Building a Linux Firewall

Matthew D. Rossmiller

November 11, 2003

1 Introduction

A major problem in modern system administration is securing Internet connected networks. The goal is to maximize security without losing the benefits of the network. A major weapon system administrators have in the quest for security is the firewall. A well tuned firewall is the first line of defense for networked machines.

This report covers firewall configuration options for the Linux operating system. It assumes a general familiarity with firewall and Internet Protocol (IP) networking concepts. Addendum A provides an overview of firewall concepts and terminology for the uninitiated. Addendum B contains a brief introduction to the Internet Protocol (IP) networking concepts relevant to a discussion of firewall systems.

The remainder of this paper is organized into four sections. In section two we will look at four separate firewall implementations for Linux; three open-source and one commercial. Then, in sections three and four we will examine one of the four implementations, netfilter/iptables, in detail. Section 5 contains some general advice for firewall configuration and general network security.

2 Linux Firewall Implementations

Open-source firewall support for Linux has existed in one form or another for nearly a decade. Since the first version, ipfwadm, two additional major versions have been created; ipchains, and iptables. This section evaluates the features of each of the major open-source firewall implementations for Linux. We'll also look at Firewall-1 from Check Point Software Technologies Inc. — a commercial firewall available for many operating systems, including Linux. Figure 1 contains an overview of the features of each firewall covered in this section.

2.1 ipfwadm

In 1994, ipfw was ported to Linux (kernel version 1.1.x) from BSD. A user-space tool for configuring the ported ipfw code named ipfwadm was created. The name, 'ipfwadm' is commonly used to refer to the first version of firewall support in Linux.

Ipfwadm is a fairly competent layer 3 & 4 packet filter. While it supports NAT, it does not perform general connection tracking needed for a stateful firewall. Logging and accounting of accepted/dropped packets is good, and the configuration options for accounting are quite flexible. ipfwadm also supports 'transparent proxying' or redirecting network traffic seamlessly.

Ipfwadm supported three basic rule-sets; input, output and forward. Every packet forwarded by the system traverses all three rule-sets. In addition to the three basic rule-sets, ipfwadm also supports a 'masquerade' rule-set for specifying NAT rules.

2.2 ipchains

In 1998 a number of changes were made to the Linux firewall code for kernel version 2.2. A new user-space tool was developed, called ipchains. The name 'ipchains' is commonly used to refer to the second incarnation of firewall support in Linux.

Firewall	Kernel Version	Packet Filtering	NAT	Connection tracking/ Stateful inspection	Application Intelligence	Integrated VPN Capability	Integrated QOS Marking
ipfwadm	1.0	X	X				
ipchains	2.2.0	X	X			X	X
netfilter/iptables	2.4.0	X	X	X	Partial	X	X
Firewall-1	2.4.0	X	X	X	X	X	X

Figure 1: Linux Firewalls by Version

ipchains is basically ipfwadm with a new face. It supports all of the features of ipfwadm, but adds the concept of chaining. In ipchains speak, a chain is a ordered list of rules that may be used as the target of a rule specification. Jumping to a user defined chain is roughly analogous to a programming language procedure call. User defined chains allow complex rule-sets to be expressed more simply, and traversed more efficiently.

ipchains also supports actions beyond the add/drop actions supported in ipfwadm. Using the masquerade and redirect actions NAT configurations can be selectively interspersed with firewall rules. The new actions allow more complex configuration than is possible with ipfwadm.

2.3 netfilter/iptables

In 1999 a major re-write of the underlying kernel hooks required for firewall features was performed, and the new infrastructure was named 'netfilter'. Since kernel version 2.4, filtering, NAT, and other packet handling extensions now use the netfilter interface. The user-space tool for configuring many of these packet handling extensions, (notably NAT, filtering and mangling) is named 'iptables'. People generally refer to the current incarnation of Linux firewall support as either netfilter or iptables, although they actually describe two distinct components.

The single most visible new feature of iptables is stateful firewall support. This means that the connection tracking subsystem will mark each packet as part of a NEW, ESTABLISHED, RELATED, or INVALID connection. Chain rules can match the associated connection state of the packet to dramatically improve upon the security of a conventional packet filter.

Iptables also provides a way to mark packets based on firewall rules. This marking can be used with other networking modules to provide Quality of Service (QOS) routing.

