THE LINUX EXPERIENCE

TLUG Presentation

"Netfilter" - Firewalling and Routing with Linux

Madison Kelly,

Lead Technician, TLE

March 09, 2004

http://TheLinuxExperience.com   416-361-6377

Intro:

My name is Madison Kelly. I am the Lead Technician for my employer, "The Linux Experience". I want to give this talk tonight as a way of saying thank you to TLUG for helping me so many times when I have turned to you all for help.

Like many of you, I am a Linux sysadmin who wants to share a bit of what she has learned about the Linux Netfilter package. The application I will talk about this evening is the '`iptables`' command available in just about every distribution of Linux with the 2.4 kernel and up.

There is a lot of information I hope to cover in the next hour. I ask you all to refrain from making comments or asking questions until I am finished. If I make a mistake please feel free to correct me, if it is significant. If we can get through the presentation within the one hour time period we will have at least half an hour for a question and answer (and correction) period.

Finally before we begin; My goal today is not to have you leave and immediately be able to write wonderfully secure firewalls and fully effective routers. I will not be covering things like syn flood protection or other more "advanced" ways of matching, mangling and handling packets. That would require far more time than we have. What I do want to do is help you see how a firewall is built and how to address some of the most common pitfalls. Once you have the basics down it should be relatively easy to apply anything else you will learn later. The beauty of '`iptables`' is how logical it is to use. Everything is a variation on a theme.

Let's Begin:

We commonly use the term "Router" & "Firewall" interchangeably. This is because the two functions are very often implemented together. They are different though, so let me cover quickly what each actually is.

 - *Firewall:* Makes decisions on an IP packet's fate. Done by looking at the packet's routing info and sequentially comparing it against a list of "rules". If a match is found, do what the rule says. If no match is found then apply the "default" rule. The firewall can mangle a packet which changes some aspect of the packet like source or destination address and it can choose to ACCEPT or DROP (deny) a packet.

 - *Router*: Connects two or more subnets (networks). The routing of packets is simply a function. The decision of whether a packet belongs where it is going has already been made by the firewall. The routing aspect is simply the function of pushing packets onto different subnets.

 - *Bridge*: I won't be covering bridges but I do want to mention them here for clarity. A bridge is a router between two entirely different types of networks. An example would be a machine used to "bridge" a common TCP/IP network and a Novell IPX/SPX network. Given the near total implementation of TCP/IP by all modern vendors bridges are rarely needed any more.

Routing & Firewall Concepts:

If you use a firewall simply to protect your machine then you are probably only using a firewall and not doing any routing or packet mangling. If you have two or more network connections on your machine though (ie: a broadband Internet link and a NIC connected to a home LAN) then you probably want to implement a router with your firewall. It is very rare that you would route packets though without some type of firewalling.

http://TheLinuxExperience.com   416-361-6377

If all you need is a basic firewall to protect your personal computer or server then you are in luck because that is quite basic and easy to implement. In this case all we need to worry about is what we will allow in to our machine (INPUT) and what we are willing to let out (OUTPUT).

If you want to connect two or more TCP/IP networks though (ie: share your Internet connection with other computers) then you are still in luck because even the more "advanced" features of routing are easy to learn and logical. We do have more features and options to learn though like "FORWARD"ing, PREROUTING, POSTROUTING and user added chains.

Basic Structure of an iptables Firewall:

'iptables' rules are always structured in the following manner:

```
# /path/to/iptables -t <table> -A|I <chain> <what-to-match> <what-to-do>
```

For example;

```
# /sbin/iptables -t filter -A INPUT -i eth1 -p tcp d-dport 22 -j ACCEPT
     ^              ^         ^        ^        ^                 ^
     1              2         3        4        5                 6
```
1. The 'iptables' path and executable
2. Add this to the 'filter' table, the table which filters packets
3. Append rule to chain 'INPUT' which catches packets destined for the firewall itself
4. Match packets coming in on interface 'eth1'
5. Match packets trying to connect on TCP port 22 (ssh)
6. If it matches, let it in!

In this case, we are telling 'iptables' [/sbin/iptables] to work in the 'filter' table [-t filter] , specifically in the 'INPUT' chain [-A INPUT] (which catches any packets destined for the firewall machine itself). We are telling 'iptables' to 'Append' the rule to the end of the 'INPUT' chain and to match packets coming in the interface 'eth1' [-i eth1] trying to connect to the TCP port 22 [-p tcp --dport 22]. When a match is made, we will jump to the 'ACCEPT' chain [-j ACCEPT] which is a special built-in destination that allows the packet through.

Another example would be;

```
# iptables -t nat -I PREROUTING -d 1.2.3.4 -j DNAT --to-destination 3.4.5.6
   ^         ^      ^             ^           ^
   1         2      3             4           5        6
```
1. This time, 'iptables' is in our $PATH so we can omit the directory
2. Here we are going to work in the 'nat' table, the table used for modifying packets
3. This time we are going to 'Insert' the rule to the beginning of our PREROUTING chain
4. Match packets destined for the IP 1.2.3.4
5. When matched, jump to the special 'DNAT' chain used to write a new destination IP
6. Make the new destination IP 3.4.5.6

In this case we are telling 'iptables' [iptables] to work in the 'nat' table [-t nat] which is a special table specifically for the three built-in chains 'PREROUTING', 'POSTROUTING' and 'MASQUARADING' (a specialized version of POSTROUTING). These NAT (Network Address Translation) chains are used to modify a packet's routing information. In this example we specifically are going to match packets originally destined for the IP address '1.2.3.4' [-d 1.2.3.4] and when a match is found, jump it to the 'DNAT' chain [-j DNAT] (Destination NAT). Normally this would be where the rule would end but because

http://TheLinuxExperience.com   416-361-6377

this is a special built-in chain we need to tell it what we want to make the new destination IP. Specifically we want the new destination to be the IP address '3.4.5.6' [--to-destination 3.4.5.6].

