Reducing iptables configuration complexity using chains

Dieter Adriaenssens
Ghent University, Belgium

LinuxTag - Berlin
May 8th, 2014

1 Abstract

After a short introduction of iptables, a local firewall for Linux systems, we go more into detail what chains are, and how these can be used to reduce the complexity of an iptables configuration by grouping similar filtering rules. This results in a better structured configuration that is easier to maintain and that runs faster.

The audience should have a basic understanding of network concepts like protocols, ports, IP-addresses and packets.

Contents

2 Introduction

Setting up a firewall on your *nix box, being it a workstation, laptop, or server, is always a good idea. In most cases, you can do with some simple firewall rules, f.e. on your laptop, block all incoming requests (except the established connections, i.e. the replies on the outgoing requests you made), or on a simple webserver (allow port 80 only).

But if you need more complex rules, f.e. a server that hosts a website available for the entire internet, but with an SSH and samba service that should only be available for the local subnet, or even some specific IP addresses, it becomes a bit more complex.
And if you want to filter the outgoing traffic as well, your iptables rules get a mess after a while, and when you want to change anything, chances of a mistake or forgetting something are high, which may result in locking yourself out of your box (at least for remote access), or leaving something open that shouldn't.

To make your rules more manageable, you can make use of chains in your iptables rules. This paper is inspired by an article that uses chains to make iptables more efficient (faster). The goal of this paper is to get iptables rules that are easier to read and configure, but it will result in faster handling of packets as well.

3 Brief introduction to iptables

Iptables is a tool for creating the rulesets for netfilter, a packet filtering framework which was introduced in the linux 2.4 kernel.

Netfilter is rule based. When an IP packet arrives, it is checked against a set of rules, fe. :

```
iptables -A INPUT -m tcp -p tcp \dd dport ssh -j ACCEPT
iptables -A INPUT -m tcp -p tcp \dd dport http -j ACCEPT
iptables -A INPUT -m tcp -p tcp \dd dport https -j ACCEPT
iptables -A INPUT -j DROP
```

Every rule consists of several parts :

- the chain the rule belongs to

- parameters filtering for a type of packet
- the target : action to be taken when a packet matches

3.1 Default chains

There are several predefined chains :

- INPUT
- OUTPUT
- FORWARD
- PREROUTING
- POSTROUTING

Defining custom chains is also possible.

In this example, the rule is added to the INPUT chain :

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

3.2 Filters

Filter on packet properties :

- protocol (tcp, udp, icmp, ...)
- destination/source port
- destination/source IP address
- in/outgoing interface (eth0, ...)
- ...

In this example, the filter matches any TCP packet with destination port 22 (SSH) originating from the 10.0.0.0/8 network :

```
iptables -A INPUT -m tcp -p tcp --dport ssh -s 10.0.0.0/8 -j ACCEPT
```

3.3 Targets

What happens if a packet matches a rule :

- ACCEPT
- DROP
- QUEUE → userspace
- RETURN → leave current chain
- LOG
- jump to a custom chain

In this example, the packet is accepted if it matches the rule :

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

3.4 Putting it all together

All rules are checked one for one, and if one matches the target is executed, in this case, the packet is accepted :

```
iptables -A INPUT -m tcp -p tcp --dport ssh -j ACCEPT
iptables -A INPUT -m tcp -p tcp --dport 80 -j ACCEPT
iptables -A INPUT -m tcp -p tcp --dport https -j ACCEPT
iptables -A INPUT -j DROP
```

The last rule matches everything, so if a packet didn't match a previous rule, it will be rejected.
Remark: the order of the rules is important.

3.5 More about iptables

This was just a brief introduction to iptables/netfilter. If you want to know more about iptables configuration, you can have a look at this excellent and detailed tutorial by Oskar Andreasson : http://www.frozentux.net/iptables-tutorial/iptables-tutorial.html

4 Optimizing configuration

4.1 Example setup definition

- A web service should be available from all networks (i.e. internet) on port 80 (http) and 443 (https)
- The server can be managed remotely using SSH (port 22) and a web-based admin tool (port 10000), but only from a limited set of IP addresses (admin PC's).
- The server hosts a samba service (several TCP ports), that should only be available from a limited set of IP addresses (admin + webmaster PC's).
- Outgoing connections will be filtered, but some services should be allowed (DNS, DHCP, SMTP, NTP) and some external websites should be available to get updates.

4.2 ESTABLISHED state

When using this option, you can filter for established connections. If you define it in both the INPUT and OUTPUT rules, you only have to define in the INPUT rules which NEW incoming requests should be allowed, and in the OUTPUT rules which NEW outgoing request are allowed. The established connections will be allowed and should not be redefined (making the configuration a lot more readable and maintainable). An example allowing only an SSH service without using the ESTABLISHED state would be :

```
iptables -A INPUT -p tcp --dport ssh -j ACCEPT
iptables -A INPUT -j REJECT
iptables -A OUTPUT -p tcp --sport ssh -j ACCEPT
iptables -A OUTPUT -j REJECT
```

Basically, every incoming/outgoing connection is dropped, except if the incoming packet has port 22 (SSH) as destination, or if the outgoing packet was sent from port 22 (which is the reply of the SSH server).

