Linux Security

IPTABLES FIREWALL

Table of Contents

List of Tables

1. Introduction to routers

What follows is a very brief introduction to using Linux as a router.

10.1. router or firewall

A router is a device that connects two networks. A firewall is a device that besides acting as a router, also contains (and implements) rules to determine whether packets are allowed to travel from one network to another. A firewall can be configured to block access based on networks, hosts, protocols and ports. Firewalls can also change the contents of packets while forwarding them.

10.2. packet forwarding

Packet forwarding means allowing packets to go from one network to another. When a multihomed host is connected to two different networks, and it allows packets to travel from one network to another through its two network interfaces, it is said to have enabled packet forwarding.

10.3. packet filtering

Packet filtering is very similar to packet forwarding, but every packet is individually tested against rules that decide on allowing or dropping the packet. The rules are stored by iptables.

10.4. stateful

A stateful firewall is an advancement over stateless firewalls that inspect every individual packet. A stateful firewall will keep a table of active connections, and is knowledgeable enough to recognise when new connections are part of an active session. Linux iptables is a stateful firewall.

10.5. nat (network address translation)

A nat device is a router that is also changing the source and/or target ip-address in packets. It is typically used to connect multiple computers in a private address range (rfc 1918) with the (public) internet. A nat can hide private addresses from the internet.
It is important to understand that people and vendors do not always use the right term when referring to a certain type of nat. Be sure you talk about the same thing. We can distuinguish several types of nat.

10.6. pat (port address translation)

nat often includes pat. A pat device is a router that is also changing the source and/or target tcp/udp port in packets. pat is Cisco terminology and is used by snat, dnat, masquerading and port forwarding in Linux. RFC 3022 calls it NAPT and defines the nat/pat combo as "traditional nat". A device sold to you as a nat-device will probably do nat and pat.

10.7. snat (source nat)

A snat device is changing the source ip-address when a packet passes our nat. snat configuration with iptables includes a fixed target source address.

10.8. masquerading

Masquerading is a form of snat that will hide the (private) source ip-addresses of your private network using a public ip-address. Masquerading is common on dynamic internet interfaces (broadband modem/routers). Masquerade configuration with iptables uses a dynamic target source address.

10.9. dnat (destination nat)

A dnat device is changing the destination ip-address when a packet passes our nat.

10.10. port forwarding

When static dnat is set up in a way that allows outside connections to enter our private network, then we call it port forwarding.

10.11. /proc/sys/net/ipv4/ip_forward

Whether a host is forwarding packets is defined in /proc/sys/net/ipv4/ip_forward. The following screenshot shows how to enable packet forwarding on Linux.

```
root@router~# echo 1 > /proc/sys/net/ipv4/ip_forward
```

The next command shows how to disable packet forwarding.

```
root@router~# echo 0 > /proc/sys/net/ipv4/ip_forward
```

Use cat to check if packet forwarding is enabled.

```
root@router~# cat /proc/sys/net/ipv4/ip_forward
```

10.12. /etc/sysctl.conf

By default, most Linux computers are not configured for automatic packet forwarding. To enable packet forwarding whenever the system starts, change the net.ipv4.ip_forward variable in /etc/sysctl.conf to the value 1.

```
root@router~# grep ip_forward /etc/sysctl.conf
net.ipv4.ip_forward = 0
```

10.13. sysctl

For more information, take a look at the man page of sysctl.

```
root@debian6~# man sysctl
root@debian6~# sysctl -a 2>/dev/null | grep ip_forward
net.ipv4.ip_forward = 0
```


10.14. practice: packet forwarding

0. You have the option to select (or create) an internal network when adding a network card in VirtualBox or VMWare. Use this option to create two internal networks. I named them leftnet and rightnet, but you can choose any other name.

1. Set up two Linux machines, one on leftnet, the other on rightnet. Make sure they both get an ip-address in the correct subnet. These two machines will be 'left' and 'right' from the 'router'.