All rules you will ever build for 'iptables' will follow this pattern. It is this common method that makes using 'iptables' so easy and powerful!

The last thing to mention before we begin is the flow of the rules. The 'iptables' program will initially look at a packet and pass it through the 'nat' table's 'PREROUTING' chain to see if and DNAT rules apply.

Next it determines if the packet is leaving the firewall machine itself, coming into the firewall machine itself or last if it is coming in one network device and asking to go out another. Once it determines this it will place the packet onto the 'filter' table's 'OUTPUT', 'INPUT' or 'FORWARD' chain respectively. Regardless of which chain the packet is placed on the flow of the packet will essentially be the same. The packet will be compared with each rule in the chain in sequential order until a rule is matched. If a rule is matched the packet will be jumped to the appropriate chain. If the rule fails to match all the rules in a chain, the packet will be jumped to the chain's default chain (ACCEPT or DROP).

A packet will inevitably be jumped to either the 'ACCEPT' or 'DROP' filter chains where the packets ultimate fate is met. Before it reaches there though a packet may (though by no means must it) meet any number of special or custom chains. Special chains are like the one we saw above called 'DNAT'.

A custom chain on the other hand is a chain you as the user can define to help you sort and keep track of your rules. A custom chain is hit from within one of the three built-in chains; 'INPUT', 'OUTPUT' or 'FORWARD'. You can jump from a custom chain to another custom chain but ultimately the packet will return to the source chain if no rules are matched along the way. An example of a use for a custom chain would be to separate lists of rules based on what device a packet is coming in on or trying to get out on. For example, let's say the Internet connection is on device 'eth0' and our office LAN is on device 'eth1' we may wish to create these rules right at the beginning of our 'filter' table's 'FORWARD' chain:
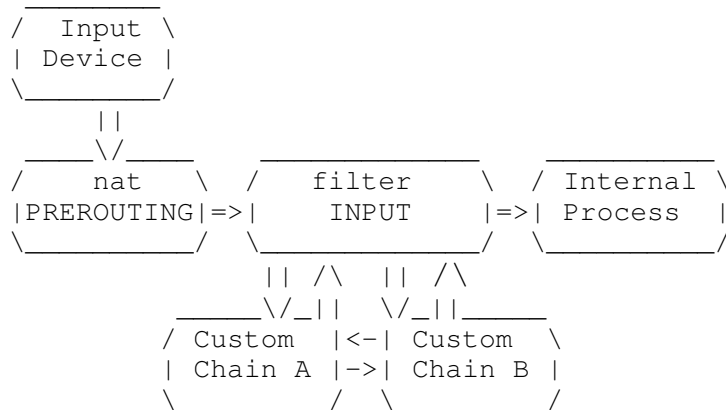
```
# iptables -t filter -A FORWARD -i eth0 -o eth1 -j LANIN
# iptables -t filter -A FORWARD -i eth1 -o eth0 -j LANOUT
```

These rules would immediately shunt packets going to the given network devices onto our custom filter chains 'LANIN' and 'LANOUT'. From there on we would append any rules specific to packets coming into our LAN to our custom chain 'LANIN' and vice versa. This is particularly helpful when we have multiple network cards with networks requiring different rules. Without these custom chains it would be far harder to keep the rules for each network apart. We will see an example of this in the third example firewall.
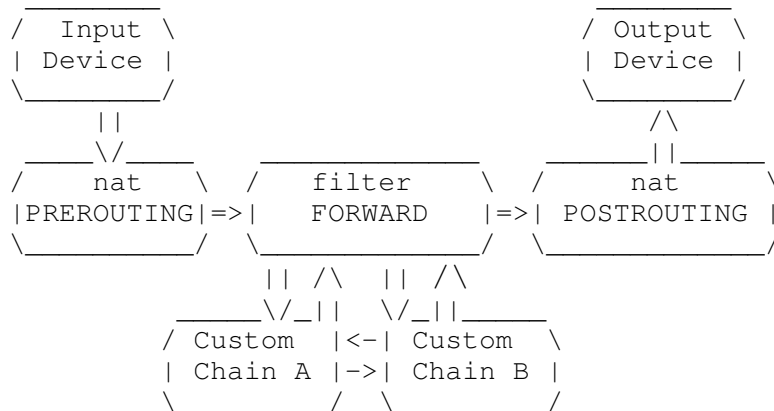
Finally, after a packet has traveled the PREROUTING nat chain and our filter chains it is passed through one last nat chain before being sent back out; POSTROUTING. This is the nat chain where Source NAT and it's specialized sister 'MASQUARADING' is applied. If a packet matches a rule in the nat chain POSTROUTING is jumped to either the 'SNAT' or 'MASQ' chain where the original source of the packet is re-written to have the new source defined in the rule.

