# Reducing iptables configuration complexity using chains

Dieter Adriaenssens

@dcadriaenssens - Ghent University, Belgium

LinuxTag - Berlin
May 8th, 2014

# Overview

- Brief introduction to iptables/netfilter
- Optimizing configuration with a tutorial use case
- Conclusion

# What is iptables/netfilter?

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

# Rules

When an IP packet comes in, it is checked against a set of rules.

## Example

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

# Chains

Predefined chains :

- INPUT
- OUTPUT
- FORWARD
- PREROUTING
- POSTROUTING

## Example

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

Custom chains can be defined

# Match

Filter on packet parameters :

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

## Example

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

# Targets

What to do if a packet matches a rule :

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

## Example

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

# Putting it all together

All rules are checked one by one, and if one matches, the target is executed :

### Example

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

The last rule matches everything, so if a packet didn't match a previous rule, it will be rejected.

# Putting it all together

All rules are checked one by one, and if one matches, the target is executed :

### Example

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

# Default target DROP

A default target can be set for the predefined chains:

### Example

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

~~-A INPUT -j DROP~~

If none of the rules match, the default target is executed.

In this example, a packet is dropped.

# Default target ACCEPT

**Exercise:**

What happens if the default target is ACCEPT?

## Example

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

# Default target ACCEPT

**Exercise:**
What happens if the default target is ACCEPT?

## Example

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

All packets are accepted, even if no rule matches them.

# Default target ACCEPT

**Exercise:**
What happens if the default target is ACCEPT?

### Example

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

All packets are accepted, even if no rule matches them.
**Always define a default target or use an all-matching rule at the end!**

# Example : firewall for your server

Filter both incoming and outgoing traffic :

### Example

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

# Example : firewall for your server

Filter both incoming and outgoing traffic :

### Example

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

**Problem:** incoming packets are accepted, but replies are dropped!

# Example : firewall for your server

Also add rules for outgoing traffic :

## Example

```
-P INPUT DROP
-P OUTPUT DROP
-A INPUT -m tcp -p tcp --dport ssh -j ACCEPT
-A INPUT -m tcp -p tcp --dport 80 -j ACCEPT
-A INPUT -m tcp -p tcp --dport https -j ACCEPT
-A OUTPUT -m tcp -p tcp --sport ssh -j ACCEPT
-A OUTPUT -m tcp -p tcp --sport 80 -j ACCEPT
-A OUTPUT -m tcp -p tcp --sport https -j ACCEPT
```

# Example : firewall for your server

Also add rules for outgoing traffic :

### Example

```
-P INPUT DROP
-P OUTPUT DROP
-A INPUT -m tcp -p tcp --dport ssh -j ACCEPT
-A INPUT -m tcp -p tcp --dport 80 -j ACCEPT
-A INPUT -m tcp -p tcp --dport https -j ACCEPT
-A OUTPUT -m tcp -p tcp --sport ssh -j ACCEPT
-A OUTPUT -m tcp -p tcp --sport 80 -j ACCEPT
-A OUTPUT -m tcp -p tcp --sport https -j ACCEPT
```

This can get complicated if more rules are added.

# Established state

**Solution:** check for established state

### Example

```
-P INPUT DROP
-P OUTPUT DROP
-A INPUT -m tcp -p tcp --dport ssh -j ACCEPT
-A INPUT -m tcp -p tcp --dport 80 -j ACCEPT
-A INPUT -m tcp -p tcp --dport https -j ACCEPT
-A OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

This makes the rules more manageable, especially when output rules are also defined.

# Intermezzo : firewall on your PC

Protect your own PC, block all incoming requests :

### Example

```
-P INPUT DROP
-P OUTPUT ACCEPT
```

# Intermezzo : firewall on your PC

Protect your own PC, block all incoming requests :

## Example

```
-P INPUT DROP
-P OUTPUT ACCEPT
-A INPUT -i lo -j ACCEPT # allow local traffic
-A INPUT -p icmp -j ACCEPT # allow ping, etc
-A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Add ESTABLISHED, to allow replies of outgoing connections you intiated!

