Lab 2

IPTables

Contents

2.1 Objective

Iptables is the user space command line program used to configure the IPv4 packet filtering ruleset in Linux 2.4.x and 2.6.x. It is targeted towards system administrators. In addition, iptables can be also used to configure any variant of Network Address Translation (NAT), such as static NAT, dynamic NAT and NAT overload (also known as Port Address Translation (PAT)).

The objective of this session is to familiarize with the basic terminal commands of iptables.

2.2 General iptables description

When a packet first enters the firewall, it hits the hardware and then gets passed on to the proper device driver in the kernel. Then the packet starts to go through a series of steps in the kernel, before it is either sent to the correct application (locally), or forwarded to another host - or whatever happens to it (e.g. dropped). Figure 2.1 illustrates these steps.

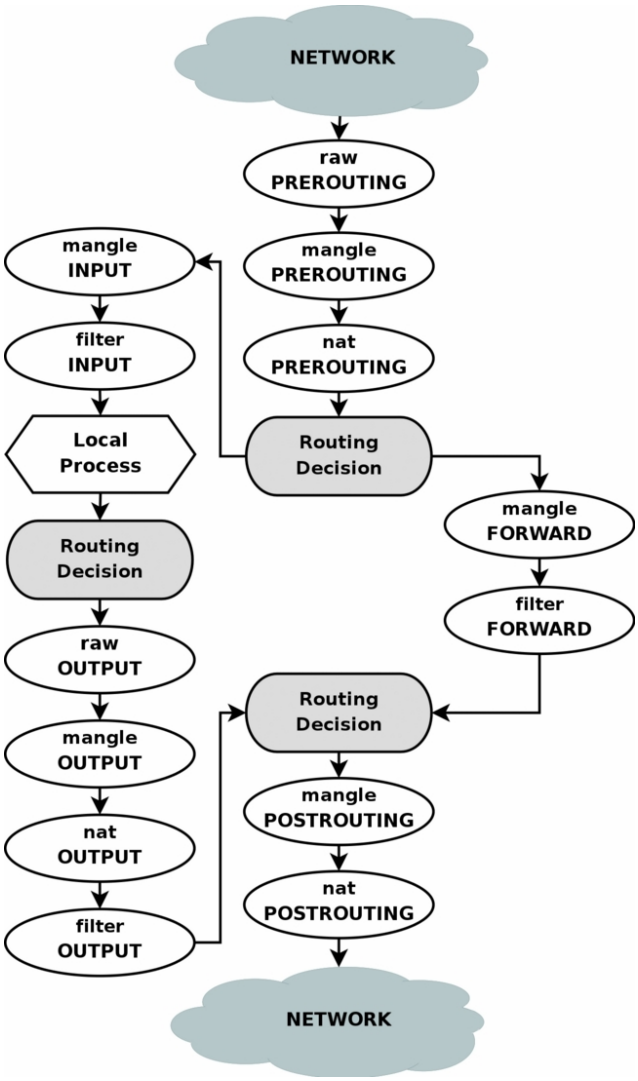

Figure 2.1: Graphical representation of the iptables states.

To clarify Figure 2.1, consider this. If we get a packet into the first routing decision that is not destined for the local machine itself, it will be routed through the FORWARD chain. If the packet is, on the other hand, destined for an IP address that the local machine is listening to, we would send the packet through the INPUT chain and to the local machine.

Also worth a note, is the fact that packets may be destined for the local machine, but the destination address may be changed within the PREROUTING chain by doing NAT. Since this takes place before the first routing decision, the packet will be looked upon after this change. Because of this, the routing may be changed before the routing decision is done. Do note that all packets will be going through one or the other path in this image. If you DNAT (destination NAT, when you change the destination IP of a packet going through a NAT process in router or a firewall) a packet back to the same network that it came from, it will still travel through the rest of the chains until it is back out on the network.

2.3 Tables and Chains description

As you have observed in the Figure 2.1, iptables is organized in tables and chains, each one having a concrete purpose. Following, we provide a brief description about the usage of each table and chain:

NAT table The nat table is used mainly for Network Address Translation. "NAT"ed packets get their IP addresses altered, according to our rules. Packets in a stream only traverse this table once. We assume that the first packet of a stream is allowed. The rest of the packets in the same stream are automatically "NAT"ed or Masqueraded etc, and will be subject to the same actions as the first packet. These will, in other words, not go through this table again, but will nevertheless be treated like the first packet in the stream. This is the main reason why you should not do any filtering in this table, which we will discuss at greater length further on. The PREROUTING chain is used to alter packets as soon as they get in to the firewall. The OUTPUT chain is used for altering locally generated packets (i.e., on the firewall) before they get to the routing decision. Finally we have the POSTROUTING chain which is used to alter packets just as they are about to leave the firewall.