Here are some rough ASCII graphic of how a packet travels through 'iptables'. Remember that a packet could be jumped to 'DROP' in the 'filter' chains and never make it through. In fact, that is the very purpose of a firewall; restricting packets:
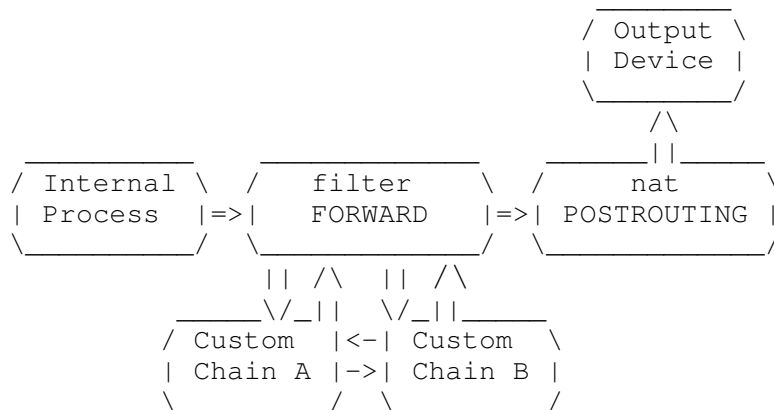
```
                        [ INPUT chain ]
         _____
        /  Input  \
        | Device  |
        _____/
            ||
        ____\/____      _____     _____
       /   nat    \    /   filter    \   / Internal \
       |PREROUTING|=>|    INPUT     |=>| Process  |
       _____/    _____/   _____/
                        || /\   || /\
                   _____\/_||   \/_||_____
                  / Custom  |<-| Custom  \
                  | Chain A |->| Chain B |
                  _____/   _____/




                       [ FORWARD chain ]
         _____                           _____
        /  Input  \                         /  Output \
        | Device  |                         | Device  |
        _____/                         _____/
            ||                                  /\
        ____\/____      _____     _____||_____
       /   nat    \    /   filter    \   /    nat      \
       |PREROUTING|=>|   FORWARD    |=>| POSTROUTING |
       _____/    _____/   _____/
                        || /\   || /\
                   _____\/_||   \/_||_____
                  / Custom  |<-| Custom  \
                  | Chain A |->| Chain B |
                  _____/   _____/




                        [ OUTPUT chain ]
                                            _____
                                           /  Output \
                                           | Device  |
                                           _____/
                                               /\
         _____     _____     _____||_____
        / Internal \   /   filter    \   /    nat      \
        | Process  |=>|   FORWARD    |=>| POSTROUTING |
        _____/    _____/   _____/
                        || /\   || /\
                   _____\/_||   \/_||_____
                  / Custom  |<-| Custom  \
                  | Chain A |->| Chain B |
                  _____/   _____/




                        [ Chain Path ]

         Filter Chain
         _____
```

```
[PREROUTING]-->| rule 1        |          Custom Chain A
               | rule 2        |      _____
               | rule 3        | /-->| rule 1         |          Custom Chain B
               | rule 4, jump --/    | rule 2         |      _____
               |  - return -  |<-\   | rule 3         | /-->| rule 1          |
               | rule 5        | |   | rule 5, jump --/    | rule 2          |
               | rule 6        | |   |  - return -  |<-\   | rule 3          |
               | rule 7        | |   | rule 6        | |   | rule 4          |
               | rule 8        | |   | rule 7        | |   | rule 4          |
               |  [Default]    | |   |               |  \---  [Fall back] |
[POSTROUTING]<---  [Policy ]    | |   |  - return -  |<-\   | rule 4          |
               |_____|     \---  [Fall Back] |    |_____|
                                     |_____|
```

Let's Start Building:

Let's look at a basic firewall simply designed to protect the computer running it. In this case, as I mentioned, all we need to worry about is controlling what we are going to let in and what we want to keep out. For this we only need to work with two chains; INPUT and OUTPUT.

Before we do anything we need to ensure that we are starting with a clean slate. To do this we need to "flush" any existing chains. This will always include the three built-in filter chains INPUT, OUTPUT and FORWARD plus the two built-in nat chains PREROUTING and POSTROUTING If you have any custom chains that you are going to re-use remember to flush them too.

```
# iptables -t filter -F INPUT
# iptables -t filter -F OUTPUT
# iptables -t filter -F FORWARD
# iptables -t nat -f PREROUTING
# iptables -t nat -f POSTROUTING
```

It is safe to say that we will trust everything leaving our computer but we most certainly do NOT trust everything trying to get in. An exception to this "Trust everything going out" rule would be, for example, wanting to block your kids from accessing certain websites like 'sex.com'. We will see how to do this in the third example firewall.

Before we build any rules we first need to set "Default Policies" for our three built-in filter chains. We already decided to let everything out so we will set our OUTPUT chain to ACCEPT any packet hitting the end of it's chain by default. Seeing as we plan to add no more rules, this will then apply to everything leaving the machine.

```
# iptables -t filter -P OUTPUT -j ACCEPT
```

We are not going to provide any routing at all so we will set the FORWARD chain to DROP any packet that get placed onto this chain by default. If you only have one network device (say a normal modem or a network card connected to a high-speed modem) then no packet should ever be placed on the FORWARD chain anyway because by definition the FORWARD chain is there to filter packets coming in one network device and aimed at leaving another.

```
# iptables -t filter -P FORWARD -j DROP
```

To be safe we also want to set the default policy for the INPUT chain to DROP. This way only what we specifically allow will get in and everything else will be blocked. This is always the best way by far to

http://TheLinuxExperience.com   416-361-6377

build a safe firewall; be it simple or complex.

```
# iptables -t filter -P INPUT -j DROP
```

Now that we have our default policies in place we need to start adding our rules. Rules work by examining one or more aspects of a packet and seeing if it matches the criteria we have defined in a rule and if it does, what do we do with it. Lets use an example `INPUT` filter rule I use very frequently in real life. It is a good place to start because it shows a few different ways to match packets to rules. Remember though that in order for a match to be made the packet must match ALL criteria. If you get carried away packets you want to match might be missed. On the other hand don't be too lax or the opposite might happen and you could match (and `ACCEPT`) packets you would rather keep out.