Finally, the versatile netfilter framework allows easy extension for individual situations. Many open source developers have taken advantage of this capability to create sophisticated extensions. Some currently available extensions include:

Match Extensions:
fuzzy - a fuzzy logic controller for adaptive rate limiting rules
ipv4options - match IP options in packet headers
psd - port scan detector and adaptive blocking
quota - impose byte limits on matched traffic
string - match arbitrary string anywhere in the packet
Target Extensions:
FTOS - rewrite the TOS field
IPV4OPTSTRIP - strip ipv4 options from the packet
XOR - encrypt packets using a simple XOR encryption
Connection Tracking Extensions:
H.323 - support h.323 (and netmeeting) connection tracking
MMS - support Microsoft Streaming Media Services connection tracking
amanda - support amanda backup tool protocol connection tracking

Also available for netfilter/iptables are GUI configuration builders and logfile analyzers. These products are relatively new however, and are currently receiving mixed reviews from the user community.

2.4 Firewall-1

In late 1999, Check Point Software Technologies Inc. announced it would port its popular Firewall-1 commercial firewall package to Linux. The Firewall-1 system is an enterprise scale system with slick GUI tools for log analysis and intrusion detection. Check Point is the 900 pound gorilla in the commercial firewall market, and the Linux implementation is a professional grade product.

In addition to the common filtering and connection tracking capabilities offered by other firewalls, Firewall-1 supports a technology that Check Point calls 'Application Intelligence'. This is basically fancy-talk for a stateful firewall that knows about upper level (layers 5-7) protocols. The engine for this upper level stateful inspection is programmable, using a custom language defined by Check Point. This allows the application specific stateful inspection capabilities to be updated without new software releases. What's the point? According to Check Point, Firewall-1 can block packets that would exploit known weaknesses in application programs, protecting the corporate network better than conventional stateful inspection.

Firewall-1 also includes advanced features that can detect attacks in progress and react to them. The software is able to detect SYN floods, frag attacks, and other common protocol exploits and lock down the sender(s) of such attacks without operator intervention.

2.5 Evaluation

Linux kernel firewall support has grown from a trivial packet filter into a robust and highly extensible framework for network security. For good protection on a budget, netfilter/iptables is an ideal choice. In a primarily UNIX shop the features of netfilter/iptables are more than adequate, but for protecting Microsoft Windows applications, Firewall-1's broad application support should not be overlooked. If the budget is available, the advanced features and ease of configuration make Firewall-1 a clear winner. For details on Firewall-1, refer to (http://www.checkpoint.com/).

3 netfilter

Of the firewall options listed in the previous section, only iptables and Firewall-1 are really appropriate for use in a corporate setting. Without stateful inspection it's simply too easy to circumvent the firewall. In this section, the netfilter side of the netfilter/iptables duo is examined. Although the following discussion is focused on the new netfilter architecture, many of the high level concepts are common to the previous Linux open-source firewall implementations.

3.1 Input Packets

When a packet arrives from a network interface a hardware interrupt queues the packet for processing. As the kernel schedules time for processing network traffic, this queue is serviced [1], and packets are forwarded to the appropriate protocol handler (IPv4 for most of us). [2] The IPv4 packet handler performs some basic verification of the packet and it is at this point, the **PREROUTING** hook that firewall code has its first opportunity to look at incoming packets. Figure 2 illustrates the relationship between the various kernel hooks. Assuming that our packet survives its first encounter with the firewall, IPv4 next determines whether this packet should be delivered locally or forwarded (hence the name *PREROUTING* for the first firewall hook).

