Introduction to Netfilter/Iptables

Routing in Linux

Routing is the process of transmitting data packets between different networks. Routers can be hardware devices or just regular computers using the proper software. Linux can be a perfect solution to our routing needs.

In our scenario, Network Address Traslation (NAT) is the easiest way to give access to the Internet to the hosts in the internal network: with just one public IP, all the computers will be able to reach the Internet.

Our router should have at least two network cards, one connected to the LAN and the other to the Internet.

The computers in our LAN will use as gateway the internal IP address of the router. Linux will receive, process and resend the packets to the Internet.

A very convenient service for a router is a firewall, which allows filtering inbound and outbound data packets to protect efficiently our network. Netilfter/Iptables, the packet filtering framework inside the Linux 2.4.x and 2.6.x kernel series, enables packet filtering and network address translation (NAT).

Another interesting possibility could be the installation of a proxy server like Squid. It is a caching proxy for the Web supporting HTTP, HTTPS, FTP, and more. It reduces the used bandwidth and improves response times by caching and reusing frequently-requested web pages. Squid also has extensive access controls to regulate the use of the Internet in different ways:

- blocking certain websites.
- blocking certain internal hosts.
- allowing the connection only during certain hours.

Activating routing in Linux

If we want to activate routing in the Linux kernel, we just have to set to '1' the system variable 'ip_forward'. From a terminal, we have to run as root:

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

From that moment on, and without having to restart any service, the host will start sending through a network card everything that arrives to the other one and whose final destination is not the host itself, and vice versa.

But that is not enough, for example, to send packets from our LAN to the Internet. The problem is the source address of those packets, 192.168.x.x (or other address from the private ranges), which is not valid on the Internet.

