnel to try circumventing the firewall policies;
- PTunnel – creates a tunnel by exploiting the ICMP echo-request and echo-reply packets to try circumventing the firewall policies.

IPtables

IPtables, illustrated in Fig. **??**, uses the concept of "chain": each chain is a list of rules (with an associated default policy) which can match a set of packets. Each rule specifies what to do with a packet that matches the rule. This is called a 'target' (action). Thus, a firewall rule specifies matching criteria for a packet, and a target. If the packet does not match, the next rule in the chain is the considered.
To filter the IP packets, the following three types of chains can be configured:

- `INPUT`, contains the rules that must be applied to the IP packets addressed to the machine on which IPtables is installed.
- `OUTPUT`, contains the rules that must be applied to the IP packets that originate from the machine on which IPtables is installed.
- `FORWARD`, contains the rules that must be applied to the IP packets that are neither addressed to

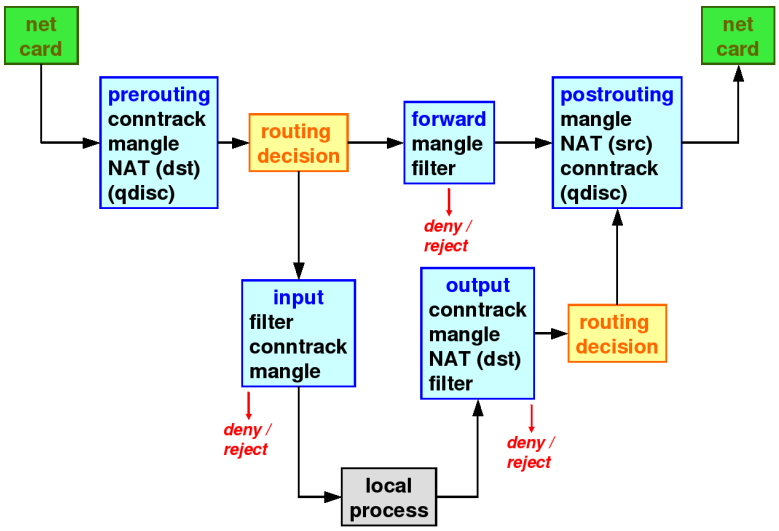

Figure 1: Architecture of Netfilter/IPtables.

nor originated from the machine on which IPtables is installed but to those packets that must be routed (forwarded) to a specific interface (if IP forwarding is enabled).

Other chains could be created based on the operating scenario.
IPtables can be configured with the command `iptables`; for further details, read the manual pages for this command by using `man iptables`. The main commands are illustrated here below:

- the option `-P chain target` allows to set the default policy (e.g. ACCEPT/DROP) for a chain (INPUT, OUTPUT, FORWARD) to the `target` value (for further details on the `target` value, see the explanations below);
- the options `-A chain` and `-I chain` allow respectively to add a new rule at the end and at the beginning of the chain `chain`;
- the options `-D chain` and -R chain allow to delete and to replace respectively a rule (called also policy) in a chain, where the rule can be referred either as an index in the list of rules or via an IP address;
- the option `-F [chain]` flushes all the rules present in a specific `chain` (or all the chains in the table if a specific `chain` is not given). Pay attention: the default policy is not modified however.
- the option `-N chain` creates a new user-defined chain with the given name. There must be no target with that name already.
- the option `-X chain` deletes the user-defined chain `chain`.
- `-L, --list [chain]` list all rules in the selected chain. If no chain is selected, all chains are listed.
- `-h` (Help) gives a (currently very brief) description of the command syntax.

For example, if you need to reset all the chains you can use the command:

```
iptables -F
```

or if you want to reset only the output chain, you can use the command:

```
iptables -F OUTPUT
```

When you specify a filtering rule, you typically specify information about various parts of the IP packet, such as the source or destination address, or the source and destination port. For this purpose, IPtables provides a series of commands (see the entire list with the command `man iptables`):

- `-s IP_source`, IP source address;
- `-d IP_dest`, IP destination address;
- `--sport port`, source port;
- `--dport port`, destination port;
- `-i interface`, name of a network interface over which a packet was received (INPUT chain);
- `-o interface`, name of a network interface over which a packet is going to be sent (OUTPUT chain);
- `-p proto`, the protocol of the packet to check. The specified protocol can be one of `tcp`, `udp`, `icmp`, or `all`, or it can be a numeric value, representing one of these protocols or a different one. A protocol name from /etc/protocols is also allowed.
- `-j action`, action to perform (the *target* in IPtables notation)
- `-y` or `--syn`, in case the option `-p tcp` is used, it identifies (only) the packets that have the flag SYN enabled;
- `--icmp-type type`, equivalent to `-p icmp`, `type` specifies the ICMP packet type
- `-l`, enables the log of the result of a rule evaluation via syslog in `/var/log/messages`

The action to be performed, that is the `target` value in an `iptables` command, can take one the following values:

- `ACCEPT` means to accept the packet, or to let the packet pass through.
- `DROP` means to drop the packet and do not send any notification error.
- `REJECT` means also to drop the packet but it sends back also an error packet in response to the matched packet: otherwise it is equivalent to DROP so it is a terminating TARGET, ending rule traversal; by default, it is sent a packet of type "ICMP port unreachable", nevertheless this can be changed by setting the option `--reject-with`
- `QUEUE` means to pass the packet to userspace (if supported by the kernel).
- `RETURN` means stop traversing this chain and resume at the next rule in the previous (calling) chain. If the end of a built-in chain is reached or a rule in a built-in chain with target RETURN is matched, the target specified by the chain policy determines the fate of the packet.