In my work I frequently need access to a client's server to install updates or make changes. To facilitate this I need to leave myself a way in. Now the easy way to do this would be to simply add the following rule;

```
# iptables -t filter -A INPUT -p tcp --dport 22 -j ACCEPT
```

There is a pretty big problem with this though. With this rule we are leaving port 22 open to every client on the Internet. It could be argued that this would be okay because the '`sshd`' daemon would protect the machine. This adds another point of compromise though. No matter what, we will always have to worry about an exploit being found in the '`iptables`' code. Why add to our worries by having to worry about the '`sshd`' code too when we can block all traffic we don't know from ever getting to '`sshd`' in the first place? I do this by only letting into port 22 packets from a network card I trust by matching it's unique MAC address.

```
# iptables -t filter -A INPUT -p tcp --dport 22 -m -mac-address \
> 00:11:22:33:44:55 -j ACCEPT
```

It is still possible that an attacker could mimic our MAC address but to do this he or she would need to know what it was in the first place and that would be -substantially- harder then simply hitting an otherwise open port 22. Of course the trade off is that our own access is limited to the client machine because now we need to be at a particular machine. However I have found that by adding the MAC address of my office & home computers to a given firewall that for the most part I can get to one of those machines when I need to.

Now that we have one rule to let ourselves into our machine in a pinch we need to let more mundane packets in. As you know, any connection to a network is useless if communication can't go both ways. What good is it to ask a web server to pass you the contents of a website if your firewall will simply block the server's response? That said, we certainly do not want to just allow anything from that server to get in, either. So, how do we resolve this dubious quandary?

Easy! There is a function in '`iptables`' called "connection tracking" that can tell that an incoming packet is there because it is part of an `ESTABLISHED` connection or a new connection based on another `RELATED` connection. An example of an `ESTABLISHED` connection would be packets coming back from the remote machine during an SSH session on port 22. To ensure these `ESTABLISHED` packets make it back in past our firewall we need to add the following rule to the `INPUT` chain;

```
# iptables -t filter -A INPUT -m state --state ESTABLISHED -j ACCEPT
```

Now lets look at `RELATED` packets. Many user-space web apps like instant messengers, P2P software and classic FTP work by opening new connections on other ports in addition to the one they first where opened on. In these cases the packets are not seen as part of an `ESTABLISHED` connection

http://TheLinuxExperience.com

416-361-6377

but rather as being new though RELATED connections. There is no reason why any of these programs should ever need to open a RELATED port below 1024 so unlike the above we will add a restriction to our RELATED rule;

```
# iptables -t filter -A INPUT -p tcp --dport 1025:65535 -m state --state RELATED \
> -j ACCEPT
```

Notice how I used the ':' to separate the first and last port number in the range? Good!

This should suffice for almost all user-space apps. Now lets say we want to host a small website on this machine. We need a way to allow packets trying to make entirely new connection requests to our web server, usually on port 80, into the machine. We can do this by adding the following rules;

```
# iptables -t filter -A INPUT -p tcp --dport 80 -j ACCEPT
# iptables -t filter -A INPUT -p udp --dport 80 -j ACCEPT
```

You can simply repeat these rules for each port number and service you want to open.

Here is a trick you might find useful; most Linux machines have the file '/etc/services' which lists most major services by name and their associated ports. If you have this file then you can use service names in the place of port numbers. For example, these rules would have worked just as well as the rules above;

```
# iptables -t filter -A INPUT -p tcp --dport http -j ACCEPT
# iptables -t filter -A INPUT -p udp --dport http -j ACCEPT
```

At this point you should now have a fully functional firewall! You can examine the rules that you have currently in place by entering '# iptables-save'. You can save these rules to a file that can be easily restored by issuing '# iptables-save >iptables.out'. To restore the firewall simply issue '# iptables-restore <iptables.out'. This can then easily be scripted and set to automatically run at boot. Alternatively you can write all of your rules into a shell script and run that when you need to [re]load your firewall. The later is my personal preference because it allows me to use variables for things that I use in multiple rules that might change like which device connects to which network.

Time for a little more fun:

You now have the basics of building a firewall for a single computer. Now it's time to have a little more fun and start playing with FORWARD'ing, NAT'ing and routing.

The very first thing we need to do is tell the kernel to turn on forwarding.

```
# echo 1 > /proc/sys/net/ipv4/ip_forward
```

In order to provide forwarding and routing we have to have at least two network devices, otherwise what would we route? We will also have to now play with the third built in chain, FORWARD.

Most of us only have one IP to play with. Most of us, being the post-modern technophiles that we are, want to make the most of this IP too. In order to setup our next scenario I need to set the stage... Our firewall box will be a dedicated firewall & router with two network cards; 'eth0' connects to a DSL modem and 'eth1' connects to our LAN via a hub. Our ISP issues us a dynamic IP each time we connect to the Internet which we do using a PPPoE client that creates the virtual device 'ppp0' that we will talk

to. On our LAN we will use the subnet `192.168.1.0/24` and '`eth1`' will have the IP `192.168.1.1`. We will share our Internet connection with any client on this internal LAN subnet.

The problem we run into when we want to share a network connection with several computers or devices is that we have only one Internet address to use on the network. We need a way to make sure that when LAN client "A" requests a web page through the router that the answer doesn't go to client "B", stop at the firewall itself or get sent to all the clients on the LAN.

This requesting of Internet resources on behalf of LAN clients and sorting and `FORWARD`'ing replies to clients behind the firewall is handled by Source Network Address Translation (`SNAT`). In our case, we have a dynamic public IP to the Internet that is likely to change if our Internet connection is lost and has to be re-established. Because of this we will use a specialized form of `SNAT` called "Masquerading" (`MASQ`). Unlike normal `SNAT`, `MASQ` does not try to maintain established Internet connections if the link is lost. Doing so is a waste because our public IP has likely changed so existing connections to Internet resources will already have been lost anyway.

When we use `SNAT` or `MASQ` the router will accept a request for an Internet resource, such as a request for the website '`kernel.org`' from the LAN client "A" at IP address `192.168.1.10`, go out and get the web page on the client's behalf, remember that client "A" made the request for '`kernel.org`' when the reply comes, and pass the page back to client "A". It can simultaneously make requests on behalf of numerous other clients, remember who asked for what and route replies back to the proper client as the various replies come back. This is the very nature or `SNAT` and `MASQ`.

