Iptables

HW8-CNS Sapienza

Manuel Ivagnes 1698903

02/01/20

1 Introduction

The aim of this homework is to better understand how to configure and use the iptables for stateful session filtering. As for the previous homework, I will use the docker containers to simulate a virtual machines network.

First, I will make a short summary of the slides and internet material that I will also reuse to study. After that, I will make some simple tests using the basic firewall configuration of the standard ubuntu docker image, then I will use the iptables to apply some rules and check the changed behavior.

2 Firewall

A firewall is a network security system that monitors and controls incoming and outgoing network traffic based on predetermined security rules. It typically establishes a barrier between a trusted internal network and untrusted external network, such as the Internet [2].

However, it does not protect from attacks originated within the network to be protected (or which could pass through it). Moreover, it is not able to avoid/block all possible viruses and worms.

There are 3 main categories:

- packet filters (stateless)
- stateful filters (session filtering)
- application-level gateway

2.1 Packet filters (stateless)

For each packet, the firewall uses a set of filtering rules based on pattern-matching, to decide whether to reject, drop or allow the packet to proceed. To decide, it uses the informations available in the packet, such as:

- IP, both source and destination
- Protocol identifier (TCP, UDP, ICMP ...)
- TCP flags (SYN, ACK, ...)
- ICMP message type

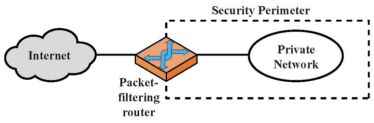
Given that, this firewall is **stateless**, each decision must be made on per-packet basis, and it cannot examine packet's context (as the TCP connection to which it belongs).

For instance, we can think to block all the incoming requests, but while we have a TCP connection already opened with a server on a specific port, this can also reply on a different port. The stateless firewall, which does not understand the connection with the already opened TCP connection, will block the packet and corrupt the communication with this server.

Packet filters weaknesses:

- Do not prevent application specific attacks (like buffer overflow in URL decoding routine)
- There is not user authentication mechanism (except the addressed-based auth, which is not reliable, due to spoofing)
- Vulnerable to TCP/IP attacks such as spoofing
- Security breaches due to misconfigurations

A common attack is the *Fragmentation attack*, which uses 2 or more packets such that each of them can pass the firewall; but when the packets are assembled together they form a packet that should have been dropped.



2.2 Stateful filters (session filtering)

Decisions are still made separately for each packet, but in the context of the connection:

- If new connection ⇒ check against security policy [1]
- If existing connection ⇒ look it up in the table and update it, if necessary

Still hard to filter stateless protocols (UDP) and ICMP. Moreover, it can be bypassed with IP tunneling.

Iptables is a software used to set up, maintain, and inspect the tables of IPv4 packet filter rules in the Linux kernel. Several different tables may be defined, each containing built-in chains and potentially user-defined chains.

A **chain** is a list of rules which can match a set of packets. Each rule specifies criteria for a packet and an associated target, namely what to do with a packet that matches the pattern.

If the packet does not match a rule, the next rule in chain is examined. If it matches, then the next rule is specified by the value of the target. The standard targets are: accept, drop, queue [2] and return [3].

Built-in chains:

- PREROUTING: Chain processed before any routing decisions have been made regarding where to send the packet.
- INPUT: Incoming packet, it is going to be locally delivered.
- FORWARD: All packets that have been routed and were not for local delivery will traverse this chain.
- OUTPUT: Outgoing packet, sent from the machine itself. It is important to consider also these packets, to avoid *Information leakage*
- POSTROUTING: Routing decision has been made. Any outgoing or forwarded packet enter this chain just before handing them off to the hardware.

