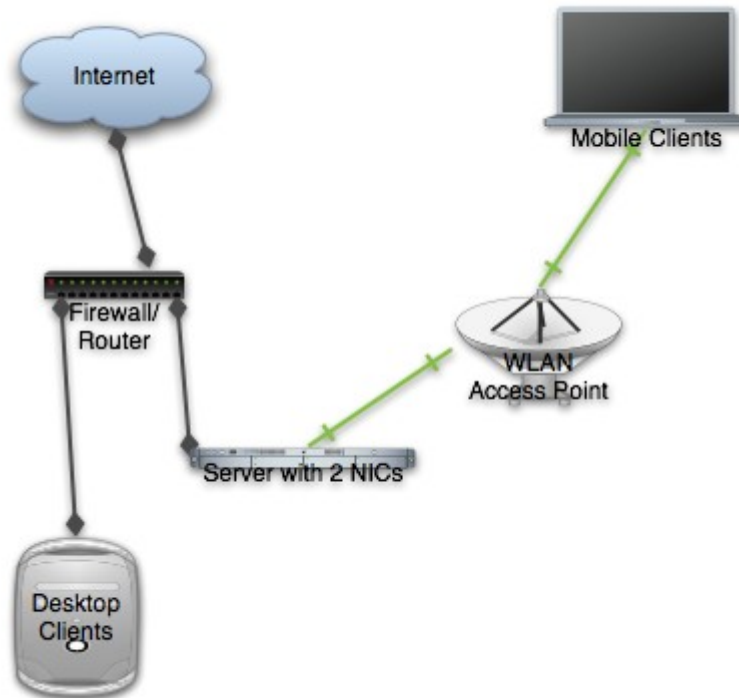


IPTABLES explained: Part 3 (Creating a complex IPTables script)

SCENARIO:

A friend of mine had a problem, he has a home network similar to mine and a Wireless Access Point (AP) that he wanted to use since mobility is everything. We sat down and designed his network and then wrote an iptables script for his setup. What follows is a more or less picture so you get the idea what the system looked like:



Now, the explanation. The AP is WEP capable only and WEP is good but not good enough for us security people (or anyone else really), so we decided to go this route: the server will have 2 NICs, one hooked up to the normal network and one going to the AP. The network on the AP is completely open, there is no WEP on the AP. This allows any mobile client to connect and get an IP in the mobile network. On the server we installed and configured OpenVPN (took about 3.492 minutes to install :)) and made it listen to a non standard port on the wireless network interface. We created a 3rd network within OpenVPN and modified it to route. Routing! I hear you scream, yes that is 3!! IP networks to access one. But here is the reason: routing, meaning different networks, allow you to VERY easily modify your IPTables scripts and restrict access. We then copied the OpenVPN server config file and named it differently, changed the virtual IP range and what interface to listen on. After a service restart we had OpenVPN now listening on both NICs. It might look like overkill but any person to try and crack your WLAN and use your internet access will go away because they will not succeed. And if you now forward port 9000 on your router to the MAIN interface IP port 9000 you can have a VPN connection from anywhere in the world to your home , securely! (use a DYNDNS service if you have a dynamic IP)

Anyway, back to our scenario. The Server has IP forwarding enabled this is the reason why the wireless network interface needs to be locked down. Once you have decided your networks please make sure you add the appropriate routes to our main router so all machines can find each other.

The 3 networks we are using here are :

- 192.168.0.0/24 for our MAIN network on eth
- 192.168.1.0/24 for our WLAN on eth1
- 192.168.2.0/24 for our OpenVPN WLAN network on tun0
- 192.168.3.0/24 for our OpenVPN PUBLIC network on tun1

SCRIPT:

Now that you have an idea of what we are trying to do, lets start writing the script that will do the following:

- * block all access from the WLAN to the internal network except port 9000 TCP for OpenVPN and ssh
- * allow all traffic from the MAIN network to the server
- * allow traffic from anywhere to the MAIN server interface port 9000 OpenVPN
- * allow traffic from the OpenVPN network to the server and main network
- * allow all localhost traffic
- * use statefulness wherever possible

Looks difficult? Well it really isn't. We start as usual with the beginning:

```
#!/bin/sh
IPT="/sbin/iptables"
#GLOBAL VARIABLES
good=?192.168.0.0/24,192.168.2.0/24,192.168.3.0/24?
bad=?192.168.1.0/24?
any=?0.0.0.0/0?
#Flush old rules, delete the firewall chain if it exists
echo "-> FLUSHING ALL TABLES"
$IPT -F
$IPT -X
#CONSTRUCT THE DEFAULT POLICY - DROP EVERYTHING
echo "-> SETTING POLICIES"
$IPT -P INPUT DROP
$IPT -P OUTPUT DROP
$IPT -P FORWARD DROP

$IPT -N local
$IPT -N state
$IPT -N lan
$IPT -N vpn
$IPT -N drop
#ENABLE IP_FORWARDING
echo "-> ENABLING IP FORWARDING"
echo 1 > /proc/sys/net/ipv4/ip_forward
```

...and most of these things you already know. The -N parts define the new chains we create already so we can add rules to them later on. As the script is a normal shell script we can add normal shell commands there as well as you see at the bottom to enable ip forwarding.

