

## ***IPTables Explained : Part 1 (a tutorial to UNDERSTANDING and CREATING your own rules)***

Ok, so you have Linux installed and maybe a few services running like FTP, WWW etc. Now you want to share your Internet connection. Not the easiest but the most efficient way to do so is to use the Linux machine as a firewall/server. Linux comes with a very powerful firewall built in. It is totally scriptable but its not the easiest to understand. What follows is (hopefully) a good yet simple guide to WHAT iptables does and HOW it works.

What we need to understand is how iptables works before we can start scripting. Once we get it down, scripting is easy, its a simple shell script that can be written by anyone. IPTables is and Netfilter are the same, iptables is the program you use to administrate the actual firewall. For simplicity we will call it iptables, but keep in mind the actual name of the software is netfilter.

### **UNDERSTANDING:**

iptables is a IP filtering firewall that is based on matching IP traffic based on rules that you specify. The objective is that you match traffic with specific groups of rules, called chains. You can have as many chains as you want but notice that performance might get a hit on slow machines if you have a lot of complex filters.

We always want to start with defining the policies in IPTables. The policies are the default rules that are applied in case your traffic does NOT match any filters you have set. So if you want to allow only certain things in or out of your network and the default policies are set to DENY, any traffic that is not allowed by any of your filters will be denied. This is called rules matching. The 3 policies that need to be set are: INPUT, FORWARD and OUTPUT.

The policies define the default action for the DEFAULT CHAINS, INPUT (traffic COMING TO the machine directly), OUTPUT (traffic GOING FROM the machine directly) and FORWARD (traffic COMING FROM or GOING TO a networked machine on the other side of the firewall). The 2 actions you can set in these policies are ACCEPT and DROP.

You can also define variables, which in this case means just a collection of ports or protocols, so that you don't have to write a new rule for each port you want to allow. In case of a web/ftp-server you could make a variable which contains ports, 21,22,80 and 443 and refer to that variable instead of writing 5 lines. These variables can then be used later in a single line.

In order for you to direct traffic to pass through a certain filter/set you have to create the filter-set first THEN direct the traffic. This makes sense as you cannot direct something to some place that does not exist. So basically the main part in your script is detailing the filters and creating new chains (since the packets go through the rules as if they were following a chain) and in the end you direct the traffic from the default chains to your filter-set. You can be VERY granular as to what traffic you want to direct where but be careful with complex scripts, it can get confusing. A good way is to first visualize what you are trying to do. As you can see in the picture, its a simple diagram of a normal IPTables flow.

So to summarize to setup a IPTables script we must follow this structure:

- define the variables first
- set the default policies second
- write your filter chains third
- point your traffic to the chains

## CREATING:

CREATING: Now that we understand what is going on inside IPTables, let's start with creating the script. Create a new file called iptables.sh and fill in the following:

```
#!/bin/sh #variables first!  
ipt="/sbin/iptables"  
std_ports="22,80,443"  
lan="10.0.0.0/24,192.168.0.0/24"  
any="0.0.0.0/0"
```

As you can see in the beginning we start a standard shell script, and then we made 4 variables, `ipt`, `std_ports`, `lan` and `out`. The `ipt` variable is more of a convenience than anything else. It allows us later to start each line that will be added to IPTables with `$ipt` instead of typing `/sbin/iptables`. The second variable, `std_ports`, is actually a listing of ports that we consider "standard access" in our example. It is SSH, HTTP (www) and HTTPS (ssl www). The `lan` and any variables are actually networks, as with the port list, you can have multiple listings.

Now follows the first "real" thing. We want to clear out any old rules before we apply our own. To do that we add these 3 lines:

```
$ipt -F  
$ipt -Z  
$ipt -X
```

This will do 3 things, it will FLUSH (-F) all standard chains, or all rules one by one. ZERO (-Z) all standard chains and statistic counters, and ERASE (-X) all user created chains.