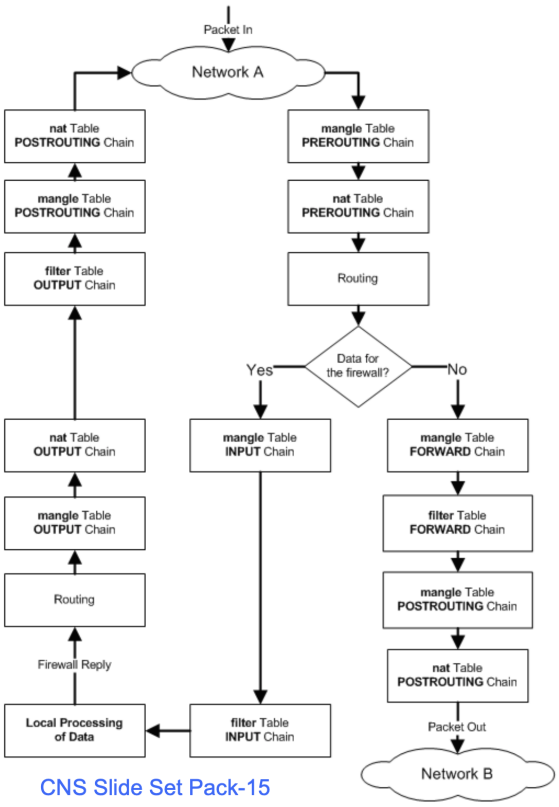
[1] Usually, for default, the firewall is set to use a white list, which denies everything that is not allowed in the list

[2] Pass the packet to the userspace, the behavior depends on the queue handler

[3] Stop traversing this chain and resume at the next rule in the previous (calling) chain

Standard tables:

- FILTER: default table, contains the built-in chain INPUT, FORWARD and OUTPUT
- NAT: (Network Address Translation) used to transform the destination IP address to be compatible with the firewall's routing table. Contains PREROUTING, OUTPUT and POSTROUTING
- MANGLE: TCP header modification. Contains PREROUTING, INPUT, OUTPUT, FORWARD and POSTROUTING
- RAW: to provide a mechanism for marking packets in order to opt-out of connection tracking.



The explanation of the real usage of the iptables is in the next section.

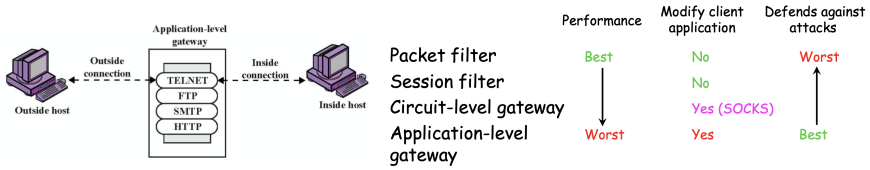
2.3 Application-level gateway

An application level gateway is implemented through a **proxy server**, which acts as an intermediary between a client and a server.

A client application from within the protected network may request services originating from less secure networks such as the internet. After the client's authentication has been confirmed, the requests for services are relayed onwards by the proxy server, provided that they are allowed by the security policies in force. All subsequent data exchanges in relation to the service request are handled by the proxy server.

From the view of both the clients within the protected network and the remote servers, the proxy server is seen as the end user. The originating client and the remote server are hidden from each other [4].

To notice that each service needs its own proxy server.



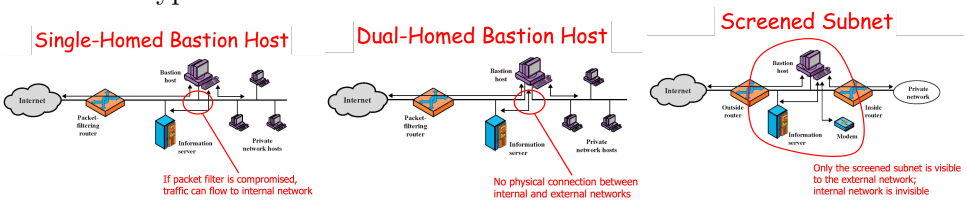
2.4 Where to place the Firewall

The **Bastion host** is a hardened system implementing application-level gateway behind packet filter, all the traffic flows through it. It has the following characteristics:

- Trustable operating systems: run only few applications and all non-essential services are turned off, to reduce vulnerabilities (as a simple compiler could become).
- Application-specific proxies for supported services:
  - Each proxy supports only a subset of application's commands
  - Traffic is logged and audited (to analyze attacks)
  - Disk access restricted, read-only file system
  - Runs as a non-privileged user in a separate directory
- Support for user authentication

Servers inside the network should be reachable from the outside, without compromising the entire system. To allow these interactions, the *DeMilitarized Zone (DMZ)* is set to allow external connections/users to reach these servers but still avoiding the access to the internal network, blocked by the Bastion host.