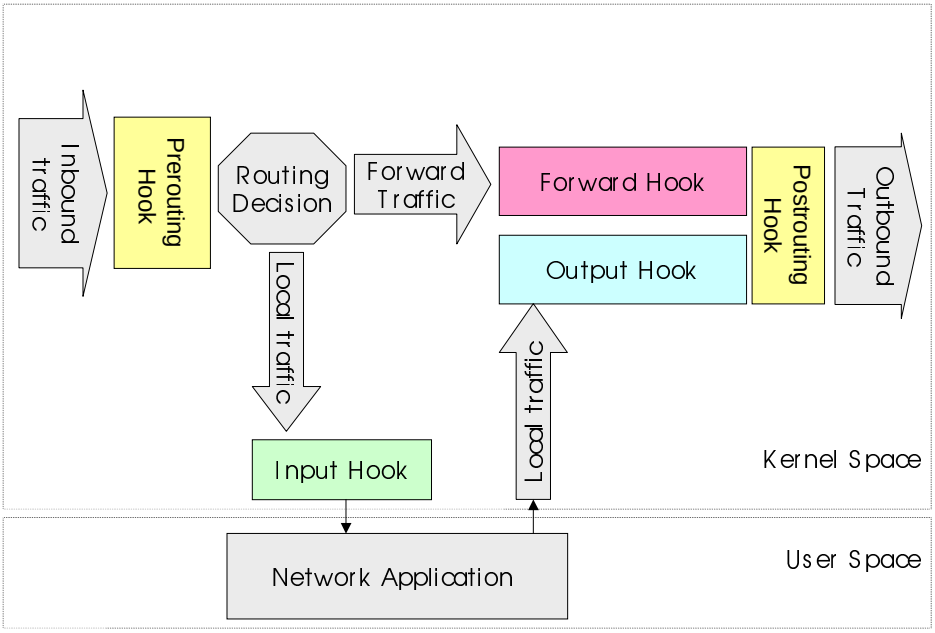

Figure 2: Linux Kernel Hooks for Firewalls

3.2 Locally Delivered Packets

In older kernels, packets destined for local delivery would be accepted once they passed the PREROUTING firewall hook. With netfilter, an additional **INPUT** hook has been added between the local delivery decision and actual delivery of the packet. After surviving the INPUT hook, packets are delivered locally.

This INPUT hook can be a source of confusion when moving between ipchains and iptables, because in ipchains, the PREROUTING hook was known as the *input* chain[3].

[1] The actual process for detecting data and servicing this queue depends on your kernel version.

[2] Actually, things are a bit more complicated when dealing with dialup links, because the link protocol (typically PPP) has its own set of work to perform before it can pass packets to/from IP.

[3] Actually, the preferred place to filter inbound packets in netfilter is the INPUT hook, not the PREROUTING hook, so when the new hook was added, the designers felt it would help encourage people to do their filtering in the 'correct' location if the new hook was named 'INPUT'

3.3 Forwarded Packets

When forwarding packets, (again IPv4 for most of us) appropriate routing operations (such as TTL decrement) are taken care of, and the **FORWARD** hook is traversed. If the packet makes it past the FORWARD hook, the remainder of the protocol work needed to forward a packet is performed and the packet is prepared for transmission.

3.4 Locally Originated Packets

Outbound packets are formatted with appropriate headers and prepared for transmission. Before passing the packet on for sending, netfilter requires packets to traverse the **OUTPUT** hook. Older kernel versions did not have this hook.

3.5 Sending Packets

Packets created on the local machine or being forwarded to another host must pass one final firewall hook before they are transmitted. Once packets traverse the **POSTROUTING** hook they are passed to the link level formatter (PPP, Ethernet, etc) for transmission. The POSTROUTING hook was known as the *output* chain in kernel versions prior to 2.4.[4]

3.6 Traversing a Netfilter Hook

Now let's take a look at what actually happens when a hook is traversed. When a hook is encountered, the kernel passes the packet information to the netfilter code. Netfilter then checks to see if any other kernel modules are 'listening' for this hook. If anyone is listening for this hook, it is given the opportunity to 'mangle' the packet — basically, change any of the packet's contents — and will return to netfilter instructions on what to do with the packet next.[5] The instructions supported are:

Accept - continues normal processing.
Drop - discards the packet, further processing is aborted.
Stolen - the returning listening module now owns the packet, further processing is aborted.
Queue - the packet should be queued for further processing.
Repeat - repeat the current hook.

Any protocol may implement hooks to netfilter, and once implemented, these hooks may be accessed through the netfilter interface. In previous incarnations, the hooks in each protocol needed to directly call each supported 'packet mangle'. The addition of the netfilter interface simplifies adding new extensions to packet handling and reduces the code change required in individual protocols when extensions are desired.

4 iptables

Now that we understand what the kernel does for us, lets take a look at how we can put that capability to use. We'll lead off with a simple example; a single machine connected to the Internet via a cable modem. The figure 3 illustrates the network topology of our first example.

4.1 Enabling the Firewall

Most major Linux distributions come with netfilter compiled into the kernel, and a copy of iptables. To ensure the kernel has the necessary components built and installed use the following command.

```
iptables -nL
```

[4]The motivation for the name change is congruent with that for the PREROUTING hook name change

[5]If multiple modules are listening for a hook they are processed according to each modules relative priority within the netfilter module

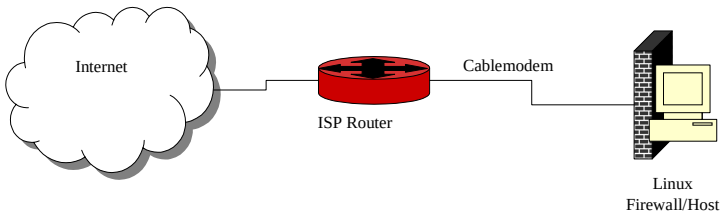

Figure 3: a home user network topology

If the command runs without error and lists the (probably empty) contents of the INPUT, OUTPUT, and FORWARD chains you'll know the kernel is ready to filter.

