Firewalls

IPtables – Advanced Configuration

Miguel Frade & Francisco Santos

POLITÉCNICO DE LEIRIA
ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

Stateful Configuration

STATEFUL CONFIGURATION

Monitoring of the connections state with `-m state --state <state-name>`

Possible states:

- `NEW` – to match the first packet of a connection

STATEFUL CONFIGURATION

Monitoring of the connections state with `-m state --state <state-name>`

Possible states:

- `NEW` – to match the first packet of a connection
- `ESTABLISHED` – to match packets after the first one, in both directions, and will then continuously match those packets

STATEFUL CONFIGURATION

Monitoring of the connections state with `-m state --state <state-name>`

Possible states:

- `NEW` – to match the first packet of a connection
- `ESTABLISHED` – to match packets after the first one, in both directions, and will then continuously match those packets
- `RELATED` – to match packets related to another already `ESTABLISHED` connection, *e. g.* the FTP data connection

STATEFUL CONFIGURATION

Monitoring of the connections state with `-m state --state <state-name>`

Possible states:

- `NEW` – to match the first packet of a connection
- `ESTABLISHED` – to match packets after the first one, in both directions, and will then continuously match those packets
- `RELATED` – to match packets related to another already `ESTABLISHED` connection, *e. g.* the FTP data connection
- `INVALID` – to match packets that cannot be identified or that don't have any state

STATEFUL CONFIGURATION

Monitoring of the connections state with `-m state --state <state-name>`

Possible states:

- `NEW` – to match the first packet of a connection
- `ESTABLISHED` – to match packets after the first one, in both directions, and will then continuously match those packets
- `RELATED` – to match packets related to another already `ESTABLISHED` connection, *e. g.* the FTP data connection
- `INVALID` – to match packets that cannot be identified or that don't have any state
- `UNTRACKED` – ask to not track some packets, must be set explicitly in the raw table

STATEFUL CONFIGURATION

Monitoring of the connections state with `-m state --state <state-name>`

Possible states:

- `NEW` – to match the first packet of a connection
- `ESTABLISHED` – to match packets after the first one, in both directions, and will then continuously match those packets
- `RELATED` – to match packets related to another already `ESTABLISHED` connection, *e. g.* the FTP data connection
- `INVALID` – to match packets that cannot be identified or that don't have any state
- `UNTRACKED` – ask to not track some packets, must be set explicitly in the raw table
  - outside the scope of this course

Generic stateful rules

```
$IPT -A INPUT  -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPT -A OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Do not write all rules in `stateful` mode

- it has an impact in performance and there are situations where it's irrelevant

Generic stateful rules

```
$IPT -A INPUT  -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPT -A OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Do not write all rules in `stateful` mode

- it has an impact in performance and there are situations where it's irrelevant

Recommend script structure

```
# script initialization
...
# stateless rules
...
# generic stateful rules
# additional stateful rules
...
```

Example to allow SSH as client

- stateless mode:

```
$IPT -A OUTPUT -p tcp --sport 1024:65535 --dport ssh -j ACCEPT
$IPT -A INPUT  -p tcp --sport ssh --dport 1024:65535 -j ACCEPT
```

- stateful mode:

```
$IPT -A OUTPUT -p tcp --sport 1024:65535 --dport ssh -m state --state NEW -j ACCEPT
# the return packet will be allowed by the generic INPUT stateful rule
```

Special cases

- some protocols create a second related connection
- the ports of those connections are negotiated dynamicly
- `iptables` needs a module to understand those protocols

Example to allow FTP as client

```
# load kernel module
/sbin/modprobe ip_conntrack_ftp

# FTP stateful rule (as client)
$IPT -A OUTPUT -p tcp --sport 1024:65535 --dport ftp -m state --state NEW -j ACCEPT
```

Deny invalid packets

- this rule is optional because those packets will be denied by the default policy
- however, it helps the firewall performance if they are denied at the beginning of the script
- example

```
# Deny input invalid packets
$IPT -A INPUT -m state --state INVALID -j DROP
```

Deny invalid packets

- this rule is optional because those packets will be denied by the default policy
- however, it helps the firewall performance if they are denied at the beginning of the script
- example

```
# Deny input invalid packets
$IPT -A INPUT -m state --state INVALID -j DROP
```

- in some cases it might be useful to LOG before the DROP
  - but be carefull, this might increase a lot the size of the log files

EXERCISES

1. copy the script `firewall-on.sh` to `firewall-stateful-on.sh`
2. edit the file `firewall-stateful-on.sh` and
   - insert comments to follow the recommended structure (see slide 2)
   - insert the generic stateful rules
   - change to stateful mode the existing rules for DNS, HTTP, HTTPs, and SSH
   - add a stateful rule for FTP as a client
3. test the new firewall configuration

Personalized Chains

PERSONALIZED CHAINS

IPtables already has some default chains

- `INPUT`, `OUTPUT`, `FORWARD`, ...