3 main types of connection:



2.5 General Problems with Firewalls

- Interfere with networked applications
- Don't solve the real problems, like Buggy software (ex. buffer overflow exploits) or Bad protocol design (ex. WEP in 802.11b)
- Generally don't prevent denial of service (DoS)
- Don't prevent insider attacks
- Increasing complexity and potential for misconfiguration

3 Iptables in practice

3.1 Preliminaries

For the initial setup of the VMs network, I will reuse the same considerations of the last homework. Before the beginning of the tests, here is a list of the basic commands:

- **-t** ⇒ ⟨ - *-table* table ⟩ specifies the packet matching table which the command should operate on (filter as default).
- **-A** ⇒ ⟨ - *-append* chain rule-specification ⟩ Append one or more rules to the end of the selected chain.
- **-D** ⇒ ⟨ - *-delete* chain rule-specification / chain rulenum ⟩ Delete one or more rules from the selected chain (can both be selected by number or specification).
- **-I** ⇒ ⟨ - *-insert* chain [rulenum] rule-specification ⟩ Insert one or more rules in the selected chain as the given rule number (default 1).
- **-R** ⇒ ⟨ - *-replace* chain rulenum rule-specification ⟩ Replace a rule in the selected chain.
- **-L** ⇒ ⟨ - *-list* [chain] ⟩ List all rules in the selected chain (If no chain is selected, all chains are listed).
- **-F** ⇒ ⟨ - *-flush* chain ⟩ Flush the selected chain (equivalent to deleting all the rules one by one).
- **-Z** ⇒ ⟨ - *-zero* chain ⟩ Zero the packet and byte counters in all chains.
- **-N** ⇒ ⟨ - *-new-chain* chain ⟩ Create a new user-defined chain by the given name.
- **-X** ⇒ ⟨ - *-delete-chain* chain ⟩ Delete the optional user-defined chain specified.
- **-P** ⇒ ⟨ - *-policy* chain target ⟩ Set the policy for the chain to the given target.

For parameters, options and match extensions I will directly use the man page.

3.2 Some tests

First, let's try with **Ping**, a network administration utility used to check the connectivity status between a source and a destination computer/device over an IP network.

It implements the ICMP(Internet Control Message Protocol) to send and receive echo messages from and to the host and destination computers respectively to tell us about the performance of the network. An ICMP request message is sent to the destination computer; if the destination IP address is available, it sends an ICMP message response to the host computer. This tells us about the connectivity status of the network such as round-trip time-the time taken to send and receive a packet of information [6]. The basic command:

ping 172.17.0.3 (IP)

```
PING 172.17.0.3 (172.17.0.3) 56(84) bytes of data.
64 bytes from 172.17.0.3: icmp_seq=1 ttl=64 time=0.177 ms
64 bytes from 172.17.0.3: icmp_seq=2 ttl=64 time=0.195 ms
64 bytes from 172.17.0.3: icmp_seq=3 ttl=64 time=0.195 ms
^C
--- 172.17.0.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2057ms
```

Now, with the basic rules of the Firewall, the connection works fine. If the other VM does not want to accept anymore message from this machine, the simplest command it can use is:

sudo iptables -A INPUT -s 172.17.0.2 -j DROP

```
--- 172.17.0.3 ping statistics ---
8 packets transmitted, 0 received, 100% packet loss, time 7297ms
```

```
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       all  --  172.17.0.2           anywhere

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Now, the other VM does not accept anymore packets from our IP. However, let's delete this command and try some other protocols and network utilities, to remove the new rule:

iptables -D INPUT 1

For the second test I will exchange some files using the simple and well known **FTP** (File Transfer Protocol). To setup the server on one of the 2 machines, I will use vsFTPd, which after some configurations simply allows to build the server by:

sudo service vsftpd start

Now, on the other machine, to start the conversation:
ftp 172.17.0.3

And then use the *put* commnand to send the file.