Now we know what `SNAT` and `MASQ` are, and remember that for now we are going to use `MASQ` because our public IP is dynamic, lets add the rule to `MASQ` our LAN subnet `192.168.1.0/24` behind the firewall's public IP;

```
# iptables -t nat -A POSTROUTING -s 192.168.1.0/24 -o ppp0 -j MASQ
```

What this rule will do is match a packet coming from the LAN subnet `192.168.1.0/24` wanting to go out device '`ppp0`' (the Internet) and when matched jump the packet through the special chain '`MASQ`' in the `nat` table '`POSTROUTING`'. '`MASQ`' is a chain that will change the source of a packet to appear to come from the IP address of the firewall automatically so we do not need the '`--to-source <a.dd.re.ss>`' that we normally need with `SNAT`. In normal `SNAT` '`iptables`' would try to maintain connections if the Internet link was lost and re-established.

By far the most common mistake people make when building a firewall is confusing the `POSTROUTING` chains `SNAT`/`MASQ` which are implemented *-after-* a packet goes through the firewall with the `POSTROUTING` chain `DNAT` which is implemented *-before-* a packet goes through the firewall. You must keep this straight in your mind when building your rules. If you try to match an the original IP of a packet that has been `DNAT`'ed to a new one, you will miss it. Likewise in reverse for trying to match the final IP address of a packet that will be `SNAT`'ed on it's way out will miss, too.

At this point we can now build a firewall that will protect itself, share it's Internet connection with the clients on our LAN subnet and protect these clients from the crud on the Internet.

The Big Time:

The last major function of a firewall & router is to protect a group of servers from attack using more than one public IP. It is common for a company to have a corporate Internet connection with which they get a slice of static public IP addresses.

For example, where I work we have 32 public IP addresses, 29 of which we can assign to machines that we would like in some capacity to make available on the Internet. For reference, a subnet with 32 IP addresses is known as a '/27' subnet which is the same as saying our subnet uses the subnet mask of '255.255.255.224'. When writing a firewall rule both '111.222.33.32/27' and '111.222.33.32/255.255.255.224' mean the same thing.

In the next few examples I will use the following info; our specific subnet will be 111.222.33.32/27, it will have the public IP address 111.222.33.34 on 'eth0' and have two separate subnets behind it. One for a LAN of office computers and one for a LAN of public servers. The office LAN will have the subnet '192.168.1.0/24' and the firewall will use device 'eth1' at '192.168.1.1' to connect to it. Our server LAN will be on the subnet '198.168.2.0/24' and the firewall will use the IP address '192.168.2.1' through the device 'eth2' to connect to it .

Each server will be configured as follows:

Server 1:
Internal IP = 192.168.2.10
External IP = 111.222.33.35
Running services/ports = SSH (22), POP3 (25), HTTP (80), SMTP (110)

Server 2:
Internal IP = 192.168.2.11
External IP = 111.222.33.36
Running services/ports = SSH (22), HTTP (80), NNTP (119), HTTPS (443)

We could add as many servers to our firewall as we wish (and have public IPs for) by simply adding more of the same types of rules.

Now to pull it all together! What we want our firewall/router to do;

- Only let into itself certain machines on ports 22 (ssh). We will limit connections to machine's with specific network card MAC address just as we did before. We will let everything out of the firewall.
- Only let established connections and related connections above ports 1024 into the office LAN. We will let most connections out onto the Internet. We will block instant messengers (MSN Messenger for this example), block connections to Hotmail and Yahoo! Mail and log but allow connections to 'Monster.ca' and 'Workopolis.com' job search pages. (Yes, we are evil ;)). We will SNAT the entire LAN behind the firewall's public IP address.
- For each server on the server LAN we will DNAT (PREROUTING) connection requests to each given public IP onto the appropriate given internal IP before filtering it and conversely SNAT (POSTROUTING) connections back from the internal IP to the proper public IP just before sending packets back out. We will limit connections to each server to only the ports needed for the services they are offering. We will NOT allow any server from the server subnet to connect to a client on the office LAN and we will force office LAN clients to connect to servers using their public IPs as though they were on the Internet proper.

Now we have what we want to do clear in our mind it is time to start building our list of rules. Because our firewall router is going to be a little more complex we are going to start using custom chains to help us sort everything out.

Now lets begin;

http://TheLinuxExperience.com
416-361-6377

First off we need to build our NAT rules. Remember that we will SNAT our trusted LAN clients behind our firewall's static public IP and NAT in both directions our server LAN clients to their respective IPs.

```
# iptables -t nat -A POSTROUTING -s 192.168.1.0/24 -j SNAT --to-source \
>  111.222.33.34
# iptables -t nat -A POSTROUTING -s 192.168.2.10 -j SNAT --to-source \
>  111.222.33.35
# iptables -t nat -A POSTROUTING -s 192.168.2.11 -j SNAT --to-source \
>  111.222.33.36
# iptables -t nat -A PREROUTING -d 111.222.33.35 -j DNAT --to-destination \
>  192.168.2.10
# iptables -t nat -A PREROUTING -d 111.222.33.36 -j DNAT --to-destination \
>  192.168.2.11
```

Next we will create our custom chains so that us poor humans can keep track of what is going on better. I use the following myself but remember that you can use any names or sort rules in any way that you want. For me, I like creating a chain for inbound and outbound connections for the firewall itself and for each LAN subnet.

```
# iptables -t filter -N FWIN
# iptables -t filter -N FWOUT
# iptables -t filter -N SRVIN
# iptables -t filter -N SRVOUT
# iptables -t filter -N LANIN
# iptables -t filter -N LANOUT
```