When using ESTABLISHED state, this will become :

```
iptables -A INPUT -p tcp --dport ssh -j ACCEPT
iptables -A INPUT -j REJECT
iptables -A OUTPUT -m state --state ESTABLISHED -j ACCEPT
iptables -A OUTPUT -j REJECT
```

Now, every incoming/outgoing connection is dropped, except if the incoming packet has port 22 (SSH) as destination, or if the packet belongs to an established connection. Because incoming connections to port 22 are allowed, the firewall will remember a packet coming in, creating a 'connection' for the host/port the packet originates from when the SSH server replies to it. So when the reply of the SSH server is sent out, it matches an 'established' connection and will be allowed out.

In this example, the benefit of using the connection state is not clear, but when more allowed incoming services are added, they only have to be added on the INPUT chain, but not on the OUTPUT chain, because they are covered by the ESTABLISHED rule.
In the first example (without the ESTABLISHED rule), every allowed incoming connection should be repeated in the OUTPUT chain, matching the packets sent for the outgoing connection, which results in an equal amount of rules on both chains.
If you want to do filtering in both directions (allowing incoming request for listening services and outgoing request for remote services), this can become very messy, and almost unmaintainable without making mistakes.

4.3 Introducing custom chains

When two services (on different ports) should be available to a limited but identical list of IP addresses. Without using chains, for every combination of port and IP a rule should be created :

```
iptables -A INPUT -p tcp -m tcp -s 10.100.2.3 --dport 22 -j ACCEPT
iptables -A INPUT -p tcp -m tcp -s 10.100.2.4 --dport 22 -j ACCEPT
iptables -A INPUT -p tcp -m tcp -s 10.100.2.7 --dport 22 -j ACCEPT

iptables -A INPUT -p tcp -m tcp -s 10.100.2.3 --dport 10000 -j ACCEPT
iptables -A INPUT -p tcp -m tcp -s 10.100.2.4 --dport 10000 -j ACCEPT
iptables -A INPUT -p tcp -m tcp -s 10.100.2.7 --dport 10000 -j ACCEPT
```

Resulting in a lot of rules, and when an IP address has to be changed, added or removed, this has to be done for every corresponding rule.

When using custom chains, this can be much easier. Imagine, that you first check if the packet matches the destination port, and if it does, jump to a new chain, where a list of IP addresses is checked. :

```
// create new custom chain admin_IP
iptables -N admin_IP

// add rules to custom chain admin_IP
iptables -A admin_IP -s 10.100.2.3 -j ACCEPT
iptables -A admin_IP -s 10.100.2.4 -j ACCEPT
iptables -A admin_IP -s 10.100.2.7 -j ACCEPT
// drop all packets that are not matched by previous rules
iptables -A admin_IP -j DROP

// filter ports in INPUT chain
iptables -A INPUT -p tcp -m tcp --dport 22 -j admin_IP
iptables -A INPUT -p tcp -m tcp --dport 10000 -j admin_IP
```

As you can see, there is are several benefits of putting the IP addresses in a separate chain :

- the list of IP addresses in the separate chain can be reused for both ports, so they have to be defined only once.
- adding/changing/removing an IP address is much easier
- there is a better overview of the firewall rules

4.4 Actual configuration

4.4.1 INPUT chain

```
iptables -A INPUT -i lo -j ACCEPT
iptables -A INPUT -m state --state ESTABLISHED -j ACCEPT
iptables -A INPUT -m state --state RELATED -j ACCEPT
iptables -A INPUT -p icmp -j icmp_in
iptables -A INPUT -p tcp -m tcp --dport 22 -j admin_IP
iptables -A INPUT -p tcp -m tcp --dport 10000 -j admin_IP
iptables -A INPUT -p tcp -m tcp --dport 80 -j ACCEPT
iptables -A INPUT -p tcp -m tcp --dport 443 -j ACCEPT
iptables -A INPUT -p tcp -m tcp --dport 139 -j webmaster_IP
iptables -A INPUT -p tcp -m tcp --dport 445 -j webmaster_IP
iptables -A INPUT -j DROP
```

This is the INPUT chain, allowing :

- all local traffic (not leaving the physical PC)

- established and related connections
- ICMP packets (ping, etc.) are handled in a seperate chain icmp_in
- some services
  - SSH (tcp 22) and webmin (tcp 10000) allowed for admins (admin_IP chain)
  - website (tcp 80 and 443) for everyone
  - SMB (tcp 139 and 445) for webmasters (and admins, see definition of webmaster_IP chain)
- everything else is not allowed (dropped)