2. Set up a third Linux computer with three network cards, one on leftnet, the other on rightnet. This computer will be the 'router'. Complete the table below with the relevant names, ip-addresses and mac-addresses.

Table 10.1. Packet Forwarding Exercise

leftnet computer the router rightnet computer
MAC
IP

3. How can you verify whether the router will allow packet forwarding by default or not ? Test that you can ping from the router to the two other machines, and from those two machines to the router. Use arp -a to make sure you are connected with the correct mac addresses.

4. Ping from the leftnet computer to the rightnet computer. Enable and/or disable packet forwarding on the router and verify what happens to the ping between the two networks. If you do not succeed in pinging between the two networks (on different subnets), then use a sniffer like wireshark or tcpdump to discover the problem.

5. Use wireshark or tcpdump -xx to answer the following questions. Does the source MAC change when a packet passes through the filter ? And the destination MAC ? What about source and destination IP-addresses ?

6. Remember the third network card on the router ? Connect this card to a LAN with internet

connection. On many LAN's the command dhclient eth0 just works (replace eth0 with the correct interface).

```
root@router~# dhclient eth0
```

You now have a setup similar to this picture. What needs to be done to give internet access to leftnet and rightnet.

10.15. solution: packet forwarding

1. Set up two Linux machines, one on leftnet, the other on rightnet. Make sure they both get an ip-address in the correct subnet. These two machines will be 'left' and 'right' from the 'router'.

The ip configuration on your computers should be similar to the following two screenshots. Both machines must be in a different subnet (here 192.168.60.0/24 and 192.168.70.0/24). I created a little script on both machines to configure the interfaces.

```
root@left~# cat leftnet.sh
pkill dhclient
ifconfig eth0 192.168.60.8 netmask 255.255.255.0
root@right~# cat rightnet.sh
pkill dhclient
ifconfig eth0 192.168.70.9 netmask 255.255.255.0
```

2. Set up a third Linux computer with three network cards, one on leftnet, the other on rightnet. This computer will be the 'router'. Complete the table below with the relevant names, ip-addresses and mac-addresses.

```
root@router~# cat router.sh
ifconfig eth1 192.168.60.1 netmask 255.255.255.0
ifconfig eth2 192.168.70.1 netmask 255.255.255.0
#echo 1 > /proc/sys/net/ipv4/ip_forward
```

Your setup may use different ip and mac addresses than the ones in the table below.

Table 10.2. Packet Forwarding Solution

leftnet computer the router rightnet computer
08:00:27:f6:ab:b9 08:00:27:43:1f:5a 08:00:27:be:4a:6b 08:00:27:14:8b:17
192.168.60.8 192.168.60.1 192.168.70.1 192.168.70.9

3. How can you verify whether the router will allow packet forwarding by default or not ? Test that you can ping from the router to the two other machines, and from those two machines to the router. Use arp -a to make sure you are connected with the correct mac addresses.

This can be done with "grep ip_forward /etc/sysctl.conf" (1 is enabled, 0 is disabled) or with sysctl -a | grep ip_for.

```
root@router~# grep ip_for /etc/sysctl.conf
net.ipv4.ip_forward = 0
```

4. Ping from the leftnet computer to the rightnet computer. Enable and/or disable packet forwarding on the router and verify what happens to the ping between the two networks. If you do not succeed in pinging between the two networks (on different subnets), then use a sniffer like wireshark or tcpdump to discover the problem.

Did you forget to add a default gateway to the LAN machines ? Use route add default gw 'ip-address'.

```
root@left~# route add default gw 192.168.60.1
root@right~# route add default gw 192.168.70.1
```

You should be able to ping when packet forwarding is enabled (and both default gateways are properly configured). The ping will not work when packet forwarding is disabled or when gateways are not configured correctly.

5. Use wireshark or tcpdump -xx to answer the following questions. Does the source MAC change when a packet passes through the filter ? And the destination MAC ? What about source and destination IP-addresses ?