Other valid targets are all the chains defined by the user.

Useful commands

The exercises will require to exchange messages between two computers, by using the `scp` utility. Consequently, you will have to generate the ssh host key with the `ssh-keygen`:

```
ssh-keygen -A
```

and start the `ssh` server:

```
systemctl start ssh
```

To copy a file (e.g. `prova`) from the `ssh` client machine into the `root` directory on the machine running `ssh` server, you can use the command:

```
scp prova kali@IP_address_ssh_server:/home/kali
```

Insert the password 'kali' when asked.
The exercise will require also to use the Apache web server. To start the Apache server use the command:

```
systemctl start apache2
```

To view the IP address on a host, you can run the command:

```
ifconfig
```

2 Packet filter

2.1 Personal firewall

This section familiarizes you with the use of the command `iptables`, and explains afterward how to configure a local packet filter, in order to protect a single machine.

Run:

```
sudo su
```

Verify the configuration of IPtables on your machine with the command:

```
iptables -L -v -n
```

Which authorisation policy is configured by default on your machine?

→

Which chain do you have to modify to protect your machine from external connections?

→

Form two groups of hosts, named Alice and Bob, where Alice is the machine to be protected with the packet filter, and Bob will be in charge with verifying/testing the configuration of the packet filter of Alice. On Alice's machine you need to start the services and configure the firewall, while Bob will test the firewall configuration on Alice's host.

Execute now the following steps:

- Alice starts the Apache web server and the SSH daemon (server);
- Bob verifies whether he can reach Alice via ping, and whether any service is active on Alice. He can do these actions by executing the following commands:

  ```
  ping IP_Alice
  ```

  ```
  nmap -sT -Pn -n -p 80,22 -v IP_Alice
  ```

Check with `wireshark` the traffic exchanged when Bob connects with his browser to Alice's host. You should observe the 3-way handshake messages and the significant ports.

Write down the `iptables` command to modify the authorisation policy of Alice to reject any input traffic (hint: you need to modify the default policy for the INPUT chain from ACCEPT to DROP):

→

Bob verifies now that he CANNOT ping any more Alice, and that he cannot connect anymore to Alice via SSH and HTTP (with the browser).

Check with nmap (running on Bob) the status of the ports 22 and 80 on Alice's host. What is their status?

→

Write down the `iptables` command to add a rule to the authorisation policy on Alice (for the input traffic) to enable ICMP traffic (for simplicity, we provide you some of the parameters of the command):

```
iptables -A ...  -p icmp -j ACCEPT
```

→

In the same way, on Alice, write down the `iptables` command to allow the TCP input traffic towards the port 80:

```
iptables -A ...  -p tcp --dport ...  -j ...
```

→

Perform the modifications and check the new configuration of IPtables. Verify that Bob can ping (again) Alice and that he can connect to Alice via HTTP (with the browser).

Check again with nmap (running on Bob's host) the status of the ports 22 and 80 on Alice's host. What is their status?

→

How can `nmap` distinguish between filtered ports and closed ports? Verify by analysing the traffic exchanged between the two machines (e.g. with `wireshark`).

→

Finally, start up Apache server on Bob's host and try to connect with a web browser from Alice's host to the web server running on Bob.

Why Alice is not able to browse web content on Bob's host? Verify by analysing the traffic among the two machines (with `wireshark`).

→

Before passing to the execution of the next exercise, restore on Alice's host the authorisation policy of type "ACCEPT ALL". Hint: You need to delete all the current rules and specify afterwards the rules for the input chain.

→

2.2 Packet filter stateless

Form a groups of 3 hosts, let's say Alice, Bob and Frank, where Frank acts as a firewall between Alice, acting as server, and Bob, acting as client in the scenario described below.

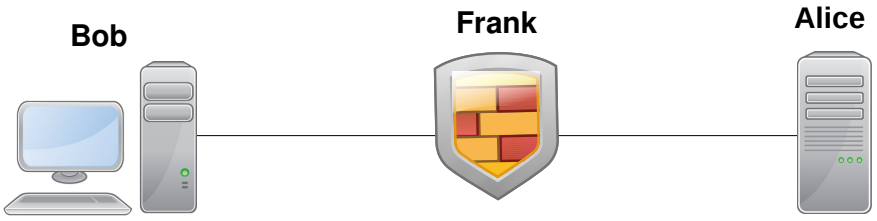

Figure 2: Reference configuration

To "simulate" in the laboratory the configuration illustrated in Figure **??**, proceed as follows:

1. Alice configures a routing rule so that the packets addressed to Bob will pass through Frank's host. For this purpose, you can either use the `ip` command:

   ```
   ip route add IP_Bob via IP_Frank dev interfaccia
   ```

   or the `net-tools` command:

   ```
   route add -host IP_Bob gw IP_Frank
   ```

2. similarly, Bob configures a routing rule to route the packets addressed to Alice to pass through Frank's host.