First we want to make the state traffic ready. This way when traffic comes in it wont have to traverse all rules until it hits the state chains if it is a statefull connection already.

```
#Check for statefullness
echo "-> CHECKING STATE TRAFFIC"
$IPT -A state -p tcp,udp -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPT -A INPUT -p tcp,udp -j state
$IPT -A OUTPUT -p tcp,udp -j state
$IPT -A FORWARD -p tcp,udp -j state
```

Then, for our first real rule we start by adding the localhost traffic as allow and then pointing to the local tale from INPUT, and OUTPUT as FORWARD is not needed since we are not forwarding traffic through localhost.

```
#ACCEPT LOOPBACK
echo "-> ACCEPTING LOCALHOST TRAFFIC"
$IPT -A local -s 127.0.0.1/8 -d 127.0.0.1/8 -j ACCEPT
$IPT -A local -s 127.0.0.1/8 -d 0.0.0.0/0 -j ACCEPT
$IPT -A local -j drop
$IPT -A INPUT -s 127.0.0.1/8 -j local
$IPT -A OUTPUT -s 127.0.0.1/8 -j local
```

Next we write the rules to deny all traffic coming towards our MAIN network on the WLAN interface:

```
#BLOCK WiFi FROM REACHING 192.168.1.0/24 or 192.168.0.0/24
echo "-> BLOCKING WiFi"
$IPT -A FORWARD -i eth1 -p all -s $any -d $good -j DROP
```

Then we allow DNS queries from our MAIN network. Keep in mind, the default policies are locked down completely. This means you have to allow every service specifically.

```
#accept dns
echo "-> ALLOWING DNS"
$IPT -A INPUT -i eth0 -p udp -s $good -d 192.168.0.9/32 --dport 53 -j ACCEPT
```

Now we will allow traffic between all GOOD networks and traffic coming to the server on the MAIN interface.

```
#ACCEPT FROM LAN
echo "-> ALLOWING LAN"
$IPT -A lan -i eth0 -p all -s $good -d $good -j ACCEPT
$IPT -A lan -i eth0 -p all -s $good -d $good -j ACCEPT
$IPT -A lan -i tun1 -p all -s $good -d $good -j ACCEPT
```

```
$IPT -A lan -i tun0 -p all -s $good -d $good- -j ACCEPT  
$IPT -A INPUT -j lan  
$IPT -A OUTPUT -j lan  
$IPT -A FORWARD -j lan
```

Now we will accept OpenVPN connections on both interfaces and SSH on both. This will nail it down quite nicely.

```
#Accept OPENVPN. Duh.  
echo "-> ACCEPTING OPENVPN"  
$IPT -A vpn -i eth0 -p tcp -s $any -destination-port 9000 -j ACCEPT  
$IPT -A vpn -i eth1 -p tcp -s $bad -destination-port 9000 -j ACCEPT  
$IPT -A vpn -o eth1 -p tcp -source-port 9000 -j ACCEPT  
$IPT -A vpn -o eth0 -p tcp -source-port 9000 -j ACCEPT  
$IPT -A INPUT -i eth1 -p tcp -s $bad -destination-port 22 -j ACCEPT  
$IPT -A INPUT -i eth0 -p tcp -s $any -destination-port 22 -j ACCEPT  
$IPT -A INPUT -j vpn  
$IPT -A OUTPUT -j vpn
```

And last but not least we make the log chain, this allows us to prefix the log entry with anything so it is easier to spot in the logs.

```
$IPT -A log -j LOG -log-level info -log-prefix "IPTABLES: "  
$IPT -A log -j DROP  
$IPT -A INPUT -j log  
$IPT -A OUTPUT -j log  
$IPT -A FORWARD -j log
```



There you have it, now you can save the file and run it. After that you can check with iptables -L and see if your tables are there. Also verify your traffic, is everything working. If it isn't check the logs it tells you from where it did not allow something.

Any thoughts and comments are welcome. A special thanks goes to mobsec.com for letting me write about this adventure?