Both MAC addresses are changed when passing the router. Use tcpdump -xx like this:

```
root@router~# tcpdump -xx -i eth1
root@router~# tcpdump -xx -i eth2
```

84
6. Remember the third network card on the router ? Connect this card to a LAN with internet connection. On many LAN's the command dhclient eth0 just works (replace eth0 with the correct interface.

```
root@router~# dhclient eth0
```

You now have a setup similar to this picture. What needs to be done to give internet access to leftnet and rightnet.
The clients on leftnet and rightnet need a working dns server. We use one of Google's dns servers here.

```
echo nameserver 8.8.8.8 > /etc/resolv.conf
```

85

2. Iptables firewall

This chapter introduces some simple firewall rules and how to configure them with iptables. iptables is an application that allows a user to configure the firewall functionality built into the Linux kernel.

11.1. iptables tables

By default there are three tables in the kernel that contain sets of rules.
The filter table is used for packet filtering.

```
root@debian6~# iptables -t filter -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
```

The nat table is used for address translation.

```
root@debian6~# iptables -t nat -L
Chain PREROUTING (policy ACCEPT)
target prot opt source destination
Chain POSTROUTING (policy ACCEPT)
target prot opt source destination
Chain OUTPUT (policy ACCEPT)
target prot opt source destination
```

The mangle table can be used for special-purpose processing of packets.
Series of rules in each table are called a chain. We will discuss chains and the nat table later in this chapter.

11.2. starting and stopping iptables

The following screenshot shows how to stop and start iptables on Red Hat/Fedora/CentOS and compatible distributions.

```
[root@centos6 ~]# service iptables stop
[root@centos6 ~]# service iptables start
iptables: Applying firewall rules [ ok ]
[root@centos6 ~]#
```

Debian and *buntu distributions do not have this script, but allow for an uninstall.

```
root@debian6~# aptitude purge iptables
```


11.3. the filter table

11.3.1. about packet filtering

Packet filtering is a bit more than packet forwarding. While packet forwarding uses only a routing table to make decisions, packet filtering also uses a list of rules. The kernel will inspect packets and decide based on these rules what to do with each packet.

11.3.2. filter table

The filter table in iptables has three chains (sets of rules). The INPUT chain is used for any packet coming into the system. The OUTPUT chain is for any packet leaving the system. And the FORWARD chain is for packets that are forwarded (routed) through the system. The screenshot below shows how to list the filter table and all its rules.

```
[root@RHEL5 ~]# iptables -t filter -nL
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
[root@RHEL5 ~]#
```

As you can see, all three chains in the filter table are set to ACCEPT everything. ACCEPT is the default behaviour.

11.3.3. setting default rules

The default for the default rule is indeed to ACCEPT everything. This is not the most secure firewall.

A more secure setup would be to DROP everything. A package that is dropped will not continue in any chain, and no warning or error will be sent anywhere.

The below commands lock down a computer. Do not execute these commands inside a remote ssh shell.

```
root@debianpaul~# iptables -P INPUT DROP
root@debianpaul~# iptables -P OUTPUT DROP
root@debianpaul~# iptables -P FORWARD DROP
root@debianpaul~# 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
```

11.3.4. changing policy rules

To start, let's set the default policy for all three chains to drop everything. Note that you might lose your connection when typing this over ssh ;-).

```
[root@RHEL5 ~]# iptables -P INPUT DROP
[root@RHEL5 ~]# iptables -P FORWARD DROP
[root@RHEL5 ~]# iptables -P OUTPUT DROP
```

Next, we allow the server to use its own loopback device (this allows the server to access its services running on localhost). We first append a rule to the INPUT chain to allow (ACCEPT) traffic from the lo (loopback) interface, then we do the same to allow packets to leave the system through the loopback interface.

```
[root@RHEL5 ~]# iptables -A INPUT -i lo -j ACCEPT
[root@RHEL5 ~]# iptables -A OUTPUT -o lo -j ACCEPT
```

