LeaseWeb labs
A blog about web hosting and software development

Setup your own Linux router using iptables – Part 1

When using Linux on servers we all know that one basic tool to secure the setup is iptables. Iptables not only allows you to secure your setup, it will also allow you create a routing service in a very controlled and efficient way.

Every distribution has it's how way to configure and deploy the desired set of rules, Ubuntu and it's variants use ufw service, Redhat and it's clones iptables service, OpenSuSE uses SuSEfirewall2 service, etc.

To be honest my preferred way of managing is the Redhat style. It's cleaner, doesn't have hidden features and it doesn't try to make it easier for the user. Although some of those functions may seam good in the long run they may become limitations.

So, why not make your own custom iptables startup script and iptables rule set that will be compatible with most of Linux distributions? This blog post will focus in two distinct areas:

1. Create an init script to enforce our iptables rule set
2. Create a set of iptables rules that could be put in place on a Linux gateway

Assumptions:

1. You have ip_forwarding enabled on your system (if not you may check this link)
2. You have a mainstream Linux distro with iptables and it's most used modules (like conntrack)

3. You have root access to the Linux GW
4. I'm using OpenSuSE 13.1 x86_64, nevertheless this guide should work on other distributions only with minor changes on configuration variables

Lets start by defining the scope of the rule set, the example scenario will be (the IPs were randomly selected) one Linux box serving as gateway for three networks):

- eth1 172.17.31.0/29 – lets call it orange and use it as DMZ (we will have the server providing external services in this network)
- eth2 192.168.9.0/24 – lets call it green and use it as users networks, having some servers on the subnet 192.168.9.0/9, we also want to give VPN users to this subnet
- tun0 10.238.23.1/28 – lest call it blue and use it for VPN access

In the examples we use different network cards but it can also be done with VLANs and/or with a plain network interface having all the logic based on src/dst IPs, ports and protocols.

On the Internet access interface (eth0, we may call it red) we will have 6 usable IP addresses 21.32.181.1-6. These IPs will be exposed to the Internet.

On those six available IP addresses we will want different services also exposed to the Internet, the network address translation table will be:

- 21.32.181.1 port 53 -> 172.17.31.1 port 53 (udp)
- 21.32.181.1 port 53 -> 172.17.31.1 port 53 (tcp)
- 21.32.181.2 port 80 -> 172.17.31.2 port 80 (tcp)
- 21.32.181.3 port 443 -> 172.17.31.3 port 443 (tcp)

Apart from the NAT table we will also make sure that:

- Access to other destination IP/port/protocol combination will be denied
- The default gateway to access the wan will be 21.32.181.6

In this example we won't talk about having more than one gateway, although it's easily achievable.

Finally we won't allow internal IP traffic to access the Internet freely (we all know how dangerous it is :)), so we will limit the access to the following ports/protocols:

- TCP: 25,80,110,144,443,465,993
- UDP: 53,1194
- ICMP: type 8

Please note that this doesn't limit the access from our Linux GW to the Internet, it will be applied on the forward chain so the effect will only be visible on the clients using our Linux GW as their GW. The Linux GW will have complete access to all the networks, including unfiltered access to the Internet.

Bellow is a diagram about how the network should like:

```
          ------------         ---------------------------------------
          |          |         | Linux GW:                           |
          | Internet |---------| Ext net (eth0): 21.32.181.1/29  *   |
          |          |         | Int net (eth1): 172.17.31.14/28     |
          ------------         | Int net (eth2): 192.168.9.254/24    |
                               | VPN net (tun0): 10.238.23.14/28     |
                               ---------------------------------------
                                /               |                \
             output            /          input |    output       \  output
            (Internet/        /      everywhere |   (to Internet)  \(to orange / Internet)
              orange)        /      to services |                   \
                  ----------------         ------------        ----------------
                     green                   orange                      blue
            192.168.9.0/24               172.17.31.0/28              10.238.23.0/28
                                         172.17.31.1/28 - udp 53
                                         172.17.31.2/28 - tcp 80
                                         172.17.31.3/28 - tcp 443

* the external network includes all the IPs between 21.32.181.1 - 21.32.181.6
```

Lets get back to the first target on this blog post: Create an init script.