Mangle table This table is used mainly for mangling packets. Among other things, we can change the contents of different packets and that of their headers. Examples of this would be to change the TTL (time-to-live), TOS (type-of-service) or MARK. Note that the MARK (mark the packet with an identifier or sequence number) is not really a change to the packet, but a mark value for the packet is set in kernel space. Other rules or programs might use this mark further along in the firewall to filter or do advanced routing on; tc is one example. The table consists of five built in chains, the PREROUTING, POSTROUTING, OUTPUT, INPUT and FORWARD chains. PREROUT-

ING is used for altering packets just as they enter the firewall and before they hit the routing decision. POSTROUTING is used to mangle packets just after all routing decisions have been made. OUTPUT is used for altering locally generated packets after they enter the routing decision. INPUT is used to alter packets after they have been routed to the local computer itself, but before the user space application actually sees the data. FORWARD is used to mangle packets after they have hit the first routing decision, but before they actually hit the last routing decision. Note that mangle can't be used for any kind of Network Address Translation or Masquerading, the nat table was made for these kinds of operations.

Raw table The raw table and its chains are used before any other table in netfilter. It was introduced to use the NOTRACK target. This table is rather new and is only available, if compiled, with late 2.6 kernels and later. The raw table contains two chains. The PREROUTING and OUTPUT chain, where they will handle packets before they hit any of the other netfilter subsystems. The PREROUTING chain can be used for all incoming packets to this machine, or that are forwarded, while the OUTPUT chain can be used to alter the locally generated packets before they hit any of the other netfilter subsystems.

Filter table The filter table should be used exclusively for filtering packets. For example, we could DROP, LOG, ACCEPT or REJECT packets without problems, as we can in the other tables. There are three chains built in to this table. The first one is named FORWARD and is used on all non-locally generated packets that are not destined for our local host (the firewall, in other words). INPUT is used on all packets that are destined for our local host (the firewall) and OUTPUT is finally used for all locally generated packets.

2.4 Iptables commands

For this session, we will use the same Ubuntu distribution used in the previous section. You may re-download the Ubuntu 18.04 linux image from the FIB software repository so you have a fresh VM or re-use the VM you already used and configured in the previous section. If you choice to use a new VM, remember to not install it but rather configure the VM with the "I will install the OS later" option checked, then in the configuration option load the VM in the virtual CD drive, start the virtual machine and selected the "run without installation" starting option.

Enable the root user:

```
# sudo passwd root
```

enter password 'toor', and switch to the root user (su).

Typing:

```
iptables -L
```

lists your current rules in iptables (default table is **filter**). If you have just set up your server, you will have no rules, and you should see

```
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
```

As you can see, there are three chains (INPUT, FORWARD and OUTPUT), and the default policy is ACCEPT.

Here are explanations for some of the iptables options you will see in this session. Don't worry about understanding everything here now, but remember to come back and look at this list as you encounter new options later on.

-L List the current filter rules.

-A Append this rule to a rule chain. Valid chains for what we're doing are INPUT, FORWARD and OUTPUT, but we mostly deal with INPUT in this tutorial, which affects only incoming traffic.

-D Delete this rule from rule chain.

-m conntrack Allow filter rules to match based on connection state. Permits the use of the –ctstate option.

–**ctstate** Define the list of states for the rule to match on. Valid states are:

- NEW - The connection has not been seen yet.
- RELATED - The connection is new, but is related to another connection already permitted.
- ESTABLISHED - The connection is already established.
- INVALID - The traffic couldn't be identified for some reason.

-m limit Require the rule to match only a limited number of times. Allows the use of the –limit option. Useful for limiting logging rules.

- –limit - The maximum matching rate, given as a number followed by "/second", "/minute", "/hour", or "/day" depending on how often you want the rule to match. If this option is not used and -m limit is used, the default is "3/hour".

-p The connection protocol used.

–**dport** The destination port(s) required for this rule. A single port may be given, or a range may be given as start:end, which will match all ports from start to end, inclusive.