```
user1@c88ab4cadada:~$ ftp 172.17.0.3
Connected to 172.17.0.3.
220 (vsFTPd 3.0.3)
Name (172.17.0.3:user1): user1
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> put a.txt
local: a.txt remote: a.txt
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 Transfer complete.
46 bytes sent in 0.00 secs (591.0773 kB/s)
```

As showed in the picture, the file has been transferred without problems. Now, knowing the FTP protocol sends commands over the port 21, the server can block it by:
iptables -A INPUT -p tcp - -destination-port 21 -j DROP

```
ftp> put a.txt
local: a.txt remote: a.txt
421 Timeout.
```

If, for instance, I want to allow only FTP from the other machine:
iptables -I INPUT -s 172.17.0.2 -p tcp –destination-port 21 -j ACCEPT

Notice that is important to use -I inserted of -A, or it will append the rule at end, then the message will be dropped before finding the new rule.

It's time for **Secure Shell (SSH)**, which is a cryptographic network protocol for operating network services securely over an unsecured network. Typical applications include remote command-line, login, and remote command execution, but any network service can be secured with SSH [7]. It is similar to the well known **telnet**, but it is more secure providing encryption on exchanged data.

For the installation on the client terminal:
sudo apt-get install openssh-client

For the installation on server side:
sudo apt-get install openssh-server ii

To generate the keys, run:
ssh-keygen

Now, by sharing the key and authentication informations with the ssh server, the client will be able to control the server.

```
user1@c88ab4cadada:~$ ssh 172.17.0.4
user1@172.17.0.4's password:
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 4.9.184-linuxkit x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage
This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

user1@f385bdcaaa4d:~$ touch file.txt
```

```
root@f385bdcaaa4d:/home/user1# ls
file.txt
```

3.3 Main activity

First at all, we want to block all the possible network traffic of the machine. This can be easily done by changing the policy:
iptables -P chain DROP

```
root@ce8838375880:/# sudo iptables -L
Chain INPUT (policy DROP)
target     prot opt source               destination