   For this purpose, you can either use the `ip` command:

   ```
   ip route add IP_Alice via IP_Frank dev interface
   ```

   or:

   ```
   route add -host IP_Alice gw IP_Frank
   ```

3. Frank disables the sending of ICMP redirect packets, and enables the packet forwarding:

   ```
   echo 0 > /proc/sys/net/ipv4/conf/*/send_redirects
   ```

   ```
   echo 1 > /proc/sys/net/ipv4/ip_forward
   ```

4. Alice and Bob start up respectively the Apache server and the SSH daemon.

Before proceeding, check (for example by using `ping` command) that all the hosts (Alice, Frank, Bob) can communicate among themselves. Verify also that the traffic exchanged between Alice and Bob passes effectively through Frank (use for example `wireshark`). For example, on Frank you should see 4 ICMP request messages, every time Alice pings Bob, as shown in Fig. **??**. In this example, Alice has the IP address 10.0.2.5 and Bob has the address 10.0.2.6. Check the Source and Destination addresses in each of the four ICMP messages.

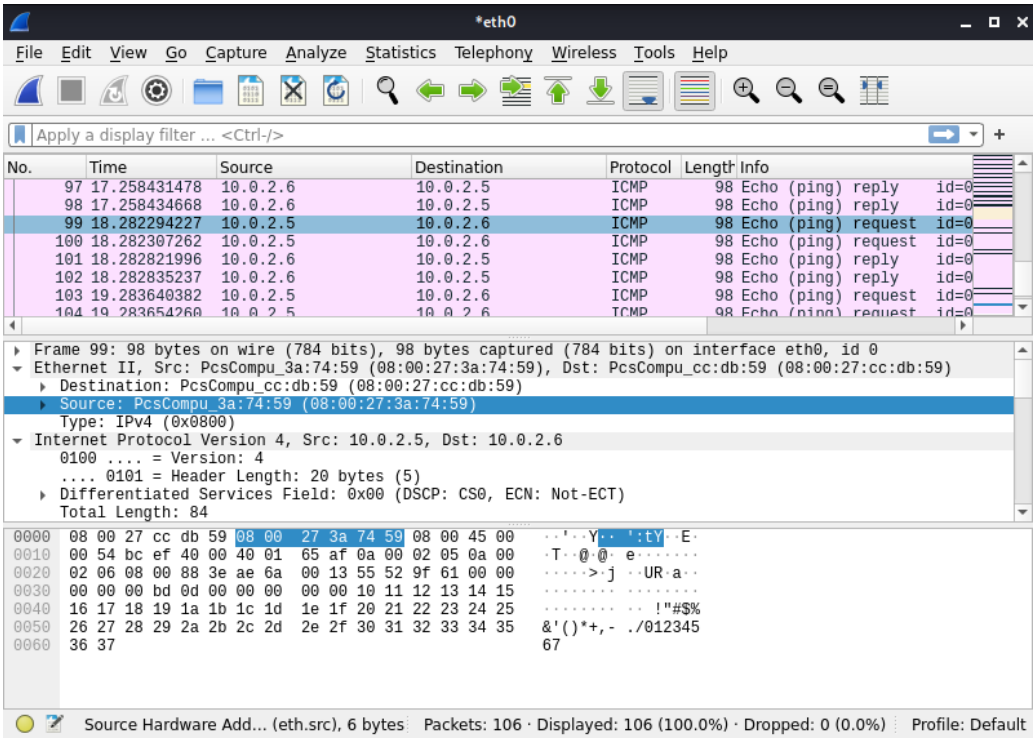

Figure 3: Intercepting ICMP traffic exchanged between Alice and Bob (with wireshark) on Frank's host.

NOTE

In some cases, the disabling of ICMP "send redirect" packets on Frank doesn't work. Thus, Alice receives anyway the ICMP Host Redirect packet. This may be observed when checking the traffic generated with the `ping` command, because the first output generated is something like (where 192.168.1.8 is the IP address of Frank)

```
Redirect Host(New nexthop: 192.168.1.8)
```

If you encounter this situation, Alice and Bob have to execute the following commands:

- cancel the local route cache with the command:

  ```
  ip route flush cache
  ```

- add a rule for the INPUT chain of iptables, which rejects the receipt of ICMP redirect packets, with the following command:

  ```
  iptables -A INPUT -p icmp --icmp-type redirect -j DROP
  ```

2.2.1 Output traffic

At this step, Frank will apply an authorisation policy to allow Alice to navigate on any external web server: Alice will act as client on the protected network and Bob will act as external web server.

Start the web server on Bob:

```
systemctl start apache2
```

Configure the following authorisation policy on the packet filter of Frank:

```
iptables -P FORWARD DROP
iptables -A FORWARD -p tcp -s IP_Alice --dport 80 -j ACCEPT
iptables -A FORWARD -p tcp -d IP_Alice --sport 80 -j ACCEPT
```

and respond to the following questions:

On which chain operate the above commands?

→

Can Alice connect to the web server running on Bob's host?

→

Which is the purpose of the last rule?

→

Can Alice connect to the web server running on another machine, when his traffic passes through Frank's host?

→

Start the Apache web server on Alice (if not already started). Can Bob connect with his browser to the web server running on Alice's host?

→

Stop the Apache web server on Bob. Next, on Bob's host run the command:

```
curl http://IP_Alice --local-port 80
```

Can Bob download the web page from Alice's host?