-j Jump to the specified target. By default, iptables allows four targets:

- ACCEPT - Accept the packet and stop processing rules in this chain.
- REJECT - Reject the packet and notify the sender that we did so, and stop processing rules in this chain.
- DROP - Silently ignore the packet, and stop processing rules in this chain.
- LOG - Log the packet, and continue processing more rules in this chain. Allows the use of the –log-prefix and –log-level options.

–**log-prefix** When logging, put this text before the log message. Use double quotes around the text to use.

–**log-level** Log using the specified syslog level. 7 is a good choice (debug-level messages) unless you specifically need something else.

-i Only match if the packet is coming in on the specified interface.

-I Inserts a rule. Takes two options, the chain to insert the rule into, and the rule number it should be.

- -I INPUT 5 would insert the rule into the INPUT chain and make it the 5th rule in the list.

-v Display more information in the output. Useful if you have rules that look similar without using -v.

-s –**source** address[/mask] source specification

-d –**destination** address[/mask] destination specification

-o –**out-interface** output name[+] network interface name ([+] for wildcard).

-t –**table** table to manipulate (default: 'filter').

-P –**policy** sets the default policy for the chain to the given target.

2.4.1 Allowing established sessions

We can allow established sessions to receive traffic:

```
iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED
-j ACCEPT
```

The above rule has no spaces either side of the comma in ESTABLISHED,RELATED. It appends in the INPUT chain of the default table (filter) the specified rule, which ACCEPTs incoming connections that are already ESTABLISHED or RELATED to already established ones. Of course, before this command, the firewall (actually, the filter table) was already accepting such incoming connections but, when we will later append a DROP rule as the last one, all the traffic

that is not explicitly allowed is dropped. Therefore, a canonical way of proceeding is adding ACCEPT rules and then blocking all the traffic other than the specified.

If the line above doesn't work, you may be on a VPS (Virtual Private Server) that uses OpenVZ or doesn't have some kernel extensions installed. In that case, try this line instead:

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

2.4.2 Allowing incoming traffic on specific ports

You could start by blocking traffic, but you might be working over SSH, where you would need to allow SSH before blocking everything else.

To allow incoming traffic on the default SSH port (22), you could tell iptables to allow all TCP traffic on that port to come in.

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

Referring back to the list above, you can see that this tells iptables to:

- append this rule to the input chain (-A INPUT) so we look at incoming traffic
- check to see if it is TCP (-p tcp).
- if so, check to see if the input goes to the SSH port (–dport ssh).
- if so, accept the input (-j ACCEPT).

Let's check the rules: (only the first few lines shown, you will see more)

```
iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere            state
    RELATED,ESTABLISHED
ACCEPT     tcp  --  anywhere             anywhere            tcp dpt:
    ssh
```

Now, let's allow all incoming web traffic (port 80):

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

Checking our rules (iptables -L), we have:

```
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere            state
    RELATED,ESTABLISHED
ACCEPT     tcp  --  anywhere             anywhere            tcp dpt:
    ssh
ACCEPT     tcp  --  anywhere             anywhere            tcp dpt:
    www
```

We have specifically allowed tcp traffic to the ssh and web ports, but as we have not blocked anything, all traffic can still come in (remember that the default policy for the filter table is ACCEPT).

2.4.3 Blocking traffic

Once a decision is made to accept a packet, no more rules affect it. As our rules allowing ssh and web traffic come first, we can still accept the traffic we want as long as our rule to block all traffic comes after them. All we need to do is put the rule to block all traffic at the end:

```
iptables -A INPUT -j DROP
```

that is, drop whatever incoming traffic which is not explicitly allowed. List the rules:

```
iptables -L
```

You should see:

```
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere            state
    RELATED,ESTABLISHED
ACCEPT     tcp  --  anywhere             anywhere            tcp dpt:
    ssh
ACCEPT     tcp  --  anywhere             anywhere            tcp dpt:
    www
DROP       all  --  anywhere             anywhere
```

Because we didn't specify an interface or a protocol, any traffic for any port on any interface is blocked, except for web and ssh.

2.4.4 Editing iptables

The only problem with our setup so far is that even the loopback port is blocked.

Try to ping the loopback interface:

```
ping 127.0.0.1
```

it does not work; let's temporary remove the last blocking rule:

```
iptables -D  INPUT -j DROP
```

and try to ping again the loopback; now it works. Let's insert again the blocking rule (iptables -A INPUT -j DROP) to go back to the previous state. Check with iptables -L that everything is like before (dropping not allowed traffic).