Looking at the filter table again (omitting -t filter because it is the default table).

```
[root@RHEL5 ~]# iptables -nL
Chain INPUT (policy DROP)
target prot opt source destination
ACCEPT all -- 0.0.0.0/0 0.0.0.0/0
Chain FORWARD (policy DROP)
target prot opt source destination
Chain OUTPUT (policy DROP)
target prot opt source destination
ACCEPT all -- 0.0.0.0/0 0.0.0.0/0
```


11.3.5. Allowing ssh over eth0

This example show how to add two rules to allow ssh access to your system from outside.

```
[root@RHEL5 ~]# iptables -A INPUT -i eth0 -p tcp --dport 22 -j ACCEPT
[root@RHEL5 ~]# iptables -A OUTPUT -o eth0 -p tcp --sport 22 -j ACCEPT
```

The filter table will look something like this screenshot (note that -v is added for more

verbose output).

```
[root@RHEL5 ~]# iptables -nvL
Chain INPUT (policy DROP 7 packets, 609 bytes)
pkts bytes target prot opt in out source destination
0 0 ACCEPT all -- lo * 0.0.0.0/0 0.0.0.0/0
0 0 ACCEPT tcp -- eth0 * 0.0.0.0/0 0.0.0.0/0 tcp dpt:22
Chain FORWARD (policy DROP 0 packets, 0 bytes)
pkts bytes target prot opt in out source destination
Chain OUTPUT (policy DROP 3 packets, 228 bytes)
pkts bytes target prot opt in out source destination
0 0 ACCEPT all -- * lo 0.0.0.0/0 0.0.0.0/0
0 0 ACCEPT tcp -- * eth0 0.0.0.0/0 0.0.0.0/0 tcp spt:22
[root@RHEL5 ~]#
```

11.3.6. Allowing access from a subnet

This example shows how to allow access from any computer in the 10.1.1.0/24 network, but only through eth1. There is no port (application) limitation here.

```
[root@RHEL5 ~]# iptables -A INPUT -i eth1 -s 10.1.1.0/24 -p tcp -j ACCEPT
[root@RHEL5 ~]# iptables -A OUTPUT -o eth1 -d 10.1.1.0/24 -p tcp -j ACCEPT
```

Together with the previous examples, the policy is expanding.

```
[root@RHEL5 ~]# iptables -nvL
Chain INPUT (policy DROP 7 packets, 609 bytes)
pkts bytes target prot opt in out source destination
0 0 ACCEPT all -- lo * 0.0.0.0/0 0.0.0.0/0
0 0 ACCEPT tcp -- eth0 * 0.0.0.0/0 0.0.0.0/0 tcp dpt:22
0 0 ACCEPT tcp -- eth1 * 10.1.1.0/24 0.0.0.0/0
Chain FORWARD (policy DROP 0 packets, 0 bytes)
pkts bytes target prot opt in out source destination
Chain OUTPUT (policy DROP 3 packets, 228 bytes)
pkts bytes target prot opt in out source destination
0 0 ACCEPT all -- * lo 0.0.0.0/0 0.0.0.0/0
0 0 ACCEPT tcp -- * eth0 0.0.0.0/0 0.0.0.0/0 tcp spt:22
0 0 ACCEPT tcp -- * eth1 0.0.0.0/0 10.1.1.0/24
```


11.3.7. iptables save

Use iptables save to automatically implement these rules when the firewall is (re)started.

```
[root@RHEL5 ~]# /etc/init.d/iptables save
Saving firewall rules to /etc/sysconfig/iptables: [ OK ]
[root@RHEL5 ~]#
```