The next step to configuring the firewall is to install the relevant kernel modules. The modprobe command is used to install Linux kernel modules. In order to enable the connection tracking necessary for stateful firewall, we need the ip_conntrack module installed.

```
modprobe ip_conntrack
modprobe ip_conntrack_ftp
```

Whenever working on the firewall, it's a good idea to disable IP forwarding, so any transient state in the firewall rules won't open up an exploitable hole. In Linux, you can disable IP forwarding with the following command.

```
echo 0 > /proc/sys/net/ipv4/ip_forward
```

Once we have connection tracking enabled, the next step is to lock the firewall down tight. This may be accomplished by flushing any rules currently in the firewall and setting the default policy for each of the INPUT, OUTPUT, and FORWARD chains to 'DROP'. Note however that this operation should only be done from the console, as it will block ALL network traffic to/from/through the firewall.

```
# flush any existing rules
iptables -F
iptables --table nat --flush
iptables --delete-chain
iptables --table nat --delete-chain

# set the default policy to be DROP
```

```
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP
```

4.2 A First Rule-set

At this point our firewall is locked down tight. Nothing gets in, nothing gets out, nothing gets through; It's so secure it's useless! In fact, the loopback interface is even blocked...we can't so much as 'telnet localhost' The first order of business is to let the loopback interface work again. The loopback occurs at layer 2/1... way down below all our packet filters, that means in order to successfully loopback a packet, both the INPUT and OUTPUT hooks need to be traversed. The following commands will add rules to allow all loopback packets.

```
iptables -A INPUT -i lo -p all -j ACCEPT
iptables -A OUTPUT -o lo -p all -j ACCEPT
```

Now that our machine can talk to itself we probably want to be able to take advantage of our Internet connection. Assuming that our Cable modem link uses DHCP to give us an IP address, the next order of business is letting DHCP do its thing. DHCP uses UDP ports 67 and 68. We can use this information to allow DHCP traffic past the INPUT/OUTPUT hooks. The following commands will allow DHCP request/responses on the Internet interface (which I'm assuming from here on out is eth0).

```
iptables -A INPUT -p UDP -i eth0 --sport 67 --dport 68 -j ACCEPT
iptables -A OUTPUT -p UDP -o eth0 --dport 67 --sport 68 -j ACCEPT
```

Now lets enable outbound connections (such as web surfing) from the firewall. The following commands will enable connection tracking, and make sure that only outbound connections are allowed.

```
iptables -A INPUT -i eth0 -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A OUTPUT -o eth0 -m state --state NEW,ESTABLISHED,RELATED -j ACCEPT
```

Last but not least, we should log inbound traffic. We don't want to log every single dropped packet though, or some bad guy could fill up our log partition. We'll use the -m limit option with the default parameters to keep the average log rate to log 3 dropped packets per hour, with a burst rate of 5 packets.

```
iptables -A INPUT -m limit -j LOG
```

This is great! Now we can do pretty much whatever we like on the Internet, and nobody will be able to open any connections to our machine. Only problem is, if we reboot for any reason, all these nice firewall rules will be lost. We need to save these rules and set up a startup script to do this work for us every time the firewall starts. The following command may be used to dump the current rules to a file.

```
iptables-save > /etc/iptables.rules
```

To restore these rules at the next boot, add the following command to your network startup script in /etc/init.d.

Note: you want to enable the firewall BEFORE you bring up your network interfaces, so the network interfaces are protected from the outset.

```
iptables-restore < /etc/iptables.rules
```

This is good for the trivial cases, but for larger installations, administrators usually just write a shell script to call iptables directly to add/remove rules as appropriate at system startup/shutdown.

4.3 iptables Syntax

Now that we've seen a simple example, lets figure out what all those commands mean.

`iptables -P <chain> ACCEPT|DROP|...`

This command sets the default policy for a chain. Only the built in chains (INPUT, OUTPUT, and FORWARD) may have a default policy associated with them.