First thing we will do is look at which device a packet is coming in on and what it's destination is and push packets to the relevant chain. For this we will match devices using '-i' to match the input device and '-o' to match the output device. For the firewall custom chains we will add our jumps in the INPUT and OUTPUT chains. The rest we will match in the FORWARD chain.

```
# iptables -t filter -A INPUT -j FWIN
# iptables -t filter -A OUTPUT -j FWOUT
# iptables -t filter -A FORWARD -i eth0 -o eth2 -j SRVIN
# iptables -t filter -A FORWARD -i eth2 -o eth0 -j SRVOUT
# iptables -t filter -A FORWARD -i eth0 -o eth1 -j LANIN
# iptables -t filter -A FORWARD -i eth1 -o eth0 -j LANOUT
# iptables -t filter -A FORWARD -i eth1 -o eth2 -j SRVIN
# iptables -t filter -A FORWARD -i eth2 -o eth1 -j LANIN
```

This set of rules should catch any packet going through the firewall in any direction. Now we can start adding rules to our custom chains without stepping on our own feet.

We will add the usual MAC address matching rule to our FWIN chain and let everything out of the firewall so FWOUT will remain empty for now.

We are not letting anything other than the ESTABLISHED and RELATED packets into the LAN so we will leave 'LANIN' with those rules only.

We do want to place restrictions on what our office LAN clients are allowed to connect to. Normally our rule of thumb is to block everything and then poke holes. This is one case of an exception. It would be virtually impossible to poke enough holes to make the Internet useful to our LAN clients unless we had -very- restrictive rules. Instead we are going to open everything and simply block problem websites (by name or IP, our choice) and ports.

http://TheLinuxExperience.com   416-361-6377

```
# iptables -t filter -A LANOUT -p tcp --dport 6891:6900 -j DROP
# iptables -t filter -A LANOUT -p udp --dport 6891:6900 -j DROP
# iptables -t filter -A LANOUT -d hotmail.com -j DROP
# iptables -t filter -A LANOUT -d passport.net -j DROP
# iptables -t filter -A LANOUT -d mail.yahoo.com -j DROP
```

Some companies may be paranoid about employee dissatisfaction. The following rules will tell them when and who visits certain websites like the job seeker websites "Monster.ca" and "Workopolis.com". Keep in mind of course that if employees find out this feature exists it may make otherwise happy people mad!

```
# iptables -t filter -A LANOUT -d monster.ca -m limit --limit 1/minute \
> -j LOG --log-prefix "Monster.ca - Someone isn't happy: "
# iptables -t filter -A LANOUT -d workopolis.com -m limit --limit 1/minute \
> -j LOG --log-prefix "Workopolis.com - Someone isn't happy: "
```

Notice the two new rules? They tell 'iptables' to match a maximum of once per minute [-m limit --limit 1/minute] and when they do match to log the details to 'syslogd' with a custom prefix [-j LOG --log-prefix "Monster.ca - Someone isn't happy: "]. If you wish, you can add to '--log-level #' (ie: '-m limit --limit 1/minute -j LOG --log-level 6 --log-prefix "<string>"' or use 'syslog.conf' to define how much data is written to the logs.

We will let everything out from our server LAN which means that for now 'SRVOUT' will be left empty. We simply need to add rules to open ports into each server for the services they will be providing . These will essentially be the same rules we used on our first firewall to allow 'http' connections into the firewall except now we will add them to the 'SRVIN' chain and match using the DNAT'ed server IP addresses. By not adding any further rules we will by default block any connections between our two LANs because our default FORWARD policy is 'DROP'.

```
# iptables -t filter -A SRVIN -d 192.168.2.10 -p tcp --dport 22 -j ACCEPT
# iptables -t filter -A SRVIN -d 192.168.2.10 -p udp --dport 22 -j ACCEPT
# iptables -t filter -A SRVIN -d 192.168.2.10 -p tcp --dport 25 -j ACCEPT
# iptables -t filter -A SRVIN -d 192.168.2.10 -p udp --dport 25 -j ACCEPT
  And so on...

# iptables -t filter -A SRVIN -d 192.168.2.10 -p tcp --dport 22 -j ACCEPT
# iptables -t filter -A SRVIN -d 192.168.2.10 -p udp --dport 22 -j ACCEPT
  And so on...
```

Firewall One:

Protecting our own single-honed (one network device) computer from the outside world.

```
#!/bin/bash
# The most basic firewall we can build; protecting our own machine!

# First we flush the built-in chains:
iptables -t filter -F INPUT
iptables -t filter -F OUTPUT
iptables -t filter -F FORWARD
iptables -t nat -f PREROUTING
iptables -t nat -f POSTROUTING

# Now we will set our default policies for the built-in chains:
```

```
iptables -t filter -P OUTPUT -j ACCEPT
iptables -t filter -P FORWARD -j DROP
iptables -t filter -P INPUT -j DROP

# Leaving a way into the machine from a remote machine.
# - Option 1
# The following rule isn't that safe but it is the most flexible (do you trust
#  sshd to protect your machine?) Uncomment this if you do.
#iptables -t filter -A INPUT -p tcp --dport 22 -j ACCEPT
#
# - Option 2
# This is much safer than the above rule but it is far less flexible because
#  you have to be at the machine with the specified MAC address.
iptables -t filter -A INPUT -p tcp --dport 22 -m -mac-address \
 00:11:22:33:44:55 -j ACCEPT

# Here we are telling the firewall to let packets in that are connected to
#  an ESTABLISHED or RELATED (above port 1025) connection.
iptables -t filter -A INPUT -m state --state ESTABLISHED -j ACCEPT
iptables -t filter -A INPUT -p tcp --dport 1025:65535 -m state --state RELATED \
 -j ACCEPT

# We are going to run a small website on our machine so we need to open
#  TCP and UDP ports 80. We could use either:
#
#iptables -t filter -A INPUT -p tcp --dport http -j ACCEPT
#iptables -t filter -A INPUT -p udp --dport http -j ACCEPT
# or
#
iptables -t filter -A INPUT -p tcp --dport 80 -j ACCEPT
iptables -t filter -A INPUT -p udp --dport 80 -j ACCEPT

# Done!
```