→

Can Alice and Bob connect to the corresponding SSH servers of the counterparts, that is can Alice connect to SSH server of Bob and viceversa? (hint: think what happens if Bob uses the port 80 as source port)

→

NOTE

You can use `nmap` to verify the configuration. For example, execute the following command on Bob's host:

```
nmap -sS -Pn -n -p 22 IP_Alice --source-port 80
```

Or you can use the following command on Bob to establish a connection to SSH server on Alice, by

using source port 80:

```
ssh -o 'ProxyCommand nc -p 80 %h %p' IP_Alice -l kali
```

Write down (and execute) the `iptables` command which modifies the policy on Frank's host, so that to enable only the web connections from Alice towards any external user (any IP address):

NOTE

The rules are processed by the kernel in the order in which they can be viewed with the command:

```
iptables -L -v -n
```

```
iptables -I FORWARD 2 -p ...  -d IP_Alice --sport 80 --syn -j DROP
```

```
→
```

Now try again to connect from Bob to Alice's host by running (on Bob's host):

```
nmap -sS -Pn -n -p 22 IP_Alice --source-port 80
```

```
curl http://IP_Alice --local-port 80
```

Have you been successful in running the commands above?

```
→
```

2.2.2 Public service

In this exercise, Frank will apply an authorisation policy so that to allow remote access to Alice's host via SSH: Alice will act as SSH server on the protected network, and Bob will act as a remote client which is authorised to get access to Alice's host via SSH.

Write down the (complete) `iptables` commands to modify the authorisation policy on Frank's host, so that to accept SSH connections towards Alice'host coming from any IP address:

```
iptables -A FORWARD -p tcp -d IP_Alice --dport 22 -j ACCEPT
```

```
iptables -A FORWARD -p ...  -s IP_Alice --sport ...  --syn -j DROP
```

```
iptables -A FORWARD -p tcp -s IP_Alice --sport 22 -j ...
```

```
→
```

Perform the modifications and verify that Bob can connect to Alice's host via SSH, but not vice versa.

On Bob's host you can use the command:

```
nmap -sS -Pn -n 22 IP_Alice
```

or

```
ssh IP_Alice -l kali
```

NOTE

To verify the above using the `ssh` command, you might need to change the `/etc/ssh/sshd_config` file. In particular, the login as kali has to be allowed (`PermitRootLogin yes`) as well as the password authentication (`PasswordAuthentication yes`). After these changes have been applied, the `ssh` daemon shall be restarted.

2.2.3 ICMP traffic

Try to ping Alice's host from Bob.

```
ping IP_Alice
```

Are you successful?

→

Modify the authorisation policy on Frank's host, so that to enable selectively the ICMP traffic.

In particular, write down the complete `iptables` commands to enable the ICMP traffic from and to Alice's host:

```
iptables -A FORWARD -p icmp -s IP_Alice --icmp-type echo-request -j ACCEPT
```

```
iptables -A FORWARD -p icmp -d ...  --icmp-type echo-reply -j ...
```

```
iptables -A FORWARD -p icmp -d IP_Alice --icmp-type echo-request -j ...
```

```
iptables -A FORWARD -p icmp -s ...  --icmp-type echo-reply -j ...
```

→

Now try again to ping Alice's host from Bob.

```
ping IP_Alice
```

Are you successful?

→

Why is it dangerous to enable ICMP traffic without any restriction? Hint: execute `iptables -p icmp -h` for a list of types of ICMP messages.

→

Suppose that Frank must filter the traffic toward the subnet `1.2.3.0/24` instead of filtering just the traffic directed to Alice's host: what happens when Frank receives an ICMP echo request packet with destination address `1.2.3.255` if Frank's host is configured to enable the ICMP traffic (as mentioned above)?

→

2.2.4 Bandwidth limitation

Refer to the exercise in Section **??**.

What is the effect of replacing the rule:

```
iptables -A FORWARD -p icmp -d IP_Alice --icmp-type echo-request -j ACCEPT
```

with the following rule?

```
iptables -A FORWARD -p icmp -d IP_Alice --icmp-type echo-request -m limit --limit
20/minute --limit-burst 1 -j ACCEPT
```

Hint: check the use of the "`--limit`" directive at the link https://www.netfilter.org/documentation/HOWTO/packet-filtering-HOWTO-7.html#ss7.3.

→

Now, to verify your hypothesis, proceed in the following way. Delete all the rules on Frank regarding the icmp traffic.

You can delete them with the command (where the numbering of the rules starts from 1):

```
iptables -D FORWARD rule_number
```

Then, add the following rules on Frank:

```
iptables -A FORWARD -p icmp -s IP_Alice --icmp-type echo-request -m limit --limit
10/minute --limit-burst 10 -j ACCEPT
```

```
iptables -A FORWARD -p icmp -d IP_Alice --icmp-type echo-reply -j ACCEPT
```

Then, on Alice run the command:

```
ping -i 0.000000001 IP_Bob
```

What do you note? Analyze the captured ICMP packets on Frank and Bob with `wireshark`.

→

Against what kind of attack is protected Bob?

→

2.2.5 Protection from IP spoofing

In the current configuration, can Frank detect/avoid that Alice and Bob perform IP spoofing of their IP addresses (to respond to this question, think about how many network interfaces are available)?