`iptables -A <chain> <rule-specification>`

Append *rule-specification* to *chain*

`iptables -D <chain> <rule-specification>`

Delete rule matching *rule-specification* from chain *chain*

`iptables -R <chain> <rule-number> <rule-specification>`

Replace rule *rule-number* in *chain* with *rule-specification*

`iptables -I <chain> <rule-number> <rule-specification>`

Insert *rule-specification* at position *rule-number* in *chain*. When the rule number is omitted, inserts to the beginning of the chain.

Rule specifications are created by combining one or more packet characteristics to match with a jump to a target action (such as ADD, DROP, REJECT, or a user-defined chain).

Some examples of what you can match in a packet include:

-p, –protocol [!] protocol
-s, –source [!] ip_address[/mask]
-d, –destination [!] ip_address[/mask]
-j, –jump target
-i, –in-interface [!] [name]
-o, –out-interface [!] [name]
[!] -f, –fragment
–source-port [!] [port[:port]]
–destination-port [!] [port[:port]]
–tcp-flags [!] mask comp
[!] –syn
–tcp-option [!] number
–icmp-type [!] typename
–mac-source [!] mac_address

There are many more options and capabilities than we have space to discuss here. For complete details on what is allowed see the iptables(8) man page.

4.4 A More Practical Example

Now that we've seen the basic form of the commands for configuring iptables, lets take on a more reasonable example. Suppose we have a small business or home office with a DSL link to the Internet and around 10 computers. See figure 4 for a diagram of the network. The firewall machine doubles as a web server and FTP server for the business.

In order to make this configuration work we need to do several things. First, we'll need to enable NAT to convert the private network addresses into port mapped addresses on the firewall, using the DHCP assigned address. Next we'll need to poke a couple of holes in the firewall to allow the web/FTP traffic through to the servers running on the firewall. Finally, we'll need to enable IP forwarding with connection tracking to allow the protected network to access the Internet. A System-V style startup script for this configuration is shown in figure 5.

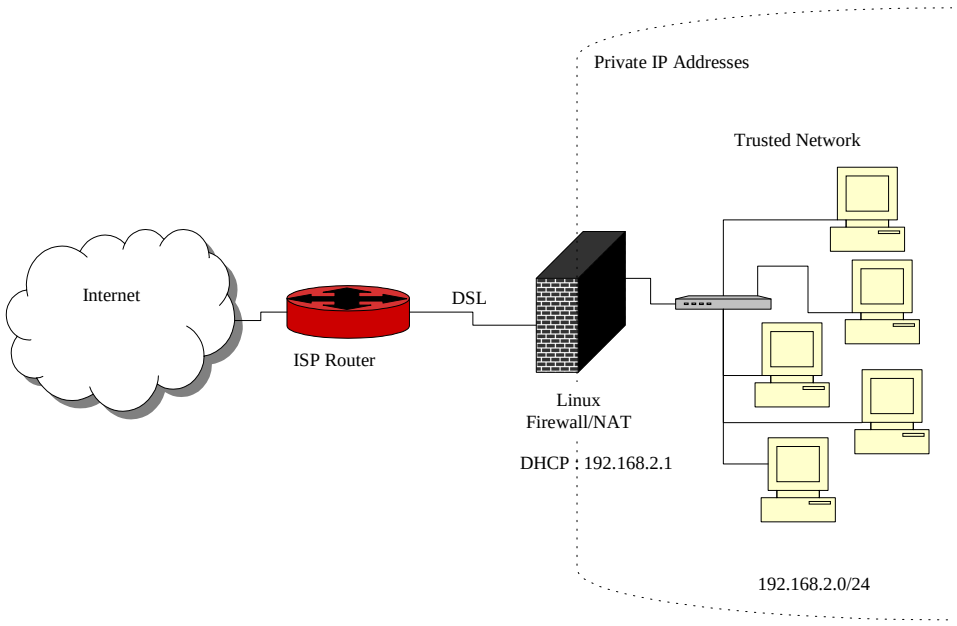

Figure 4: a small business network topology

5 Next Steps

We've gone through a couple of basic netfilter/iptables examples, but this is just the tip of the network security iceberg. The number one rule of firewall design is to be paranoid. Think about how the 'bad guy' will approach your system, and look for security holes. Conventional wisdom from the network security experts is to block all access points that are not absolutely necessary. For performance reasons you will also want to arrange rules so they match and drop a large percentage of your disallowed traffic at the beginning of the chain.

