Netfilter

The Linux firewall

Netfilter or iptables?

- Netfilter is the name of the firewall used on Linux. It is kernel based, and it allows the kernel to register *callback functions* on every packet that traverses the network stack.
- Iptables is the command that is used to manipulate Netfilter on the Linux box.
- Not only is it used for filtering incoming and outgoing packets, but I can also be used in situations like NAT.
- It has three main tables:
  - Packet filtering
  - Network address translation (NAT)
  - Packet mangling
- Each table may have a chain of rules. The following are the available chains:
  - Prerouting
  - Input
  - Forward
  - Postrouting
  - Output

The chains

- Chains may exist in a table as hook points that match the packet in the desired state.
- They can be explained as follows:
  - FORWARD: when your box is acting as a router, this matches the packet when it's coming on one interface and going out of another
  - INPUT: when the packet is incoming to your box, just before it reaches the process
  - OUTPUT: when the packet is leaving your box, just after the process generates it
  - POSTROUTING: right before the packet leaves the interface
  - PREROUTING: right after the packet enters the network interface
- By definition, those chains are available to the table according to the table's role. For example, it is meaningless to target the POSTROUTING chain in the INPUT table.

The tables

- As mentioned, iptables consists of three tables used to organize the hook points (chains) on the kernel to capture packets in the desired point in path. They can be summarized as follows:
  - Nat: used to redirect connections to different network interfaces. It contains the following chains: OUTPUT, POSTROUTNG, AND PREROUTING
  - Filter: used to control packets flowing in or out of the box. It contains the following chains: INPUT, OUTPUT, FORWARD. Unless explicitly specified, the filter table is default one used in iptables commands
  - Mangle: used to alter the packet. It's rarely used. The available chains are: FORWARD, INPUT, OUTPUT, POSTROUTING, and PREROUTING.

The packet journey in the chain

- As soon as the packet reached the network card, it traverses the chains of the tables according to their direction and destination.
- Packets are examined at each point to the chain until a *rule* is matched. When that happens, the target is applied. This target may be ACCEPT, REJECT, DROP, or LOG.
- If the packet reaches the end of the chain with no rules matched, the default rule is applied
- If no default rule is specified, a packet that does not match any of the rules are considered *accepted* and they pass
- Whenever a packet is matched, the packet and byte counter is incremented to reflect the change.

The targets

- You can specify one of the following targets to a packet:
  - ACCEPT: stop processing at the current chain and let the packet flow to the next one
  - DROP: stop processing at the current chain as well as the rest of the chains. Just dump the packet as if it did not arrive
  - REJECT: same as DROP but some feedback is returned to the user
  - LOG: useful for troubleshooting. Same as ACCEPT but it writes to the /var/log/messages. You can use it to ensure that a packet has passed or failed to pass a specific chain

Iptables configuration

- Iptables can be started and stopped by using the service command just like other services on the system:
  `service iptables start/stop`
- You can use `chkconfig` to ensure that the service gets started on system boot, at the desired runlevels
- The rules are stored in `/etc/sysconfig/iptables` file
- The iptables command itself has a large number of options and possible combinations. However, you needn't study all the possible syntaxes for your day to day system administration. The rest of this section will describe the most common uses of the iptables command.

I want to block that IP!

- The most basic usage of iptables is to block traffic from a specific IP address:
  `iptables –A INPUT –s 192.168.0.103 – j DROP`
- As mentioned, the filter table is assumed by default if not explicitly specified. We could, however mention it like this:
  `iptables -t filter –A INPUT –s 192.168.0.103 – j DROP`
- Then comes the action argument. The –A is for add, this adds the rule to the end of the chain. You can use –I to insert a rule at a specific place (using line numbers). Use `iptables –vnL` to display the rules and their line numbers. The action is followed by the chain. In our case it's the INPUT one
- The –s argument is for the *source* IP address. This can also be a range of addresses like 192.168.0.0/24 to block the whole subnet
- The –j argument specifies the target for the rule. In our case, a packet is matched coming from this IP address, drop it.