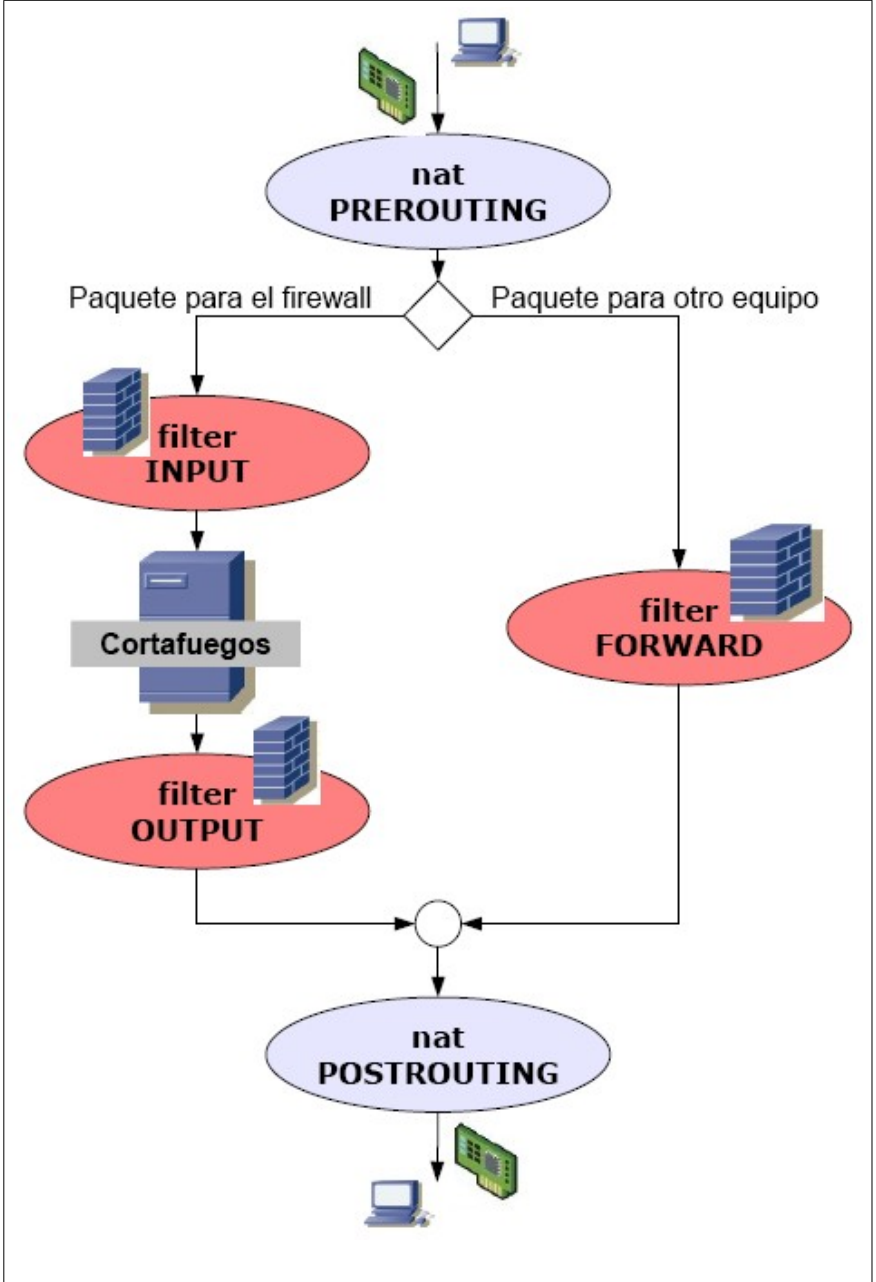
Netfilter/Iptables

Netfilter is the packet filtering framework inside the Linux 2.4 and 2.6 kernel series. **Iptables** is the command line program used to configure the Linux packet filtering rules.

Our goal for this section is to build a script to convert the Linux box into a gateway router for our LAN. It will control the network traffic, allowing certain packets and blocking the rest.

Netfilter/Iptables has many possibilities, but we are going to focus on two of them: 'packet filtering' and 'address translation'. Iptables has two **tables** (subprograms) to perform those tasks: the 'filter' table and the 'nat' table. Those tables become active in different moments or act over packets with differente destination, as shown in the following figure:

Netfilter/Iptables - Tables and chains



Simplified flow of possible routes for packets reaching the firewall

In the figure we can see five **chains** (**cadenas**).

The address translation or 'nat' can be done in two moments:

- as soon as the packet reachs the firewall and before deciding if the destination is the firewall itself or has to be forwarded. This moment (chain) is called **PREROUTING**. 'PREROUTING nat' is used in transparent proxies or when we want a computer in our LAN (behind the firewall) to be reachable from the Internet.
- just before the packet leaves the firewall, which is called **POSTROUTING** chain. This is the most common scenario, the address translation that allows all the stations in a LAN to surf the Internet sharing a public IP address is done here, in the postrouting chain.

The 'filter' table can be used with all the packets, but in different chains depending on the destination of the packet:

- packets intended for the firewall can be filtered on the **INPUT** chain. In this way, we can control the traffic that reaches the firewall. For example, if the only service our firewall is offering is the SSH server, we'll allow connections to port 22 and reject everything else.
- packets leaving the firewall can be filtered on the **OUTPUT** chain.
- packets getting to the firewall but having other hosts as final destination, this is, packets to be forwarded, can be filtered on the **FORWARD** chain.

Once we understand the previous paragraphs, we can start building the script that will allow our LAN to get to the Internet while at the same time preventing unwanted access from the outside. The script will contain a list of Iptables rules.

Erasing rules

When we add new rules to the firewall using the command 'iptables', these don't override the previous ones, they accumulate. That is why the first lines in all Iptables scripts are used to erase any rule that might exist:

```
iptables -t filter -F
iptables -t nat -F
```

In the first line, we are telling iptables to flush (-F stands for flush) the rules in the 'filter' table. The second one will erase completely the 'nat' table.
It is most common to find the previous lines in this slightly different format:

```
iptables -F
iptables -t nat -F
```

But the result is the same, because 'filter' is used when the table is not specified.

Default policy

The next step involves taking a decision about the default strategy:

- are we going to allow all traffic by default and, after it, explicitly reject the unwanted packets?

or, on the contrary,

- are we going to stop everything by default, allowing only what we want?

We'll use the second strategy. Although it is a bit more difficult to implement, it's more secure, there is less risk of leaving "holes" in our system.