Please create a file named pmso-fw (in reality you can name it whatever you like) in /etc/init.d

```
#!/bin/bash
# PMSO fw initializer
#
# pmso-fw:   Pedro Oliveira FW builder
#
# chkconfig: 345 03 03
# description:  This is a daemon for managing firewall scripts \
#               config file should be located in \
#               /etc/sysconfig/iptables
#
# notes:        please set rc.status according to your OS / Distro
# processname: pmso-fw
# pidfile: /var/run/pmso-fw.pid
#

. /etc/rc.status
rc_reset

SERVICE="pmso-fw"
IPT4="/usr/sbin/iptables"
PROCESS_APPLY="/usr/sbin/iptables-apply"
CONFIG_FILE="/etc/sysconfig/iptables"
LIST_RULES="/usr/sbin/iptables-save"
PIDFILE="/var/run/pmso-fw.pid"

test -f $CONFIG_FILE

start() {
  stop
  sleep 3650d &
  echo $! > $PIDFILE
```

```
32    echo -n $"Starting $SERVICE "
33    `$PROCESS_RESTORE < $CONFIG_FILE`
34    RETVAL=$?
35    rc_status -v
36  }
37
38  stop() {
39    echo -n $"Stopping $SERVICE "
40    $IPT4 -F && \
41    $IPT4 -X && \
42    $IPT4 -t nat -F && \
43    $IPT4 -t nat -X && \
44    $IPT4 -t mangle -F && \
45    $IPT4 -t mangle -X && \
46    $IPT4 -P INPUT ACCEPT && \
47    $IPT4 -P FORWARD ACCEPT && \
48    $IPT4 -P OUTPUT ACCEPT
49    RETVAL=$?
50    rc_status -v
51  }
52
53  restart() {
54    stop
55    start
56  }
57
58  stat() {
59    $LIST_RULES
60    RETVAL=$?
61    rc_status -v
62  }
63
64  # See how we were called.
65  case "$1" in
66    start|stop|restart)
67      $1
68    ;;
69    status)
70      echo -n "Checking status of $SERVICE "
71      rc_status -v
72    ;;
73    *)
74      echo $"Usage: $0 {start|stop|status|restart}"
75      rc_failed 2
76      rc_exit
77    ;;
78  esac
79  rc_exit
```

Lets describe what this script does:

1. The first section look a regular comment section, nevertheless it's in this zone that are configured the systemd options. Be careful if editing
2. /etc/rc.status – this line means that the /etc/rc.status file will be imported, this file contains the systemd control functions
3. General script configs and main utilities location section
4. Script functionality functions:
5. start() – The start function starts with stop, this may seam odd but if we are starting the FW service we want to guarantee that no rules will be duplicated. I also issue a sleep command, this is due to the fact that iptables isn't a process and we need to keep the process open so we have a clean process status
6. stop() – The stop function cleans all the tables
7. restart() – No explanation needed
8. stat() – Stat will use the systemd functions to return the status of the fw rules

We also need to make the script executable:

```
chmod u+x /etc/init.d/pmso-fw
```

Now you should create the systemd config file IF you use systemd. The file should be placed in: /etc/systemctl/system/pmso-fw.service and it will have the following content:

```
[Unit]
Description=PMSO-FW
After=network.target
Wants=network.service

[Service]
ExecStart=/etc/init.d/pmso-fw start
ExecStop=/etc/init.d/pmso-fw stop
RemainAfterExit=true
Type=oneshot

[Install]
WantedBy=multi-user.target
```

Finally lets setup the rule set.
The format of the this rule set is compatible with iptables-save and iptables-restore

The location of the file will be: /etc/sysconfig/iptables

```
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
:LOG-INPUT - [0:0]
:LOG-FORWARD - [0:0]
:LOG-BLUE-TO-GREEN - [0:0]
:LOG-GREEN-TO-BLUE - [0:0]
:RED-TO-ORANGE - [0:0]
:BLUE-TO-ORANGE - [0:0]
:BLUE-TO-GREEN - [0:0]
:GREEN-TO-ORANGE - [0:0]
:GREEN-TO-BLUE - [0:0]
:TO-RED - [0:0]
:TO-GREEN - [0:0]
:TO-ORANGE - [0:0]
:T0-BLUE - [0:0]
:FROM-RED - [0:0]
:FROM-GREEN - [0:0]
:FROM-ORANGE - [0:0]
:FROM-BLUE - [0:0]
###

-A LOG-INPUT -j LOG --log-prefix "IPTables-reject-INPUT: "
-A LOG-FORWARD -j LOG --log-prefix "IPTables-reject-FORWARD: "
-A LOG-BLUE-TO-GREEN -j LOG --log-prefix "IPTables-reject-BLUE-TO-GREEN: "
-A LOG-GREEN-TO-BLUE -j LOG --log-prefix "IPTables-reject-GREEN-TO-BLUE: "
###

-A INPUT -i lo -j ACCEPT
-A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A INPUT -p icmp -m limit --limit 2/sec -j ACCEPT
-A INPUT -d 172.17.31.14 -p tcp -m tcp --dport 22 -j ACCEPT -m comment --comment "Access t
-A INPUT -m limit --limit 2/sec -j LOG-INPUT
-A INPUT -m limit --limit 2/sec -j LOG-INPUT
-A INPUT -j REJECT --reject-with icmp-port-unreachable
###

-A FORWARD -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -i eth1 -o eth0 -s 172.17.31.0/28 -j TO-RED
-A FORWARD -i eth2 -o eth0 -s 192.168.9.0/24 -j TO-RED
-A FORWARD -i tun0 -o eth0 -s 10.238.23.0/28 -j TO-RED