11.3.8. scripting example

You can write a simple script for these rules. Below is an example script that implements the firewall rules that you saw before in this chapter.

```
#!/bin/bash
# first cleanup everything
iptables -t filter -F
iptables -t filter -X
iptables -t nat -F
iptables -t nat -X
# default drop
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT DROP
# allow loopback device
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT
# allow ssh over eth0 from outside to system
iptables -A INPUT -i eth0 -p tcp --dport 22 -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --sport 22 -j ACCEPT
# allow any traffic from 10.1.1.0/24 to system
iptables -A INPUT -i eth1 -s 10.1.1.0/24 -p tcp -j ACCEPT
iptables -A OUTPUT -o eth1 -d 10.1.1.0/24 -p tcp -j ACCEPT
```


11.3.9. Allowing ICMP(ping)

When you enable iptables, you will get an 'Operation not permitted' message when trying to ping other hosts.

```
[root@RHEL5 ~# ping 192.168.187.130
```

```
PING 192.168.187.130 (192.168.187.130) 56(84) bytes of data.
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
```
The screenshot below shows you how to setup iptables to allow a ping from or to your machine.
```
[root@RHEL5 ~]# iptables -A INPUT -p icmp --icmp-type any -j ACCEPT
[root@RHEL5 ~]# iptables -A OUTPUT -p icmp --icmp-type any -j ACCEPT
```
The previous two lines do not allow other computers to route ping messages through your router, because it only handles INPUT and OUTPUT. For routing of ping, you will need to enable it on the FORWARD chain. The following command enables routing of icmp messages between networks.
```
[root@RHEL5 ~]# iptables -A FORWARD -p icmp --icmp-type any -j ACCEPT
```


11.4. practice: packet filtering

1. Make sure you can ssh to your router-system when iptables is active.
2. Make sure you can ping to your router-system when iptables is active.
3. Define one of your networks as 'internal' and the other as 'external'. Configure the router to allow visits to a website (http) to go from the internal network to the external network (but not in the other direction).
4. Make sure the internal network can ssh to the external, but not the other way around.

11.5. solution: packet filtering

A possible solution, where leftnet is the internal and rightnet is the external network.
```
#!/bin/bash
# first cleanup everything
iptables -t filter -F
iptables -t filter -X
iptables -t nat -F
iptables -t nat -X
# default drop
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT DROP
# allow loopback device
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT
# question 1: allow ssh over eth0
iptables -A INPUT -i eth0 -p tcp --dport 22 -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --sport 22 -j ACCEPT
# question 2: Allow icmp(ping) anywhere
iptables -A INPUT -p icmp --icmp-type any -j ACCEPT
iptables -A FORWARD -p icmp --icmp-type any -j ACCEPT
iptables -A OUTPUT -p icmp --icmp-type any -j ACCEPT
# question 3: allow http from internal(leftnet) to external(rightnet)
iptables -A FORWARD -i eth1 -o eth2 -p tcp --dport 80 -j ACCEPT
iptables -A FORWARD -i eth2 -o eth1 -p tcp --sport 80 -j ACCEPT
# question 4: allow ssh from internal(leftnet) to external(rightnet)
iptables -A FORWARD -i eth1 -o eth2 -p tcp --dport 22 -j ACCEPT
iptables -A FORWARD -i eth2 -o eth1 -p tcp --sport 22 -j ACCEPT
# allow http from external(rightnet) to internal(leftnet)
# iptables -A FORWARD -i eth2 -o eth1 -p tcp --dport 80 -j ACCEPT
# iptables -A FORWARD -i eth1 -o eth2 -p tcp --sport 80 -j ACCEPT
# allow rpcinfo over eth0 from outside to system
# iptables -A INPUT -i eth2 -p tcp --dport 111 -j ACCEPT
# iptables -A OUTPUT -o eth2 -p tcp --sport 111 -j ACCEPT
```