However, it is possible to add personalized chains

- personalized chains help to avoid the repetition of rules
- work in a similar way to functions in a programming language

```
# create new personalized chain
$IPT -N NewChain

# adding rules to NewChain
$IPT -A NewChain -p tcp --dport http ACCEPT

# redirect packets to NewChain
$IPT -A OUTPUT -p ip -j NewChain
```

To eliminate all personalized chains

```
$IPT -X
```

Must be added to the initialization part of the script

```
...
# flush all the filtering rules
$IPT -F

# eliminate all personalized chains
$IPT -F
...
```

Targets

TARGETS

After `-j` a target must be defined

- `ACCEPT` and `DROP` were previously shown
- `RETURN`
- `LOG`
- `REJECT`
- check iptables documentation for more targets

RETURN target

- redirects packets to the previous chain
- in the default chains will redirects packets to the default policy

RETURN target

- redirects packets to the previous chain
- in the default chains will redirects packets to the default policy

Example to prevent a flood of new TCP connections

```
# create the syn_flood chain
$IPT -N syn_flood

# add rules to the syn_flood chain
$IPT -A syn_flood -m limit --limit 10/second --limit-burst 5 -j RETURN
$IPT -A syn_flood -j DROP

# redirect packets to the syn_flood change
$IPT -A INPUT -p tcp --syn -j syn_flood
```

LOG target

- stores packet information on the system's log files
- it is possible to specify the desired log level
- to check all options do `iptables -j LOG -h`

```
# create the syn_flood chain
$IPT -A INPUT -j LOG  --log-prefix "DROPPED " --log-level 4 --log-ip-options
↪  --log-tcp-options --log-tcp-sequence
```

LOG target

- stores packet information on the system's log files
- it is possible to specify the desired log level
- to check all options do `iptables -j LOG -h`

```
# create the syn_flood chain
$IPT -A INPUT -j LOG  --log-prefix "DROPPED " --log-level 4 --log-ip-options
↪  --log-tcp-options --log-tcp-sequence
```

To check the last log entries (press `ctrl+c` to end the command)

```
Terminal
user@linux:~$ tail -f /var/log/kern.log
Oct  5 18:46:24 linux kernel: [403696.818673] DROPPED INPUT IN=eth0 OUT=
↪  MAC=48:2a:e3:41:87:a5:18:e8:29:42:20:30:08:00 SRC=76.223.92.165 DST=192.168.1.6
↪  LEN=40 TOS=0x00 PREC=0x00 TTL=246 ID=12991 DF PROTO=TCP SPT=443 DPT=50196
↪  SEQ=1488720224 ACK=0 WINDOW=0 RES=0x00 RST URGP=0
```

REJECT target

- similar to DROP but the issuer will receive an answer stating that the packet was discarded
- to check all options do `iptables -j REJECT -h`

Examples:

```
# REJECT will send the default answer packet accordingly to the protocol
$IPT -A INPUT -p ip -j REJECT

# TCP connections
$IPT -A INPUT -p tcp -j REJECT --reject-with tcp-reset

# UDP connections
$IPT iptables -A INPUT -p udp -j REJECT --reject-with icmp-port-unreachable
```

Limits

LIMITS

IPtables allows to limit the amount of times a rule is applied

- to enable the limit option: `-m limit`
- to specify the rate: `--limit <rate>/<time_timeframe>`
  - `<rate>` – is a positive number
  - `<time_timeframe>` – is a time unit such as `second`, `minute`, `hour`, or `day`
- to specify the amount of packets before the limit is actulaly applied: `--limit-burst <n_packets>`
- default values: `--limit 3/hour --limit-burst 5`

Example:

```
# log only a sample of the events
$IPT -A INPUT -p ip -m limit --limit 3/hour -- limit-burst 5 -j ACCEPT
```

- the first 5 packets are accepted regardless of the rate they arrive
- then a rate of 3/hour is applied, which means it will ACCEPT 1 packet every 20 minutes
- the packets that arrive at a higher rate won't match the rule and will continue the rule checking process

Example:

```
# log only a sample of the events
$IPT -A INPUT -p ip -m limit --limit 3/hour -- limit-burst 5 -j ACCEPT
```