Let's block web traffic

- You can block traffic targeting a specific port number. For example HTTP web traffic:
  `iptables –I INPUT –p tcp --dport 80 -j DROP`
- We used the –p command argument to specify the TCP protocol. You can replace it udp for the UDP protocol.
- The --dport specifies that the matching packet is the one that wants to reach port 80 on the box. You can also replace the port number with the name as specified in /etc/services
- You can combine this with the previous example to block clients from a specific IP or IP range (hackers?) using a command like the following:
  `iptables –I INPUT –p tcp --dport 80 -s 192.168.0.103 –j DROP`

Controlling connection state

- In very tight environments, outgoing connections are also filtered. But this may cause problems when an incoming connection needs to send response.
- For example, you may want to limit outgoing ssh connections to a specific host, but allow this host to establish connection with you.
- In this case, you can use the --state predicate as follows:

```
iptables -A INPUT -p tcp --dport 22 -s 192.168.0.103 -m state
--state NEW,ESTABLISHED -j ACCEPT
```

- Now even if you block outgoing connections, clients will be able to connect to you via ssh and receive responses from your server.

The LOG action

- It is used for troubleshooting purposes.
- You can use it to determine whether or not a packet has passed for failed a specific chain. For example:
  `iptables –A INPUT –j LOG`
- By examining the `/var/log/messages` you will know whether or not the packets have been blocked or not

I don't want to be *ping*ed

- You can prevent other machines from pinging yours by blocking the ICMP packets as follows:
  ```
  iptables –A INPUT -p icmp --icmp-type echo-request –j DROP
  ```
- You can also choose the spcific network interface on which you want to apply the rule (any rule) by using the –i switch:
  ```
  iptables –A INPUT –i eth0 -p icmp --icmp-type echo-request –j DROP
  ```

The usual configuration

- Most of the time you don't want to block specific connections, instead you want to allow *only* specific connections.
- This can be done using the ACCEPT action for all the connections that you want to pass, then add one final DROP or REJECT rule to block any packet that does not match any of your rules.
- For example, the below configuration is for a system that allows only SSH and HTTP TCP connections:

```
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
iptables -A INPUT -p tcp --dport 22 -j ACCEPT
iptables -A INPUT -j LOG
iptables -A INPUT -j DROP
```

- This can be also a range of ports like 8000:9000
- The LOG rule here is for troubleshooting. If clients complain that they still cannot connect to port 80 or cannot ssh to the machine, check the log file for blocked packets.
- Any of the previous examples can be combined with the above rules to make them even more specific (like source IP)
- You can even control the source port of the connection using the --sport option

Viewing, editing, and saving rules

- You can use `iptables -vnL` to view all the configured rules. The line numbers are important here to be able to make changes.
- You can *insert* a rule in a specific location using the -I switch, but you must specify a line number. For example, I want to block all traffic from subnet 172.25.22.0/24, and if I have traffic from other sources, I want only ssh connections allowed.

```
iptables -A INPUT -s 172.25.22.0/24 -j DROP
iptables -A INPUT -p tcp --dport 22 -j ACCEPT
iptables -A INPUT -j DROP
```

- Then I installed a web server. I want the pervious configuration to stay the same, but just add web traffic to the allowed list. Consuling iptables –vnL, I want this rule to be added to line 2:

```
iptables -I INPUT 2 -p tcp --dport 80 -j ACCEPT
```

- If you want to delete a rule use the –D switch:

```
iptables -D INPUT 2
```

- You must save the iptables configuratoin to persis reboots:

```
service iptables save
```

- you can flush the rules (delete them) using `iptables -F`

LAB: configure a Linux box to act as a NAT device

- Our lab consists of two machines:
  - Machine 1: will act as the NAT device. It has 2 network interfaces one connected to the host (the outside world: eth0) and the other is connected to a private network (eth1).
  - Machine 2: has only one network card that is connected to the private network. This machine is not reachable by the host.
- Our prerequisites: we want to enable SSH access to machine 2 through machine 1 port 2222. Machine 1 will use iptables to forward the packets to machine 2 on port 22 and *masquerade* the returning packets to make it appear as if they are originating from machine 1.
- Procedure (on machine 1):
  - Enable kernel routing on machine 1 by editing /etc/sysctl.conf and making net.ipv4.ip_forward = 1. Apply changes using sysctl –w
  - Use iptables to enable port forwarding:
    ```
    iptables -t nat -A PREROUTING -p tcp -d 192.168.0.106 --dport 2222 -j DNAT
    --to 192.168.56.1:22
    ```
  - Use iptables to masquerade returning packets:
    ```
    iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
    ```
  - Test the results by connecting from the host to machine 1 through SSH specifying port 2222. You should find yourself connecting to machine 2.