11.6. network address translation

11.6.1. about NAT

A NAT device is a router that is also changing the source and/or target ip-address in packets. It is typically used to connect multiple computers in a private address range with the (public)

internet. A NAT can hide private addresses from the internet.
NAT was developed to mitigate the use of real ip addresses, to allow private address ranges to reach the internet and back, and to not disclose details about internal networks to the outside.
The nat table in iptables adds two new chains. PREROUTING allows altering of packets before they reach the INPUT chain. POSTROUTING allows altering packets after they exit the OUTPUT chain.
Use iptables -t nat -nvL to look at the NAT table. The screenshot below shows an empty NAT table.

```
[root@RHEL5 ~]# iptables -t nat -nL
Chain PREROUTING (policy ACCEPT)
target prot opt source destination
Chain POSTROUTING (policy ACCEPT)
target prot opt source destination
Chain OUTPUT (policy ACCEPT)
target prot opt source destination
[root@RHEL5 ~]#
```


11.6.2. SNAT (Source NAT)

The goal of source nat is to change the source address inside a packet before it leaves the system (e.g. to the internet). The destination will return the packet to the NAT-device. This means our NAT-device will need to keep a table in memory of all the packets it changed, so it can deliver the packet to the original source (e.g. in the private network).
Because SNAT is about packets leaving the system, it uses the POSTROUTING chain.
Here is an example SNAT rule. The rule says that packets coming from 10.1.1.0/24 network and exiting via eth1 will get the source ip-address set to 11.12.13.14. (Note that this is a one line command!)

```
iptables -t nat -A POSTROUTING -o eth1 -s 10.1.1.0/24 -j SNAT \
--to-source 11.12.13.14
```

Of course there must exist a proper iptables filter setup to allow the packet to traverse from one network to the other.

11.6.3. SNAT example setup

This example script uses a typical nat setup. The internal (eth0) network has access via SNAT to external (eth1) webservers (port 80).

```
#!/bin/bash
#
# iptables script for simple classic nat websurfing
# eth0 is internal network, eth1 is internet
#
echo 0 > /proc/sys/net/ipv4/ip_forward
iptables -P INPUT ACCEPT
iptables -P OUTPUT ACCEPT
iptables -P FORWARD DROP
iptables -A FORWARD -i eth0 -o eth1 -s 10.1.1.0/24 -p tcp \
--dport 80 -j ACCEPT
iptables -A FORWARD -i eth1 -o eth0 -d 10.1.1.0/24 -p tcp \
--sport 80 -j ACCEPT
iptables -t nat -A POSTROUTING -o eth1 -s 10.1.1.0/24 -j SNAT \
--to-source 11.12.13.14
echo 1 > /proc/sys/net/ipv4/ip_forward
```


11.6.4. IP masquerading

IP masquerading is very similar to SNAT, but is meant for dynamic interfaces. Typical example are broadband 'router/modems' connected to the internet and receiving a different ip-address from the isp, each time they are cold-booted.
The only change needed to convert the SNAT script to a masquerading is one line.

```
iptables -t nat -A POSTROUTING -o eth1 -s 10.1.1.0/24 -j MASQUERADE
```

11.6.5. DNAT (Destination NAT)

DNAT is typically used to allow packets from the internet to be redirected to an internal

server (in your DMZ) and in a private address range that is inaccessible directly form the internet.
This example script allows internet users to reach your internal (192.168.1.99) server via ssh (port 22).

```
#!/bin/bash
#
# iptables script for DNAT
# eth0 is internal network, eth1 is internet
#
echo 0 > /proc/sys/net/ipv4/ip_forward
iptables -P INPUT ACCEPT
iptables -P OUTPUT ACCEPT
iptables -P FORWARD DROP
iptables -A FORWARD -i eth0 -o eth1 -s 10.1.1.0/24 -j ACCEPT
iptables -A FORWARD -i eth1 -o eth0 -p tcp --dport 22 -j ACCEPT
iptables -t nat -A PREROUTING -i eth1 -p tcp --dport 22 \
-j DNAT --to-destination 10.1.1.99
echo 1 > /proc/sys/net/ipv4/ip_forward
```