IPTABLES - INTERESTING MODULES

Well, I suppose by now it is no secret that I like the linux iptables firewalling setup. I have used a fair number of other offerings but for what you can do versus the cost of it, iptables cannot be beat. Recently I was playing around with some of the other modules that I haven't really used, both to learn as well as to see if I could use any of them. And I found out a couple of interesting little things. Let me share with you some of my thoughts on these modules...

Connection Limiting
This is done using the *connlimit* match. What is does is allow a rule to match (or not) a specified number of tcp connections. Lets look at an example..

```
$IPT -A FORWARD -p tcp --syn --dport 22 -d 192.168.10.200 -m connlimit ! --connlimit-above 2 -j ACCEPT
```

What this does is that it limits any client's ssh connections, to the host with the ip of 192.168.10.200, to only 2 concurrent connections. Now this is useful for limiting the number of connections to a server if excess connections are a problem. Kinda sounds like it could be used as a kind of protection against DOS attacks does it not? Well try this example...

```
$IPT -A FORWARD -p tcp --syn --dport 80 -d 192.168.10.200 -m connlimit --connlimit-above 16 --connlimit-mask 24 -j REJECT
```

You see, the connlimit match allows you to also specify the netmask to which it must apply the match. It defaults to 30, which is for a single client, but in our example we have limited incoming http connections to 16 per each class C network With some tweaking this can be used quote nicely for anti-DOS purposes. This setup can also be extended to help database/email/ssh servers from excessive connections. It is also very useful if used in conjunction with a password lockout policy to protect against brute-force attacks. If you utilize it along with the *recent* match functionality your firewall can be even more unfriendly to people trying to utilize excessive connections.

Connection Size
This is using the *connbytes* match. This allows you to specify a connection size range, a direction for the monitoring, and what units the counting process must use. Look at this example..

```
$IPT -A OUTPUT -p tcp --sport 80 -m connbytes --connbytes-mode bytes --connbytes-dir both --connbytes 5000000: -j REJECT
```

Here we are saying that any tcp connection where the source port is http must be counted in bytes, traffic going both ways should be counted and anything above 5000000 bytes (roughly 5MB) should be denied. Very useful for stopping large downloads. Actually very useful for stopping large downloads even if a tunneling or p2p process is using port 80. Another possible use would be to help prevent insider threat, by stopping someone from doing an excessively large database query for example. Again, using this with the *recent* match functionality allows you to be a lot nastier to someone abusing your bandwidth. It is obviously important that this rule comes above your stateful inspection rules, otherwise it will not be able to monitor the full tcp transaction.

Quota Limiting
This is done with the *quota* match. This match allows you to specify, in bytes, how much traffic that rule is allowed to match. In using this match it is important that it happens above your stateful inspection rules so that it can see the amount of traffic. Lets take a look at an example..

```
$IPT -A FORWARD -p tcp --dport 80 -s 192.168.10.200 -m quota --quota 5000000 -j ACCEPT
```

This will allow the host 192.168.10.200 roughly 5MB of web traffic, after which the rule will no longer match and should therefore hopefully be dropped. This match is very useful for keeping your troublesome users in check, or even just to help monitor as when the rule is matched, the quota counter drops when viewing a listing of the rules. One thing to remember is that once the quota is reached that is it until you remove the rule and reinsert it. So I recommend if you are going to use it, that you create a separate ruleblock and use a cronjob to remove and reinsert your quota rules at suitable intervals.

String Matching
The *string* match, this is one match that I was keen to try for a while but because it wasn't able to be used with the 2.6 kernel I never used it. But I see that in the 2.6.15 kernel it is part of the actual kernel code for the *netfiler/iptables* section. Look like I get to play after all. This match allows the firewall to look inside the packet and try to match ascii or hex strings. Look at the example..

```
$IPT -A INPUT -p tcp --dport 80 -d 192.168.10.200 -m string --algo bm --string "badcmd" -j REJECT
```

Here I am looking at all http traffic heading to my web server, if I find the ascii string of "badcmd" in any packet, then that packet is dropped. Now this match needs to be used above the stateful inspection rules are it will only ever see the first packet of any connection. Now I am not suggesting using this match as a replacement for proxies, inline packet scrubbing or application level firewalls. But it is useful for stopping individual threats against your company, for stopping viruses/malware/exploits until the

proper ruleset or patch become available, or anything else you can think of. For example, you could stop anyone using the VRFY command on your SMTP server. Stuff like that is always useful. But bear in mind that this type of deep packet inspection can incur a performance penalty.

Time Aware

Another one is the *time* match. This allows us to match a packets timestamp. Take a look at this example..

```
$IPT -A FORWARD -p tcp --dport 22 -m time --timestart 8:00 --timestop 22:00 --days Mon,Tue,Wed,Thu,Fri -j ACCEPT
```

Here we have said we will only forward ssh traffic through the firewall between 8:00 and 22:00, and only from Monday to Friday. Any attempts to ssh through the firewall outside of these times and days will be denied. This is very useful for allowing only certain administration times, for example, most remote administrators already lock their access rules down using a source address, but you can go a step further and specify the times when you are at work. This way you can cut down even further the chances of someone brute forcing the account or using it for nefarious purposes. It is also useful to help with insider threat. You could shutdown a large part of your network during the times when no-one is supposed to be around, so even if they have a username and password, and access to the building, they are locked out - this is very useful when dealing with wireless networks and on bridging firewalls. Yes, I know you could do this type of stuff by using a cronjob with different rulesets, but this helps make it a bit easier.

Final Words

These are some of things I have been playing with and think are fairly useful, many uses can be found for these matches -and the previously mentioned ones- with just a bit of original thinking. Remember that the tighter you can make your firewall the better, so doing extra checks before you allow traffic is always a good thing. Take a look at these matches, play around, learn and have fun.