# Tutorial use case : setup

- Tcp port 80 (http) and 443 (https) is available for all.
- SSH (port 22) and a webbased admin tool (port 10000), is limited to admin PCs.
- A SMB service is limited to admin and webmaster PCs.

# Tutorial use case : webserver

Webservice, tcp port 80 (http) and 443 (https) is available for all :

## Example

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

# Tutorial use case : sysadmin PCs

Sysadmins are allowed to access SSH (tcp port 22) and a webbased admin tool (tcp port 10000).
The IP addresses of their PCs :

- 10.100.2.3
- 10.100.2.4
- 10.100.2.7

# Tutorial use case : sysadmin PCs

## Example

```
-A INPUT -p tcp -m tcp -s 10.100.2.3 --dport 22 -j ACCEPT
-A INPUT -p tcp -m tcp -s 10.100.2.4 --dport 22 -j ACCEPT
-A INPUT -p tcp -m tcp -s 10.100.2.7 --dport 22 -j ACCEPT
-A INPUT -p tcp -m tcp -s 10.100.2.3 --dport 10000 -j ACCEPT
-A INPUT -p tcp -m tcp -s 10.100.2.4 --dport 10000 -j ACCEPT
-A INPUT -p tcp -m tcp -s 10.100.2.7 --dport 10000 -j ACCEPT
```

# Tutorial use case : sysadmin PCs

## Example

```
-A INPUT -p tcp -m tcp -s 10.100.2.3 --dport 22 -j ACCEPT
-A INPUT -p tcp -m tcp -s 10.100.2.4 --dport 22 -j ACCEPT
-A INPUT -p tcp -m tcp -s 10.100.2.7 --dport 22 -j ACCEPT
-A INPUT -p tcp -m tcp -s 10.100.2.3 --dport 10000 -j ACCEPT
-A INPUT -p tcp -m tcp -s 10.100.2.4 --dport 10000 -j ACCEPT
-A INPUT -p tcp -m tcp -s 10.100.2.7 --dport 10000 -j ACCEPT
```

The same IP addresses are repeated. If there is a change in the IP addresses list, several rules have to be updated.

# Tutorial use case : sysadmin PCs

## Example

```
-A INPUT -p tcp -m tcp -s 10.100.2.3 --dport 22 -j ACCEPT
-A INPUT -p tcp -m tcp -s 10.100.2.4 --dport 22 -j ACCEPT
-A INPUT -p tcp -m tcp -s 10.100.2.7 --dport 22 -j ACCEPT
-A INPUT -p tcp -m tcp -s 10.100.2.3 --dport 10000 -j ACCEPT
-A INPUT -p tcp -m tcp -s 10.100.2.4 --dport 10000 -j ACCEPT
-A INPUT -p tcp -m tcp -s 10.100.2.7 --dport 10000 -j ACCEPT
```

The same IP addresses are repeated. If there is a change in the IP addresses list, several rules have to be updated.
**This can be done more efficient!**

# Tutorial use case : sysadmin PCs

Wouldn't it be convenient to :

- make a list of IP addresses?
- check this list when a packet matches a rule (fe. TCP port 22)?
- reuse this list when another rule matches a packet?

# Tutorial use case : sysadmin PCs

Wouldn't it be convenient to :

- make a list of IP addresses?
- check this list when a packet matches a rule (fe. TCP port 22)?
- reuse this list when another rule matches a packet?

**This is possible with custom chains!**

# Introducing custom chains

Create the custom chain and add the rules

### Example

```
-N admin_IP
```

# Introducing custom chains

Create the custom chain and add the rules

## Example

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

# Introducing custom chains

Create the custom chain and add the rules

## Example

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

- Syntax of adding rules to a custom chain is similar to adding to default targets.

# Introducing custom chains

Create the custom chain and add the rules

### Example

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

- Syntax of adding rules to a custom chain is similar to adding to default targets.
- Custom chains don't have a default target, so set a target for all packets that are not matched.