→

What about the configuration illustrated in Figure **??**?

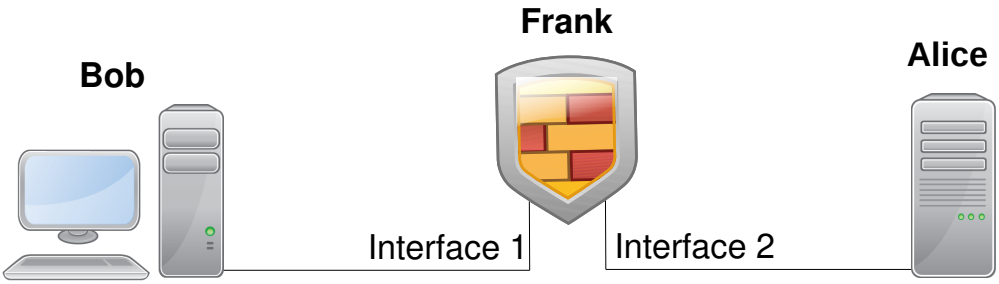

Figure 4: Two interfaces scenario.

→

2.3 Stateful packet filter

In this exercise, you will use the same groups (Alice, Frank and Bob) you have used in the Section **??**. The objective is the same as in Section **??** (i.e. allow only the web connections from Alice towards any external web server), but in this case you will configure a stateful packet filter.

On Frank, run the commands:

```
iptables -F
iptables -A FORWARD -p tcp -s IP_Alice --dport 80 -j ACCEPT
iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Start the Apache web server on Bob. Next, connect from Alice to Bob's web server (with a web browser). Are you successful ?

→

Now stop the Apache web server on Bob (if you don't do it, you will have problems executing the next command).

Then, on Bob's host run the command:

```
ssh -o 'ProxyCommand nc -p 80 %h %p' IP_Alice -l kali
```

Are you successful?

→

What is the meaning of the new rules? Hint: analyse the introduction of the section "MATCH EXTENSIONS" and the directive "–state" at the link:
https://www.netfilter.org/documentation/HOWTO/packet-filtering-HOWTO-7.html#ss7.3.

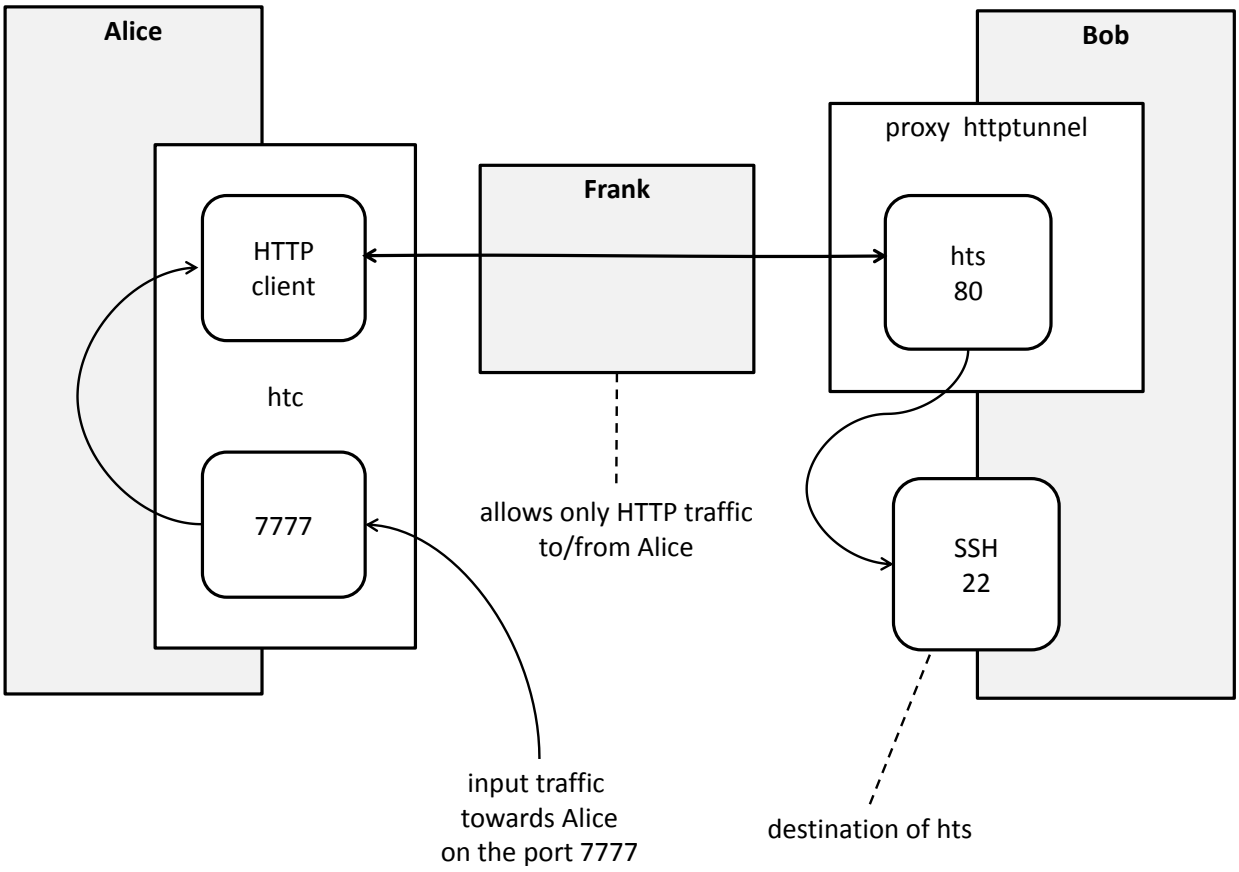

Figure 5: The scenario to implement with HTTPtunnel

→

3 Additional exercises (optional)

Creation of an HTTP tunnel

In this exercise, we provide a critical view of the concepts/tools used in the previous exercises: we will see how the authorisation policies of a firewall could be tricked when an internal user (Alice) and an external one (Bob) collude (collaborate) together.
In particular, try to respond to the following questions:

- suppose Frank enabled the web traffic from Alice towards external machines via packet filtering functionalities as in the exercises **??** and **??**: how can Alice and Bob violate the authorisation rule of Frank to open an SSH connection instead of a web (HTTP) connection?

  →

  What about using a circuit-level gateway or an application gateway as in the exercises **??** and **??**. Can still Alice and Bob violate/trick the authorisation rules of Frank as above?

3.1 Basic HTTP tunnel creation

In this exercise, Alice will act as a client, trying to create an HTTP tunnel to a server, embodied by Bob, using the `httptunnel` tool so that to trick the authorisation policy of Frank. The environment is set as in the preceding exercises:

- Alice is an internal user of the network
- Frank represents a border firewall and uses a packet filter to implement an authorisation policy allowing the internal users to access the web (and receive the responses). As seen in the previous exercise, Frank will use the following rules (remember to flush the preceding IPtables configurations and make sure that the default policy is DROP):