In order to completely seal our system by default, we should include in the script (after the lines that erase all the previous rules) the following lines:

```
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP
```

The explanation is quite intuitive: the default policy (-P) is established for each chain (INPUT, OUTPUT and FORWARD) as DROP, this is, all the packets arriving at any of the chains are completely ignored... unless otherwise is stated in subsequent rules.

As it is, we could already execute the script and it would lead to a complete isolation of the machine, which wouldn't be able to neither send nor receive any packet. We should be aware that executing such a script remotely would cut the communication and we wouldn't be able to remotely reach the machine until the SSH port is open again... locally!

Practical - Place the above 5 rules in a script called 'ipt_todocerrado.sh', run it in your firewall and check that now it cannot neither reach the network nor be accessed from it. Apart from the rules themselves, it is very convenient to add comments before each rule, explaining what the rule does in plain English.

Practical - Copy the script as 'ipt_todoabierto.sh' and replace the three DROP with ACCEPT. Execute and check that the firewall can again access the network.

The lines in 'ipt_todocerrado.sh' will be the first ones in all our scripts from now on. Every port will start closed and we'll open the services that have to be allowed.

The last script, 'ipt_todoabierto.sh', can be useful to get Ubuntu back to the original state regarding the firewall behaviour.

Forwarding packets

Now, we'll add several rules to the script 'ipt_bloqueo.sh'. It goes without saying that the rules in the scripts are executed sequentially.
If we want the firewall to forward all the packets not intended for itself, we have to add the following line to our script:

```
iptables -t filter -A FORWARD -j ACCEPT
```

or, shorter but with the same effect:

```
iptables -A FORWARD -j ACCEPT
```

In our language this means: "add a rule (-A) to the 'FORWARD' chain of the 'filter' table so it accepts (forwards) every packet (-j ACCEPT)".

But, as said before, the source address of those packets injected on the Internet with that rule won't be valid, so we have to employ NAT in this way:

```
iptables -t nat -A POSTROUTING -j MASQUERADE
```

This rule means: "add a rule (-A) to 'POSTROUTING' chain in the 'nat' table to replace the original source address of the packets with the public address of the router.

And to finish our first script, don't forget to activate packet forwarding:

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

Practical - Place all the previous commands in a script called 'ipt_rut1.sh', run it, and check it can work as a gateway for all the other computers, this is, check that the other hosts can reach the Internet.

Refining the rules

Some of the abover rules can be more specific, following the known policy of not allowing more than strictly necessary.

For example, there is no need for NAT in all the network adapters. Since it is only useful in the external network card, and supposing it is 'eth0', the MASQUERADE rule could be rewritten as:

```
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

Practical - Copy 'ipt_rut1.sh' script as 'ipt_rut2.sh', modify the previous line and check it still works for us.

Another rule that can be improved is the one forwarding everything 'iptables -t filter -A FORWARD -j ACCEPT'. If that was what we really wanted, it would be enough if we used ACCEPT instead of DROP in the policy. But we don't need everything to be forwarded, only what comes through the internal ethernet card, 'eth1' in our example. So, the rule would be:

```
iptables -t filter -A FORWARD -i eth1 -j ACCEPT
```

However, it doesn't work properly now. Our reasoning seemed flawless, but we have missed something: we allow forwarding packets from our network to the Internet, but we don't accept the answers coming back to our LAN. How to achieve it? Using the "opposite" rule: 'iptables -t filter -A FORWARD -i eth0 -j ACCEPT'? No, because this last rule and the previous one are equivalent to the one we were trying to refine. The solution is the following rule:

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

It is interpreted like this: "we are accepting the packets matching (**-m**, match, coincidir) the state (`--state`) 'ESTABLISHED,RELATED', this is, packets belonging to either already established connections or related connections.

With this arrangement:

- the stations in our LAN can start connections with computers on the Internet and the answers (ESTABLISHED,RELATED) arrive to our computers.
- hosts on the Internet won't be able to start connections to our computers and, thanks to it, many security problems are avoided.

Practical - Copy 'ipt_rut2.sh' as 'ipt_rut3.sh', made the discussed improvements and check if it works or not.

Allowing only certain services

Using 'ipt_rut3.sh', the hosts in our LAN have completely open access to the Internet, i.e., they can use any port and, therefore, any service. But that is neither necessary nor convenient, so we will start allowing web browsing for everyone, but nothing else. To achieve it, we will only forward requests to port 80. That involves modifying the line 'iptables -t filter -A FORWARD -i eth1 -j

ACCEPT', making it more restrictive:

```
iptables -t filter -A FORWARD -i eth1 -p TCP --dport 80 -j ACCEPT
```

We have included two more conditions, the protocol has to be TCP (-p TCP) and the destination port has to be 80 (dport 80).

Practical - Copy 'ipt_rut3.sh' as 'ipt_rut80.sh', modify the line and check if from one of the client computers you can reach http://www.google.com and http://10.22.23.61 (our internal http server). Can you explain it? How can the new problem be solved?

Practical - Copy 'ipt_rut80.sh' as 'ipt_rut80-pop3.sh' and add the rules needed to allow the users to receive mail using 'pop3' protocol.

Practical - Check that you cannot access 10.22.23.61 via SSH. Modify the script to make it possible.

Pings

Practical - Check that, after running any of the previous scripts allowing only certain services, pings are stopped at the firewall.

This is hardly surprising, because we have closed everything by default and opened certain services, but ICMP messages are not part of the unblocked packets. In order to allow pings, we should add a line like this:

```
iptables -t filter -A FORWARD -i eth1 -p ICMP --icmp-type 8 -j ACCEPT
```

Thanks to it, our stations are allowed to ping external hosts. These ones can answer because the line `iptables -t filter -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT` is still part of the script.

Practical - Save 'ipt_rut80.sh' as 'ipt_rut80p.sh', include the aforementioned line and check it works.

Rules involving the router

All the previous rules dealt with the FORWARD chain, this is, controlled the forwarding of packets. We haven't touched INPUT and OUTPUT chains but to establish the default policy for them to DROP. In other words, our firewall is now completely isolated from the LAN. That can be easily verified, because the firewall box cannot send pings, doesn't answer to them, cannot use FTP...

Let's then work with the INPUT and OUTPUT chains. INPUT affects the packets whose destination is the firewall and OUTPUT is for the outgoing packets generated in the firewall.

Localhost

We are going to allow 'localhost' to create and receive connections. Copy 'ipt_rut80p.sh' as 'ipt_rut80p-lo.sh' and add the following lines preceded by an explanatory coment:

```
iptables -t filter -A INPUT -i lo -j ACCEPT
iptables -t filter -A OUTPUT -o lo -j ACCEPT
```

Thanks to these lines, the 'loopback' adapter, used for the connections to a service in the host from the host itself, works again.

Practical - Run the script 'ipt_rut80p-lo.sh' and verify that 'http://localhost' is working again.

Pings

It is common practice to allow hosts answer to pings, so we are going to add to the previous script the following lines:

```
iptables -t filter -A INPUT -p ICMP --icmp-type 8 -j ACCEPT
iptables -t filter -A OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

In the first one, echo requests (pings) are accepted. In the second one, the answers to established or related connections are permitted to leave the firewall.

Practical - Add the above lines (with comments) to 'ipt_rut80p-lo.sh' and verify that the router answers pings back. Can the firewall send echo requests?

Apache and FTP servers in the firewall

In the previous unit Apache and FTP were installed in the computer now acting as a firewall, so we have to allow the other hosts in the LAN to access those services.

Practical - Copy 'ipt_rut80p-lo.sh' as 'ipt_rut80p-lop80FTP.sh' and add the lines that make possible for the LAN to access the Web and FTP servers in the firewall.

Accessing the network from the router/firewall

In a previous practical we have seen that the firewall cannot even ping other machines. To change that, we can either compeltely trust the firewall and add a rule to accept everything reaching the OUTPUT chain (which is equivalent to establishing the defaul policy to 'accept' with 'iptables -P OUTPUT ACCEPT') or follow the path of the previous scripts and accept only what is strictly necessary.

Practical - Add to the previous script the rules needed for the firewall to be able to send pings.

Final note on Netfilter/Iptables

We should note that we have discussed Netfilter/Iptables in a very simplified way. As it occurs with other units in our course, a production implementation of Iptables involves examining and using additional material.

Maybe the best tutorial about Iptables is mantained by Oskar Andreasson on the address http://www.frozentux.net/documents/iptables-tutorial/