So now we have a clean slate and we can actually start making our own policies and rules. First up, the policies, these are created by appending a -P , the policy and then the default action after the ipt command:

```
$ipt -P INPUT -j DROP  
$ipt -P FORWARD -j DROP  
$ipt -P OUTPUT -j DROP
```

By default we want to deny all traffic by default that comes TO this machine, drop all traffic destined for OTHER machines and accept traffic coming FROM this machine. This means that if

there is no rule matched with a packet that comes from/to/through this machine, it will drop it.

Now that the policies are set we are ready to create our first chain.

Our first chain that we will create is the state chain. The state chain checks if you already have had a connection to/from this machine. In which case it will not traverse all chains or rules again but just let you through. This speeds things up considerably and is very useful. This is not a security risk as the firewall keeps track of the connections. It is nearly impossible to "slip" through.

to create a state table add this to your script:

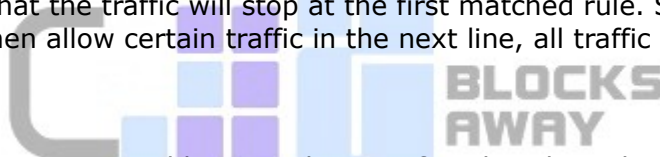
```
$ipt -N states
```

```
$ipt -A states -m state --state ESTABLISHED,RELATED -j ACCEPT
```

```
$ipt -A states -m state --state NEW -s $lan ! -d 192.168.0.250/32 -j ACCEPT $ipt -A states -j DROP
```

This is now already a pretty complex example and requires a bit of explanation. However, when you understand each building block, you will already grasp quite a bit of how IPTables rules are made.

In the first line, we created a new chain called states. When you make a new chain or filter-set you HAVE to have a -N ,which stands for NEW. So you first create the chain with -N and then you APPEND filters to the table with -A. This should explain the structure of a chain - First create it and then add to it. Please remember: all filters are matched top-to-bottom first match and exit. This means that the traffic will stop at the first matched rule. So if you first block all traffic in a table and then allow certain traffic in the next line, all traffic will be stopped as the first rule matches.



Now, to the next part in our second line, -i eth0 specifies that this rule will apply to the network interface eth0. In other words you can bind different rules to different interfaces. If no interface is given, it is applied to ALL. The -m tells IPTables to use a specific IPTables module, in this case the state module which controls and manages statefulness. For a complete listing of all modules and what they do (with a syntax explanation) check a later of this series. Some modules have more options that's why you have to specify with the -state what part of the module you are using (the short example is the multiport module which has -dport and -sport options). After you specified the -state you need to specify WHICH state you want to control with this rule. There are 3 states, NEW, ESTABLISHED and RELATED. In this case what we want is to allow already established and their related connections in and out. The last part then is the action to take, -j ACCEPT will jump to ACCEPT all related and established connections. If you would put there DROP it would do just that.

The third line is just as the second except for a few changes after the -state. In this case it will accept new (-state NEW ) connection attempts from (-s) the lans (\$lan) that are NOT (!) destined (-d) to 192.168.0.250 directly. This is useful for example if that machine is your local IP address and you do not want direct connections to it. The last line just tells the filter to DROP all the other state related things as they do not match any rules we specified. So, so far our script looks already more like a script as you can see here:

```
#!/bin/sh #variables first!
```

```
ipt="/sbin/iptables"
```

```
std_ports="22,80,443"
```

```
lan="10.0.0.0/24,192.168.0.0/24"
```

```
any="0.0.0.0/0"
```

```
$ipt -F
```

```
$ipt -Z
```

```
$ipt -X
```

```
$ipt -P INPUT -j DROP $ipt -P FORWARD -j DROP
```

```
$ipt -P OUTPUT -j DROP
```

```
$ipt -N states $ipt -A states -m state --state ESTABLISHED,RELATED -j ACCEPT
```

```
$ipt -A states -m state --state NEW -s $lan ! -d 192.168.0.250/32 -j ACCEPT $ipt -A states -j DROP
```

To summarize IPTables are not hard, just complex. Once you have the basics down, writing very complex and long rule sets becomes more and more easy. Something that helps is to visually draw your rules before you start. Also something that could help is to make flowcharts of your rule flow with the actions laid out.