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
```

Now, it is possible to create the equivalent of a white a list, to allow only required services. First, as sen in tests, let's allow the machine to receive the ICMP messages from ping:
iptables -A INPUT -p icmp –icmp-type echo-request -j ACCEPT

Unfortunately, here, using only this rule, the other machine will not receive the replies. To have the round-trip, also the output needs to be set:
iptables -A OUTPUT -p icmp –icmp-type echo-reply -j ACCEPT

Those commands gives the same results already seen during the tests, but this time everything is refused except the inbound ICMP.
Furthermore, the machine can now receive icmp messages from outside, but specifying the *–icmp-type*, it is not allowed to do the same. To allow this behavior:

iptables -A INPUT -p icmp –icmp-type echo-reply -j ACCEPT

iptables -A OUTPUT -p icmp –icmp-type echo-request -j ACCEPT

For instance, now I want to allow ping only from a specific IP:
iptables -A INPUT -s 172.17.0.2 -p icmp –icmp-type echo-request -j ACCEPT

iptables -A OUTPUT -d 172.17.0.2 -p icmp –icmp-type echo-reply -j ACCEPT

At the moment, it only works using an IP, but if I want to ping using the mnemonic, I need to able to connect to the DNS, to resolve the IP. It can be done by exchanging the icmp-types:

iptables -A OUTPUT -p udp –dport 53 –sport 1024: -j ACCEPT

iptables -A INPUT -p udp –sport 53 –dport 1024: -j ACCEPT

Someone can think to simply allow all the outbound traffic to avoid to specify every time both rules. This is not a good idea, if the attacker gain the access to the machine, he/she can easily obtain all the informations he/she needs. Moreover, will be easier to make attacks to other devices using our device as proxy.

The size of the ICMP packets received by ping can be easily changed using the -s parameter (from the standard 64). To avoid big incoming packets, or simply limit the quantity of ICMP messages exchanged in a single connection:

iptables -I INPUT -p icmp –icmp-type echo-request -m connbytes
–connbytes *num_bytes* –connbytes-dir *original, reply, both*
–connbytes-mode *packets, bytes, avgpkt* -j ACCEPT

The same match rule, for instance, can be applied also to FTP, to avoid the exchange of big files.

Now, let's introduce the FTP, allowing the corresponding ports:
iptables -A INPUT -p tcp -m tcp –dport 21 -m
conntrack –ctstate ESTABLISHED,NEW -j ACCEPT

iptables -A OUTPUT -p tcp -m tcp –sport 21 -m
conntrack –ctstate ESTABLISHED -j ACCEPT

Here we can see the power of a stateful firewall, which can understand if there is a new a connection or an already opened one. The 21 port is not enough, it requires also the "ftp data-port" 20:
iptables -A OUTPUT -p tcp -m tcp –sport 20 -m
conntrack –ctstate ESTABLISHED,NEW -j ACCEPT

iptables -A INPUT -p tcp -m tcp –dport 20 -m
conntrack –ctstate ESTABLISHED,RELATED -j ACCEPT

Notice that, when the FTP works in passive mode, it can use different ports (obviously greater than 1024). To allow this behavior:
iptables -A INPUT -p tcp -m tcp –sport 1024: –dport 1024:
-m conntrack –ctstate ESTABLISHED

iptables -A OUTPUT -p tcp -m tcp –sport 1024: –dport 1024:
-m conntrack –ctstate ESTABLISHED,RELATED

As for the ftp, the same approach can be used to allow ssh connections, just by changing the port used by the protocol:
iptables -A INPUT -p tcp –dport 22 -m state
–state NEW,ESTABLISHED -j ACCEPT

iptables -A OUTPUT -p tcp –sport 22 -m state
–state ESTABLISHED -j ACCEPT

The **curl** command allows to send http requests by shell. To allow it, we need first to allow the DNS as seen before, then the corresponding http/https ports:
iptables -I OUTPUT -p tcp -m multiport
–dports 80,443 -j ACCEPT

And for last, allow all related and established packets:

iptables -A INPUT -p tcp -m state –state RELATED,ESTABLISHED -j ACCEPT

```
root@3340529bbc70:/home/user1# iptables -L
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     udp  --  anywhere             anywhere             udp spt:domain dpts:1024:65535
ACCEPT     tcp  --  anywhere             anywhere             state RELATED,ESTABLISHED

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere             multiport dports http,https
ACCEPT     udp  --  anywhere             anywhere             udp spts:1024:65535 dpt:domain
root@3340529bbc70:/home/user1# curl www.google.com
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="it"><head><meta content
"><meta content="/images/branding/googleg/1x/googleg_standard_color_128dp.png" itemprop="image"><tit
```

An interesting rule to limit syn to the server and avoid DoS attacks:
iptables -I INPUT -p tcp –syn -m limit –limit 1/s -j ACCEPT

As seen, the Iptables commands can easily been assembled, mixing the huge amount of parameters, options and rule matches, to build a complex and efficient firewall.

References

[1] course slides

[2] https://en.wikipedia.org/wiki/Firewall_(computing)

[3] https://www.digitalocean.com/community/tutorials/a-deep-dive-into-iptables-and-netfilter-architecture

[4] https://www.open.edu/openlearn/science-maths-technology/computing-and-ict/systems-computer/network-security/content-section-9.5

[5] https://linux.die.net/man/8/iptables

[6] https://vitux.com/linux-ping-command/

[7] https://en.wikipedia.org/wiki/Secure_Shell

[8] https://en.wikipedia.org/wiki/Docker_(software)

[9] https://kapeli.com/cheat_sheets/Netcat.docset/Contents/Resources/Documents/index