We could have written the drop rule for just *eth0* by specifying *-i eth0*, but we could also add a rule for the loopback. If we append this rule, it will come too late - after all the traffic has been dropped. We need to insert this rule before that. Since this is a lot of traffic (many local processes use the loopback interface to communicate with themselves), we'll insert it as the first rule so it's processed first.

```
iptables -I INPUT 1 -i lo -j ACCEPT
```

The -I specifies to insert the rule at the specified position (1); the -i specifies the target interface (lo: loopback). Let's check how the iptables looks like now (iptables -L):

```
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere
ACCEPT     all  --  anywhere             anywhere            state
    RELATED,ESTABLISHED
ACCEPT     tcp  --  anywhere             anywhere            tcp dpt:
    ssh
ACCEPT     tcp  --  anywhere             anywhere            tcp dpt:
    www
DROP       all  --  anywhere             anywhere
```

The first and last lines look nearly the same, so we will list iptables in greater detail.

```
iptables -L -v

Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source
    destination
    0     0 ACCEPT     all  --  lo     any     anywhere
    anywhere
    0     0 ACCEPT     all  --  any    any     anywhere
    anywhere             state RELATED,ESTABLISHED
    0     0 ACCEPT     tcp  --  any    any     anywhere
    anywhere             tcp dpt:ssh
    0     0 ACCEPT     tcp  --  any    any     anywhere
    anywhere             tcp dpt:www
    0     0 DROP       all  --  any    any     anywhere
    anywhere
```

You can now see more information. This rule is actually very important, since many programs use the loopback interface to communicate with each other. If you don't allow them to talk, you could affect those programs!

2.4.5 Logging

In the above examples none of the traffic will be logged. If you wanted to log dropped packets to syslog, this would be the quickest way:

```
iptables -I INPUT 5 -m limit --limit 5/min -j LOG
--log-prefix "iptables denied: " --log-level 7
```

2.4.6 Other commands

To list the rules, use the command:

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

Adding the option $--line-numbers$ at the end of the previous command, you can see the number assigned to each line. In this way, it would be easier to delete a specific line when needed using its number. For example, if you want to delete the line number 4 from the INPUT rule set, use the command:

```
iptables -D INPUT 4
```

Another way to delete a line is to repeat the same command used to enter it but replacing the option $-A$ with $-D$. For example:

```
iptables -D INPUT -s 202.54.1.1 -j DROP
```

Finally, it is possible to save the rule set in a file:

```
iptables-save > /root/iptables.fw
```

And restore an iptables configuration from a file:

```
iptables-restore < /root/iptables.fw
```

2.5 Exercises

2.5.1 Rules without considering the package state

First, flush the iptables rules:

```
iptables -F
```

Now:

1. Show the active iptables rules
2. Change the default policy of the OUTPUT chain to DROP
3. Create a rule for accessing the web `www.upc.edu`. Note that, you need the DNS service (see /etc/resolv.conf) to find the corresponding IP address of the url name. DNS uses UDP transport protocol and port 53. Since it is a VM, you have to solve it through the loopback (interface lo). In fact, the DNS in a VM uses a service called systemd-resolved which is listening (thus, both INPUT and OUTPUT) the localhost [1].
4. Imagine that the child of a friend is becoming a very good hacker because he visits quite often web pages like `www.metasploit.com`. Your friend wants to be permissive with his child but as a good father he wants to know what the child is doing. Write down a rule to log the connections to `www.metasploit.com`.
5. Now, change all the default policies to DROP. Can you access to the previous web sites `www.upc.edu` and `www.metasploit.com`? Why?
6. Add the corresponding iptables rules to access to the previous sites but maintaining the default DROP policy of the INPUT and OUTPUT chains.
7. Save the final rules in a file called *lab2_no_state.fw* using the save command explained in Section 2.4.6.

2.5.2 Rules considering the package state

1. Repeat the previous firewall configuration taking into account that iptables is able to filter packets considering their session state. In this way it is possible to secure even more our communications. Remember that the firewall has to be configured in the following way:
   - Default chain policies: DROP
   - Allow connections to: `www.upc.edu` and `www.metasploit.com`
   - Allow to resolve DNS requests.

[1] It may be possible that the DNS service is not working properly due to use of mashup in the UPC web site that links contents from other webs (and thus names) not allowed in the iptables rules. If it is the case, use directly the @IP address instead of the url name

- Save the final rules in a file called *lab2_ state.fw* using the save command explained in Section 2.4.6.