Firewall Two:

Sharing our Internet connection with clients we trust on our LAN using masquerading.

```
#!/bin/bash
# A very common Internet sharing firewall/router script

# Kernel stuff: Turning on forwarding
echo 1 > /proc/sys/net/ipv4/ip_forward

# First we flush the built-in chains:
iptables -t filter -F INPUT
iptables -t filter -F OUTPUT
iptables -t filter -F FORWARD
iptables -t nat -f PREROUTING
iptables -t nat -f POSTROUTING

# Now we will set our default policies for the built-in chains:
iptables -t filter -P OUTPUT -j ACCEPT
```

```
iptables -t filter -P FORWARD -j DROP
iptables -t filter -P INPUT -j DROP

# Here we start off by MASQ'ing our internal LAN clients behind the firewall
#  machine. Remember that we are using MASQ because our public IP is dynamic!
iptables -t nat -A POSTROUTING -s 192.168.1.0/24 -o ppp0 -j MASQ

# Leaving a way into the machine from a remote machine.
# Keep in mind that this rule only applies to SSH connections to the firewall
#  itself because we are placing this in the filter table 'INPUT'.
iptables -t filter -A INPUT -p tcp --dport 22 -m -mac-address \
 00:11:22:33:44:55 -j ACCEPT

# Here we are telling the firewall to let packets in that are connected to
#  an ESTABLISHED or RELATED (above port 1025) connection. We will also allow
#  similar packets into the LAN from the Internet interface.
iptables -t filter -A INPUT -m state --state ESTABLISHED -j ACCEPT
iptables -t filter -A INPUT -p tcp --dport 1025:65535 -m state --state RELATED \
 -j ACCEPT
iptables -t filter -A FORWARD -i ppp0 -o eth1 -m state --state ESTABLISHED -j ACCEPT
iptables -t filter -A FORWARD -i ppp0 -o eth1 -p tcp --dport 1025:65535 \
 -m state --state RELATED -j ACCEPT

# We aren't hosting any Internet services in this scenario so that's it,
#  we are done Firewall Two!
```

Firewall Three:

Sharing our Internet connection with trusted office LAN clients and untrusted server LAN clients. We will also use NAT to host and protect server LAN clients behind the firewall. For each server we will NAT one static public IP to one static internal IP on the server LAN subnet. For this firewall we are using these conditions:

General:
Static Internet IP slice = `111.222.33.32/27`
Office LAN Subnet = `192.168.1.0/24`
Server LAN subnet = `192.168.2.0/24`

LAN:
Blocked Sites/Ports = `MSN Messenger (TCP/UDP 6891-6900), Hotmail (hotmail.com & passport.net), Yahoo! Mail (mail.yahoo.com)`
Logged Sites = `Monster.ca, Workopolis.com`

Firewall:
Internet Device = '`eth0`', static public IP = '`111.222.33.34`'
Office LAN Device = '`eth1`', static IP = '`192.168.1.1`'

Server LAN Devices = 'eth2', static IP = '192.168.2.1'

Server 1:
Internal IP = 192.168.2.10
External IP = 111.222.33.35
Running services/ports = SSH (22), POP3 (25), HTTP (80), SMTP (110)

Server 2:
Internal IP = 192.168.2.11
External IP = 111.222.33.36
Running services/ports = SSH (22), HTTP (80), NNTP (119), HTTPS (443)

```bash
#!/bin/bash
# A more advanced firewall for showing off some of the funky capabilities of
#  iptables!

# Kernel stuff: Turning on forwarding
echo 1 > /proc/sys/net/ipv4/ip_forward

# First we flush the built-in chains:
iptables -t filter -F INPUT
iptables -t filter -F OUTPUT
iptables -t filter -F FORWARD
iptables -t nat -f PREROUTING
iptables -t nat -f POSTROUTING

# Now we will set our default policies for the built-in chains:
iptables -t filter -P OUTPUT -j ACCEPT
iptables -t filter -P FORWARD -j DROP
iptables -t filter -P INPUT -j DROP

# Here we start off by SNAT'ing our internal LAN clients behind the firewall's
#  public IP and NAT'ing our servers in both directions
iptables -t nat -A POSTROUTING -s 192.168.1.0/24 -j SNAT --to-source \
 111.222.33.34
iptables -t nat -A POSTROUTING -s 192.168.2.10 -j SNAT --to-source \
 111.222.33.35
iptables -t nat -A POSTROUTING -s 192.168.2.11 -j SNAT --to-source \
 111.222.33.36
iptables -t nat -A PREROUTING -d 111.222.33.35 -j DNAT --to-destination \
 192.168.2.10
iptables -t nat -A PREROUTING -d 111.222.33.36 -j DNAT --to-destination \
 192.168.2.11

# Now We need to create our custom filter tables before we can jump to them!
iptables -t filter -N FWIN
iptables -t filter -N FWOUT
iptables -t filter -N SRVIN
iptables -t filter -N SRVOUT
iptables -t filter -N LANIN
iptables -t filter -N LANOUT

# We are going to route all packets directly into our custom filter chains
#  so that we can keep our rules organized easier. We do this by matching
#  a packet's input device and it's output device.
iptables -t filter -A INPUT -j FWIN
```