# Why you should set a default target

Avoid jumping back to the calling chain :

- performance issue
- unexpected behaviour, some other rules might match

# Tutorial use case : sysadmin PCs

Add the rules for TCP port 22 and 10000 using the custom chain :

### Example

```
-A INPUT -p tcp -m tcp --dport 22 -j admin_IP
-A INPUT -p tcp -m tcp --dport 10000 -j admin_IP
```

# Benefits

- the list of IP addresses in the custom chain is reused for both ports, so they have to be defined only once
- adding/changing/removing an IP address is much easier
- a better overview of the firewall rules
- better performance : the rules in the custom chain are only checked if that chain is accessed

# Tutorial use case : webmaster PCs

Let's do the same for webmasters AND admins having access to SMB.

# Tutorial use case : webmaster PCs

Let's do the same for webmasters AND admins having access to SMB.

Create a chain with webmaster IP addresses :

### Example

```
-N webmaster_IP
-A webmaster_IP -s 10.100.2.11 -j ACCEPT
-A webmaster_IP -s 10.100.2.17 -j ACCEPT
-A webmaster_IP -s 10.100.2.34 -j ACCEPT
-A webmaster_IP -j DROP
```

# Tutorial use case : access to SMB

Add the rules for SMB (tcp port 139 and 445) using the custom chains :

## Example

```
-A INPUT -p tcp -m tcp --dport 445 -j admin_IP
-A INPUT -p tcp -m tcp --dport 445 -j webmaster_IP
-A INPUT -p tcp -m tcp --dport 139 -j admin_IP
-A INPUT -p tcp -m tcp --dport 139 -j webmaster_IP
```

# Tutorial use case : access to SMB

Add the rules for SMB (tcp port 139 and 445) using the custom chains :

## Example

```
-A INPUT -p tcp -m tcp --dport 445 -j admin_IP
-A INPUT -p tcp -m tcp --dport 445 -j webmaster_IP
-A INPUT -p tcp -m tcp --dport 139 -j admin_IP
-A INPUT -p tcp -m tcp --dport 139 -j webmaster_IP
```

**This can be optimized!**

# Chaining chains

Include the sysadmin chain in the webmaster chain:

### Example

```
-N webmaster_IP
-A webmaster_IP -s 10.100.2.11 -j ACCEPT
-A webmaster_IP -s 10.100.2.17 -j ACCEPT
-A webmaster_IP -s 10.100.2.34 -j ACCEPT
~~-A webmaster_IP -j DROP~~
-A webmaster_IP -j admin_IP
```

Jump to the sysadmin chain at the end of the webmaster chain.

**Warning :** check conditions of ending chains

# Tutorial use case : access to SMB

The rules for SMB (tcp port 139 and 445) can be reduced to :

## Example

~~-A INPUT -p tcp -m tcp --dport 445 -j admin_IP~~
-A INPUT -p tcp -m tcp --dport 445 -j webmaster_IP
~~-A INPUT -p tcp -m tcp --dport 139 -j admin_IP~~
-A INPUT -p tcp -m tcp --dport 139 -j webmaster_IP

The admin chain is checked as well because the webmaster chain includes it.

# Conclusion

- use ESTABLISHED state to reduce number of rules
- using chains makes your rules easier readable and maintainable
- chains can be reused for several rules
- chains can be chained together
- faster, because it only jumps to a chain when a rule matches

# Questions

Thanks for your attention!
Questions?

Contact

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

# IPv6

Configuration is the same for IPv4, use *ip6tables* :

### Example

```
ip6tables -P INPUT DROP
ip6tables -P OUTPUT ACCEPT
ip6tables -A INPUT -i lo -j ACCEPT # allow local traffic
ip6tables -A INPUT -p icmp -j ACCEPT # allow ping, etc
ip6tables -A INPUT -m state --state ESTABLISHED -j ACCEPT
ip6tables -A INPUT -m state --state RELATED -j ACCEPT
```

Even if IPv4 rules are set, IPv6 has to be configured seperately, otherwise the defaults apply.