```

```
-A FORWARD -o eth1 -d 172.17.31.0/28 -j TO-ORANGE
-A FORWARD -o eth2 -d 192.168.9.0/24 -j TO-GREEN
#-A FORWARD -o tun0 -d 10.238.23.0/28 -j TO-BLUE

-A FORWARD -i eth1 -s 172.17.31.0/28 -j FROM-ORANGE
-A FORWARD -i eth2 -s 192.168.9.0/24 -j FROM-GREEN
-A FORWARD -i tun0 -s 10.238.23.0/28 -j FROM-BLUE

-A FORWARD -i tun0 -s 10.238.23.0/28 -o eth2 -d 192.168.9.0/24 -j BLUE-TO-GREEN
-A FORWARD -i tun0 -s 10.238.23.0/28 -o eth1 -d 172.17.31.0/28 -j BLUE-TO-ORANGE
-A FORWARD -i eth2 -s 192.168.9.0/24 -o eth1 -d 172.17.31.0/28 -j GREEN-TO-ORANGE

-A FORWARD -i eth1 -o eth1 -j ACCEPT
-A FORWARD -i eth2 -o eth2 -j ACCEPT
-A FORWARD -i tun0 -o tun0 -j ACCEPT

-A FORWARD -m limit --limit 2/sec -j LOG-FORWARD
-A FORWARD -j REJECT --reject-with icmp-port-unreachable
###

-A BLUE-TO-GREEN -d 192.168.9.0/29 -j ACCEPT
-A BLUE-TO-GREEN -m limit --limit 2/sec -j LOG-BLUE-TO-GREEN
-A BLUE-TO-GREEN -j REJECT --reject-with icmp-port-unreachable

-A BLUE-TO-ORANGE -j ACCEPT

-A GREEN-TO-ORANGE -j ACCEPT

-A GREEN-TO-BLUE -m limit --limit 2/sec -j LOG-GREEN-TO-BLUE
-A GREEN-TO-BLUE -j REJECT --reject-with icmp-port-unreachable

-A TO-RED -p icmp -m icmp --icmp-type 8  -j ACCEPT
-A TO-RED -p tcp -m multiport --dports 25,80,110,144,443,465,993 -j ACCEPT
-A TO-RED -p udp -m multiport --dports 53,1194 -j ACCEPT

COMMIT
###

*nat
:PREROUTING ACCEPT [0:0]
:INPUT ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]

-A PREROUTING  -d 21.32.181.1 -p udp --dport 53 -j DNAT --to-destination 172.17.31.1:53
-A PREROUTING  -d 21.32.181.1 -p tcp --dport 53 -j DNAT --to-destination 172.17.31.1:53

-A PREROUTING  -d 21.32.181.2 -p udp --dport 53 -j DNAT --to-destination 172.17.31.1:53
-A PREROUTING  -d 21.32.181.2 -p tcp --dport 53 -j DNAT --to-destination 172.17.31.1:53

-A PREROUTING  -d 21.32.181.3 -p tcp --dport 80 -j DNAT --to-destination 172.17.31.2:80
-A PREROUTING  -d 21.32.181.3 -p tcp --dport 443 -j DNAT --to-destination 172.17.31.3:443

-A POSTROUTING -o eth0 -j MASQUERADE

COMMIT
###

*mangle
:PREROUTING ACCEPT [0:0]
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]
COMMIT
```

Now that we have the files ready we need to setup our newly created installation script, this can be done with the following commands if you use systemv:

```
insserv /etc/init.d/pmso-fw (if you use systemv)
/etc/init.d/pmso-fw status
```

or if you use systemd

```
systemctl daemon-reload
systemctl enable pmso-fw # to make it permanent
systemctl start pmso-fw # to start it
```

you'll be able to check the status of the fw with:

```
systemctl status pmso-fw
```

if you want to see the rules in use at any given moment you may use:

```
iptables-save
```

or

```
iptables -n -L --line-numbers
iptables -n -L -t NAT --line-numbers
```

After editing the rules you can also reload the service with:

```
systemctl restart pmso-fw
```

As you might have noticed we log the dropped packets, not all because if we have an exposed router the log will be immense but we limit the writes to two entries per second. This will allow us to debug something that isn't right on our configuration but won't fill our log file system or root partition (depending on how you partitioned your system).

In OpenSuSE 13.1 you may check your logs in "/var/log/firewall".

On the next post we will get into more detail on the rules, detailing what's done on the main blocks of the iptables rule set. I hope you have as much fun adapting this how to to your needs as I had making it.

Regards,
Pedro M. S. Oliveira

December 27, 2013 Pedro Oliveira (Innovation Engineer) Networking, Security, System Administration conntrack, firewall, gateway, iptables, linux, router