http://TheLinuxExperience.com   416-361-6377

```
iptables -t filter -A OUTPUT -j FWOUT
iptables -t filter -A FORWARD -i eth0 -o eth2 -j SRVIN
iptables -t filter -A FORWARD -i eth2 -o eth0 -j SRVOUT
iptables -t filter -A FORWARD -i eth0 -o eth1 -j LANIN
iptables -t filter -A FORWARD -i eth1 -o eth0 -j LANOUT
iptables -t filter -A FORWARD -i eth1 -o eth2 -j SRVIN
iptables -t filter -A FORWARD -i eth2 -o eth1 -j LANIN


# Now that our packets have been sorted we can start adding rules to our
#  custom chains.
#
# First off let's add our standard ESTABLISHED and RELATED rules to each
#  xIN chain:
iptables -t filter -A FWIN -m state --state ESTABLISHED -j ACCEPT
iptables -t filter -A FWIN -p tcp --dport 1025:65535 -m state --state RELATED \
 -j ACCEPT
iptables -t filter -A LANIN -m state --state ESTABLISHED -j ACCEPT
iptables -t filter -A LANIN -p tcp --dport 1025:65535 -m state --state RELATED \
 -j ACCEPT
iptables -t filter -A SRVIN -m state --state ESTABLISHED -j ACCEPT
iptables -t filter -A SRVIN -p tcp --dport 1025:65535 -m state --state RELATED \
 -j ACCEPT


# We will open up port 22 into the firewall from machines with trusted
#  MAC addresses
iptables -t filter -A FWIN -p tcp --dport 22 -m -mac-address \
 00:11:22:33:44:55 -j ACCEPT


# Now we will create our trusted office LAN outbound restrictions
iptables -t filter -A LANOUT -p tcp --dport 6891:6900 -j DROP
iptables -t filter -A LANOUT -p udp --dport 6891:6900 -j DROP
iptables -t filter -A LANOUT -d hotmail.com -j DROP
iptables -t filter -A LANOUT -d passport.net -j DROP
iptables -t filter -A LANOUT -d mail.yahoo.com -j DROP


# Now we get mean ;)
# These rules do not DROP or ACCEPT a packet, they simply log to 'syslogd'
#  the matches once per minute. Once a minute should catch everyone without
#  flooding our logs.
iptables -t filter -A LANOUT -d monster.ca -m limit --limit 1/minute \
 -j LOG --log-prefix "Monster.ca - Someone isn't happy: "
iptables -t filter -A LANOUT -d workopolis.com -m limit --limit 1/minute \
 -j LOG --log-prefix "Workopolis.com - Someone isn't happy: "


# Time to open up ports for each server:
# - Server 1; SSH (22), POP3 (25), HTTP (80), SMTP (110)
iptables -t filter -A SRVIN -d 192.168.2.10 -p tcp --dport 22 -j ACCEPT
iptables -t filter -A SRVIN -d 192.168.2.10 -p udp --dport 22 -j ACCEPT
iptables -t filter -A SRVIN -d 192.168.2.10 -p tcp --dport 25 -j ACCEPT
iptables -t filter -A SRVIN -d 192.168.2.10 -p udp --dport 25 -j ACCEPT
iptables -t filter -A SRVIN -d 192.168.2.10 -p tcp --dport 80 -j ACCEPT
iptables -t filter -A SRVIN -d 192.168.2.10 -p udp --dport 80 -j ACCEPT
iptables -t filter -A SRVIN -d 192.168.2.10 -p tcp --dport 110 -j ACCEPT
iptables -t filter -A SRVIN -d 192.168.2.10 -p udp --dport 110 -j ACCEPT


# - Server 2; SSH (22), HTTP (80), NNTP (119), HTTPS (443)
iptables -t filter -A SRVIN -d 192.168.2.10 -p tcp --dport 22 -j ACCEPT
iptables -t filter -A SRVIN -d 192.168.2.10 -p udp --dport 22 -j ACCEPT
iptables -t filter -A SRVIN -d 192.168.2.10 -p tcp --dport 80 -j ACCEPT
```

http://TheLinuxExperience.com

416-361-6377

```
iptables -t filter -A SRVIN -d 192.168.2.10 -p udp --dport 80 -j ACCEPT
iptables -t filter -A SRVIN -d 192.168.2.10 -p tcp --dport 119 -j ACCEPT
iptables -t filter -A SRVIN -d 192.168.2.10 -p udp --dport 119 -j ACCEPT
iptables -t filter -A SRVIN -d 192.168.2.10 -p tcp --dport 443 -j ACCEPT
iptables -t filter -A SRVIN -d 192.168.2.10 -p udp --dport 443 -j ACCEPT

# That's all folks!
```

Thank you for your time and I hope you are able to take something useful away from this presentation. If you have any questions I can be reached on the TLUG mailing list or off-list at 'mkelly@thelinuxexperience.com'.

About "The Linux Experience";

TLE is a migration and support company in the greater Toronto area. We specialize in helping clients migrate away from proprietary solutions in favor of Open Source Software. TLE offers end-to-end support, training and services during all stages of migration including extended support post-migration. TLE serves as a single point of contact for all of their client's technology needs.

Helpful Links:

 - The Netfilter/iptables project.
`http://www.netfilter.org/`
    Note: They are the authors of 'iptables' and have immensely helpful documentation!

 - MonMotha's IPTables Firewall
`http://monmotha.mplug.org/firewall/index.php`
    Note: This is an excellent scripted firewall that supports multiple protected LANs plus DMZ'ed (not filtered) servers. I learned a -lot- dissecting this script; many great uses of advanced `iptables` features!

 - The `iptables` man page
```
$ man iptables
```

 - The Linux Experience
`http://thelinuxexperience.com`
    Note: My employer! We are a migration and support company. Our objective is migrating clients from proprietary to open-source and supporting them during and after. Hire us and I pay my rent ;).

Madison Kelly,
Lead Technician

-= The Linux Experience =-
http://thelinuxexperience.com

http://TheLinuxExperience.com   416-361-6377