4.4.2 OUTPUT chain

```
iptables -A OUTPUT -o lo -j ACCEPT
iptables -A OUTPUT -m state --state ESTABLISHED -j ACCEPT
iptables -A OUTPUT -m state --state RELATED -j ACCEPT
iptables -A OUTPUT -p icmp -m icmp --icmp-type 8 -j ACCEPT
iptables -A OUTPUT -p udp -m udp --dport 53 -j ACCEPT
iptables -A OUTPUT -p udp -m udp --dport 67 -j ACCEPT
iptables -A OUTPUT -p tcp -m tcp --dport 25 -j ACCEPT
iptables -A OUTPUT -p udp -m udp --dport 123 -j ACCEPT
iptables -A OUTPUT -p tcp -m tcp --dport 80 -j ext_websites
iptables -A OUTPUT -j DROP
```

The OUTPUT chain, allowing :

- all local traffic (not leaving the physical PC)
- established and related connections
- ICMP replies (fe. ping)
- some remote services :
  - DNS (udp 53) : name service
  - DHCP (udp 67) : dynamic IP address service
  - SMTP (tcp 25) : mail service
  - NTP (udp 123) : time service
- external websites (tcp 80), listed in chain ext_websites
- everything else is not allowed (dropped)

4.4.3 ICMP chain

```
iptables -N icmp_in
iptables -A icmp_in -p icmp -m icmp --icmp-type 8 -j ACCEPT
iptables -A icmp_in -p icmp -m icmp --icmp-type 0 -j ACCEPT
iptables -A icmp_in -p icmp -m icmp --icmp-type 3 -j ACCEPT
iptables -A icmp_in -p icmp -m icmp --icmp-type 4 -j ACCEPT
iptables -A icmp_in -p icmp -m icmp --icmp-type 11 -j ACCEPT
iptables -A icmp_in -p icmp -m icmp --icmp-type 12 -j ACCEPT
iptables -A icmp_in -j DROP
```

All allowed incoming ICMP message types.

4.4.4 admin chain

```
iptables -N admin_IP
iptables -A admin_IP -s 10.100.2.3 -j ACCEPT
iptables -A admin_IP -s 10.100.2.4 -j ACCEPT
iptables -A admin_IP -s 10.100.2.7 -j ACCEPT
iptables -A admin_IP -j DROP
```

A list of allowed IP addresses of admin PC's. Everything else is not allowed.

4.4.5 webmaster chain

```
iptables -N webmaster_IP
iptables -A webmaster_IP -s 10.100.2.11 -j ACCEPT
iptables -A webmaster_IP -s 10.100.2.17 -j ACCEPT
iptables -A webmaster_IP -s 10.100.2.34 -j ACCEPT
iptables -A webmaster_IP -s 10.100.2.50 -j ACCEPT
iptables -A webmaster_IP -j admin_IP
```

A list of allowed IP addresses of webmaster PC's. At the end of the list, it jumps to the admin_IP chain, chaining both chains.

4.4.6 External websites chain

```
iptables -N ext_websites
iptables -A ext_websites -d 212.211.132.250 -j ACCEPT
iptables -A ext_websites -d 212.211.132.32 -j ACCEPT
iptables -A ext_websites -d 195.20.242.89 -j ACCEPT
iptables -A ext_websites -d 130.89.149.225 -j ACCEPT
```

```
iptables -A ext_websites -d 86.59.118.153 -j ACCEPT
iptables -A ext_websites -d 130.89.149.227 -j ACCEPT
iptables -A ext_websites -d 128.31.0.51 -j ACCEPT
iptables -A ext_websites -d 86.59.118.153 -j ACCEPT
iptables -A ext_websites -d 67.228.198.100 -j ACCEPT
iptables -A ext_websites -d 140.211.166.6 -j ACCEPT
iptables -A ext_websites -d 140.211.166.21 -j ACCEPT
iptables -A ext_websites -j LOG
```

A list of allowed external websites for updates (mirrors of Debian, webmin and Drupal, in this example). All other requests for external websites are logged. This can be useful for monitoring : notification of abuse, or if you forgot to add an allowed website.

5 Conclusion

- When checking for the ESTABLISHED state, the number of rules can be reduced, because only the initiating packet needs to be checked
- Using chains makes your rules better structured which makes them easier to read and better maintainable
- Chains can be reused for several rules
- Chains can be chained together
- Using chains makes checking rules faster, because it only jumps to a chain when a rule matches. Not all rules have to be checked

6 About the author

Dieter Adriaenssens currently works at Ghent University and is a sysadmin since 2005. He is an active Open Source community member, who contributed to phpMyAdmin and other projects, regularly attending and speaking at conferences, like FOSDEM.
Recently, Dieter started exploring the world of Android application development, which resulted in a first release of a navigation app a few months ago. He lives in Ghent, Belgium and enjoys rock climbing.

- Blog : http://ruleant.blogspot.com
- Twitter : @dcadriaenssens