```
iptables -A FORWARD -p tcp -s IP_Ali --dport 80 -j ACCEPT
```

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

- Bob is the external user colluding with Alice. Bob starts the SSH server with the command:

```
systemctl start ssh
```

In this exercise we will use the tool HTTP tunnel to create an HTTP tunnel allowing Alice to reach the server running on Bob's host even though a firewall is present.
HTTP tunnel is composed of two components, a server (`hts`) and a client (`htc`). The HTTP tunnel server (`hts`) listens for incoming HTTP tunnel connections (by default on the port 8888) and redirects all the traffic received towards another destination, specified with the options `-F` or `-d`.
The syntax of the command `hts` and the main options are presented below:

```
hts [-F host:port] [port]
```

where:

- `-F` *host:port* specifies the IP address and the port on which the received traffic will be redirected;
- *port* specifies the port on which the `hts` server listens for the incoming connections

HTTP tunnel Client (`htc`) opens a connection with a HTTP tunnel server, listens for incoming connections on a port and redirects all the traffic received to the server. The syntax of the command `htc` and its main options are presented below:

```
htc [-F port] [server:server_port]
```

where:

- `-F port` is the port on which `htc` waits for incoming connections;
- `server:server_port` specifies the address and the port of the `hts` server with which it opens a connection and where the traffic will be redirected (that is the actual tunnel).

We'll use the commands `hts` and `htc` to establish the two endpoints of the tunnel, respectively the server and the client:
Write down the (complete) `hts` command Bob must run to start an HTTP daemon listening on the port 80, which redirects the traffic received towards the SSH server:

```
hts -F localhost:22 ...
```

```
→
```

Write down the `htc` command Alice must run to launch a process that intercepts the local traffic addressed to the local port 7777 and redirects it towards the port 80 of Bob:

```
htc -F ....  IP_Bob:80
```

→

The scenario described above is presented in the Figure **??**.
Write down the (complete) command Alice must use to connect to the SSH server of Bob through the tunnel:

```
ssh -p ...  root@localhost
```

→

Verify the content of the HTTP packets exchanged between Alice and Bob. Even though theoretically all the traffic can be redirected in this way, can you identify a simple way to recognize the traffic generated with `httptunnel`?

→

3.2 Creation of an ICMP tunnel (ping)

Another possibility to trick the firewall policies is to use the tool Ptunnel, which allows to create a reliable TCP tunnel encapsulated inside ICMP packets of type echo-request and echo-reply. Considering that firewalls typically let the ICMP traffic of type echo pass through (even though less frequently than they let pass through the HTTP traffic), Ptunnel can be used for our scope.
The scenario in this case is the same as in the previous exercises:

- Alice is an internal user of the network
- Frank represents the border firewall. Frank uses a packet filter implementing the following authorisation policy: allow the internal users to send and receive ICMP traffic (e.g. execute ping command). As observed in the previous exercises, Frank will use the following rules (before executing them, remember to flush the previous configurations done on Frank's host and check that the default policy is DROP):

  ```
  iptables -A FORWARD -p icmp -s IP_Ali --icmp-type echo-request -j
  ACCEPT
  ```

  ```
  iptables -A FORWARD -p icmp -d IP_Ali --icmp-type echo-reply -j ACCEPT
  ```

  ```
  iptables -A FORWARD -p icmp -d IP_Ali --icmp-type echo-request -j
  ACCEPT
  ```

  ```
  iptables -A FORWARD -p icmp -s IP_Ali --icmp-type echo-reply -j ACCEPT
  ```

- Bob is the external user colluding with Alice. Bob starts the SSH server:

```
systemctl start ssh
```

In the configuration of Ptunnel, we need to consider three entities: a destination server, a proxy and a client:

- the destination server is the server (typically external) that need to be reached;
- the client is a process that listens for incoming connections on a specific TCP port, and encapsulates the payload and some of the header fields of the TCP packets received into ICMP echo request packets, including the information on the destination server. In addition, it receives as response from the proxy the ICMP echo replay packets, and recomposes the TCP packet sent by the destination server;
- the proxy is a process that receives the ICMP echo request packets generated by the client. The proxy extracts the packet and the information regarding the destination server, it recreates the TCP packet and forwards it to the destination server. Additionally, the proxy receives the responses from the destination server and encapsulates them into ICMP echo reply packets, which are sent to the client.

Further details on the PTunnel tool are available at: https://www.mit.edu/afs.new/sipb/user/golem/tmp/ptunnel-0.61.orig/web/
The syntax of the `ptunnel` command is the following one:

```
ptunnel [-p proxy_address] [-lp listening_port] [-da dest_addr] [-dp dest_port]
```

where:

- `-p` *proxy_address*, serves to specify the IP address of the proxy (client);
- `-lp` *listening_port*, serves to specify the listening port where the packets to be sent to the server will be received (client);
- `-da` *dest_addr*, serves to specify the IP address of the destination server (client);
- `-dp` *dest_port*, serves to specify the port of the destination server to be reached (client).

We'll use the command `ptunnel` to create both the client and the proxy. In our simplified case, Bob will start both the proxy and the destination server. This scenario is presented in Figure **??**.
Bob starts the proxy with the following command:

```
ptunnel &
```

Write down the (complete) command to be executed by Alice in order to start a client `ptunnel` which listens on the port 8000 for the packets to be sent through the proxy (running on Bob) to the SSH server (running also on Bob):

```
ptunnel -lp 8000 -p ...  -da ...  -dp 22
```

```
→
```

Write down the ssh command to be executed by Alice in order to open a connection towards the SSH server of Bob.

```
→
```

Suppose you use an additional machine (Carlo) on which you start the Apache server. Which is the ptunnel command to be executed by Alice in order to reach the Apache server on Carlo, considering that Bob is used as proxy as in the previous case and that you use the port 8000 on the client (as above) as listening port for the packets to be sent to the Apache server?

```
ptunnel -lp 8000 -p ...  -da ...  -dp 80 &
```

→

In case you want to reach from the client more destination servers simultaneously (for example an SSH server and an Apache server on two different hosts), what should you allocate on the client? Write down and try out the ptunnel commands for this case.

→

Try to sniff the traffic between Bob and Alice. Do you note any difference between the ICMP packets generated with `ptunnel` and the "normal" ICMP packets?

→

TUTORIAL: Circuit-level gateway

In this exercise, you'll use the same groups formed for the exercise **??**. Frank removes the rules regarding the web traffic: the web traffic will be subsequently re-established by using a circuit level gateway.
We are going to use the functionality of "dynamic port forwarding" of the SSH client and SSH server in order to experiment with the functionality of a circuit-level gateway.
Before starting, Frank adds the user `ali` and starts the SSH server.

3.3 Configuration of SSH as circuit level gateway

To use the circuit level gateway, Alice creates the RSA credentials to be used for authentication at the gateway with the following command:

```
ssh-keygen
```

Save the key generated in the file `.ssh/id_rsa` (default) and use the default passphrase (null). You can note that `ssh-keygen` has indeed created two files: `id_rsa` contains the private key and `id_rsa.pub` contains the corresponding RSA public key, which will be sent to Frank.
Alice must thus send his public key to Frank so that he can authenticate himself with an asymmetric challenge and the key that has just been generated. To transfer the public key of Alice on the host of Frank you can use the following command:

```
ssh-copy-id -i .ssh/id_rsa.pub ali@IP_Fra
```

This command opens an SSH connection with Frank

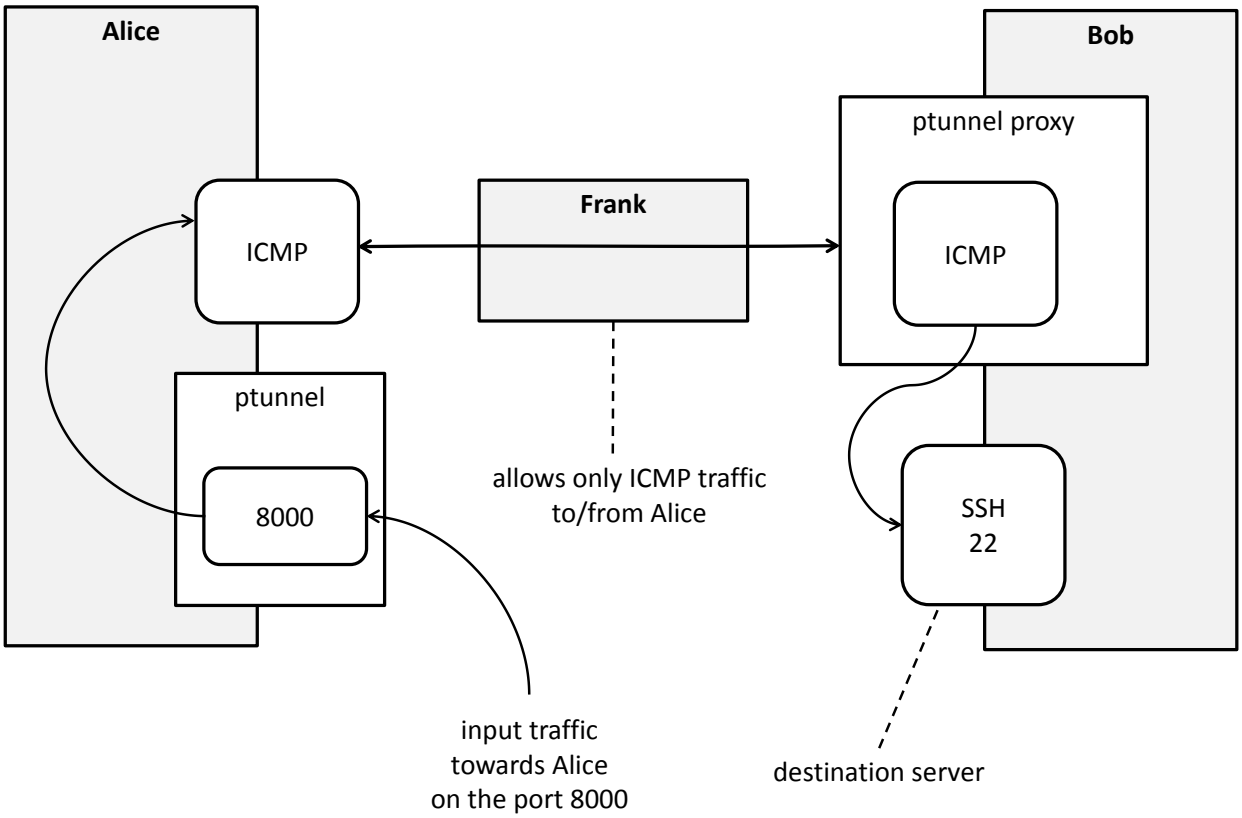

Figure 6: The scenario to implement with Ptunnel.

- creates the directory `/home/ali/.ssh`, if it does not exist already,
- adds the key in `id_rsa.pub` to the keys associated to the user `ali` (in `/home/ali/.ssh/authorized_keys`),
- registers the new key with the command `ssh-add`

At this point we will configure the circuit level gateway on Frank by executing the following steps:

- makes the following changes in the file `sshd_config`:
  1. changes the listening port by changing

     ```
     Port 22
     ```

     →

     ```
     Port 1080
     ```

  2. disables the authentication with username and password by changing

     ```
     #PasswordAuthentication yes
     ```

     →

     ```
     PasswordAuthentication no
     ```

  3. disables the port forwarding, by adding (at the end) the following command

     ```
     AllowTcpForwarding no
     ```

- restarts the SSH server with the new configuration (`service ssh restart`);

Verify that Alice can connect to the port tcp/1080 on the host of Frank with the following command:

```
nmap -sT -Pn -n -p 1080 IP_Fra
```

Now Alice tries to establish an SSH connection with the gateway:

```
ssh -p 1080 ali@IP_Fra
```

Alice closes the previous connection and tries next to connect to the server web of Bob through the circuit level gateway:

- Alice opens an SSH connection with the gateway by activating the "dynamic port forwarding":

  ```
  ssh -D 1080 -p 1080 -i id_rsa ali@IP_Fra
  ```

  The paramater `-D` is used to enable the "dynamic port forwarding" because it allocates a listening socket on localhost on the port specified as parameter (in our case, on the port *1080*). If a connection is opened toward this port, the connection will be made over an `ssh` secure channel (for further details, you are advised to check the option `-D` in the `ssh` manual by running the command `man ssh`)

- Alice configures the browser to contact at the IP address `127.0.0.1` the SOCKS interface put at disposal by the `ssh` client for "dynamic port forwarding". In Firefox, you have to select in `Preferences` → `Network Proxy` → `Manual Proxy Configuration`;

- tries to connect to the web server of Bob.

Alice cannot connect to the web server of Bob because she is not authorised yet to open web connections. You can check this by analysing the network traffic with wireshark. You should observe:

- Ali→Fra: SSH Encrypted request packet

- Fra→Ali: SSH Encrypted response packet (error)?

Thus, you must further refine the authorisation policy of the circuit level gateway so that to allow Alice to open only web connections towards Bob:

- Frank modifies the file `/etc/ssh/sshd_config` by adding at the end of the file the following lines:

  ```
  Match User ali
  AllowTcpForwarding yes
  PermitOpen IP_Bob:80
  ```

- Frank restarts the SSH server

- Alice restarts the SSH client

Alice tries again to connect to the web server and to SSH server of Bob and now she can navigate! Analyse the network traffic on the hosts of Alice, Frank and Bob:

- Ali→Fra: SSH Encrypted request packet

- Fra↔Bob: exchange (for example with HTTP protocol)

- Fra→Ali: SSH Encrypted response packet

Basically, in this exercises you obtain several advantages with respect to solution in which you enabled the web traffic via packet filtering (in the exercises **??** and **??**). In particular, you can perform user authentication and you can set up a protected channel. Additionally, in this configuration, by introducing the intermediate actor Frank we can protect better the TCP/IP stack of the machine of Alice from attacks.