- the first 5 packets are accepted regardless of the rate they arrive
- then a rate of 3/hour is applied, which means it will ACCEPT 1 packet every 20 minutes
- the packets that arrive at a higher rate won't match the rule and will continue the rule checking process

What is the purpose of this rule?

```
$IPT -A INPUT -p tcp --syn -m limit --limit 1/s -j ACCEPT
```

Save and Restore

SAVE AND RESTORE

To save active rules to a file

```
# syntax
iptables-save [-c] [-t table_name]
```

- `-c` include the current values of all packet and byte counters in the output
- `-table_name` restrict output to only one table, if not specified, output includes all available tables
- the output is sent to `STDOUT`, if desired can be redirected to a file

Example

```
Terminal
user@linux:~$ sudo iptables-save -c > /tmp/iptables.txt
```

To restore rules from a file

```
# syntax
iptables-restore [-c] [-n]
```

- `-c` restore the values of all packet and byte counters
- `-n` don't flush the previous contents of the table
- reads from the `STDIN`

Example

```
Terminal
user@linux:~$ sudo iptables-restore -c < /tmp/iptables.txt
```

To apply the firewall rules after a reboot

```
Terminal
user@linux:~$ sudo apt install iptables-persistent
```

To update the rules stored for application after a reboot

```
Terminal
user@linux:~$ sudo dpkg-reconfigure iptables-persistent
```

Exercises

Note

Start a script from scratch in stateless mode for this exercise

Create the file `limit-test.sh` and add rules to

1. start the script with a deny default policy
2. allow the loopback interface
3. your computer should only be able to send only 1 `ping` packet every 10 seconds:

```
Terminal
user@linux:~$  ping 192.168.1.254
PING 192.168.1.254 (192.168.1.254) 56(84) bytes of data.
64 bytes from 192.168.1.254: icmp req=1 ttl=127 time=0.229 ms
64 bytes from 192.168.1.254: icmp req=2 ttl=127 time=0.227 ms
64 bytes from 192.168.1.254: icmp req=3 ttl=127 time=0.165 ms
64 bytes from 192.168.1.254: icmp req=4 ttl=127 time=0.224 ms
ping: sendmsg: Operation not permitted
(...)
ping: sendmsg: Operation not permitted
64 bytes from 192.168.1.254: icmp req=11 ttl=127 time=0.221 ms
ping: sendmsg: Operation not permitted
(...)
```

Copy `limit-test.sh` to `limit-test2.sh` and reverse the way it works

1. your firewall should denied 1 outgoing `ping` packet every 10 seconds:

```
Terminal
user@linux:~$  ping 192.168.1.254
PING 192.168.1.254 (192.168.1.254) 56(84) bytes of data.
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
64 bytes from 192.168.1.254: icmp req=4 ttl=127 time=0.227 ms
64 bytes from 192.168.1.254: icmp req=5 ttl=127 time=0.165 ms
(...)
64 bytes from 192.168.1.254: icmp req=8 ttl=127 time=0.222 ms
64 bytes from 192.168.1.254: icmp req=9 ttl=127 time=0.224 ms
ping: sendmsg: Operation not permitted
64 bytes from 192.168.1.254: icmp req=11 ttl=127 time=0.221 ms
64 bytes from 192.168.1.254: icmp req=12 ttl=127 time=0.219 ms
(...)
```

Continue editing the file `firewall-stateful-on.sh` and add rules to

1. reject incoming requests to the DNS on your computer
2. allow DNS outgoing requests only to the network `10.0.0.0/8`
3. log and drop all incoming packets marked as invalid
4. limit the amount of new incoming requests to 5 per second to all new incoming TCP connections
5. the first packet of all incoming connections must be logged (use personalized chains for this purpose)

Block access to Facebook webpages

1. create a new script from scratch
2. all rules for TCP and UDP allowed services must be written in stateful mode
3. allow all outgoing DNS packets
4. allow outgoing HTTP and HTTPs except for `facebook.com`

```
Terminal
user@linux:~$ whois -h whois.radb.net -- '-i origin AS32934' | grep ^route
route:   204.15.20.0/22
route:   69.63.176.0/20
route:   66.220.144.0/20
route:   66.220.144.0/21
route:   69.63.184.0/21
route:   69.63.176.0/21
route:   74.119.76.0/22
(...)
```

- the `whois` protocol is blocked inside `ipleiria.pt` network
- more information about identifying the facebook networks

For more information:

- Linux 2.4 Packet Filtering HOWTO
- Youtube videos

Terminal

```
user@linux:~$ man iptables
user@linux:~$ man iptables-save
user@linux:~$ man iptables-restore
user@linux:~$ iptables <option> -h
```