Once your firewall is in place, you'll want to use one of the portscan packages such as nmap (http://www.insecure.org/nmap) or nessus (http://www.nessus.org/) to check for 'leaks' in your network security. In addition to the actual firewall implementation, a good firewall will take advantage of other security packages to reduce exposure to risk. tcp_wrappers for example, is a good package for selectively limiting access to services your Linux machine provides.

Some other random security advice:

You should use ssh whenever you're working across an untrusted network.

Host security is the better part of network security.

Seek to minimize the potential for harm when a 'trusted' host is compromised.

Use a logfile analyzer to help spot intrusions.

Consider using an automatic intrusion detector such as snort (http://www.snort.org/).

Use syn_cookies for syn flood protection.

Run 'crack' type programs against your passwd files.

Keep everything on your system patched up to the latest security fixes.

```
#!/bin/sh

set -e
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin
PATH=$PATH:/usr/sbin:/usr/bin

INET_IF=eth0
LAN_IF=eth1

block_all()
{
#install kernel modules
modprobe ip_conntrack
modprobe ip_conntrack_ftp

# disable forwarding
echo 0 > /proc/sys/net/ipv4/ip_forward

# flush any existing rules
iptables -F
iptables --table nat --flush
iptables --delete-chain
iptables --table nat --delete-chain

# set the default policy to be DROP
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP
}



case "$1" in
  start)
# set firewall to initial (secure) state
block_all

# enable loopback interface
iptables -A INPUT -i lo -p all -j ACCEPT
iptables -A OUTPUT -o lo -p all -j ACCEPT

# enable dhcp on the internet interface
iptables -A INPUT -p UDP -i $INET_IF \
--sport 67 --dport 68 -j ACCEPT
iptables -A OUTPUT -p UDP -o $INET_IF \
--dport 67 --sport 68 -j ACCEPT



# allow forwarding packets for open connections
iptables -I FORWARD -m state \
--state INVALID -j DROP
iptables -I FORWARD -m state \
--state RELATED,ESTABLISHED -j ACCEPT

# allow outbound connections from LAN
iptables -A FORWARD --in-interface $LAN_IF \
-j ACCEPT

# set up NAT to maqwuerade protected network
iptables --table nat --append POSTROUTING \
--out-interface $INET_IF -j MASQUERADE

# allow web and ftp traffic to the firewall

iptables -A INPUT -i $INET_IF -p tcp \
--destination-port ftp -j ACCEPT
iptables -A INPUT -i $INET_IF -p tcp \
--destination-port ftp-data -j ACCEPT
iptables -A INPUT -i $INET_IF -p tcp \
--destination-port www -j ACCEPT

# allow all service related outbound traffic
iptables -A OUTPUT -o $INET_IF -p tcp \
--source-port ftp -j ACCEPT
iptables -A OUTPUT -o $INET_IF -p tcp \
--source-port ftp-data -j ACCEPT
iptables -A OUTPUT -o $INET_IF -p tcp \
--source-port www -j ACCEPT

# allow protected hosts full access to fw
iptables -A INPUT -i $LAN_IF -j ACCEPT
iptables -A OUTPUT -o $LAN_IF -j ACCEPT

#log dropped packets
iptables -A INPUT -m limit -j LOG


# enable forwarding
echo 1 > /proc/sys/net/ipv4/ip_forward

    ;;
  stop)
#return firewall to initial (secure) state
block_all
    ;;
  *)
    if test "$@"; then
      echo "Aborting; unknown command(s):"
    fi
    ;;
esac
```

Figure 5: iptables configuration for a small business

6 Conclusion

At this point we have examined the major tools for configuring a Linux firewall. For further reading on securing Linux systems I recommend the following resources:

Michael D. Bauer, *Building Secure Servers with Linux*, O'Reilly & Associates, October 2002 ,ISBN : 0-596-00217-3

http://www.linuxsecurity.com/ : in particular the Documentation section has pointers to most of the major Linux specific security documents

http://www.netfilter.org/ : the source for the latest netfilter/iptables firewall upgrades and patches

References

[1] Manfred Bartz. Netfilter log format. netfilter-log-format.php3, 2001.

[2] D. Coulson. iptables part 1. Linux Format, March 2003.

[3] D. Coulson. iptables part 2. Linux Format, June 2003.

[4] D. Coulson. Mastering iptables. Linux Format, May 2003.

[5] Fabrice Marie. Netfilter extensions howto. netfilter-extensions-HOWTO.html, 2002.

[6] J. (ed.) Postel. *User Datagram Protocol.* USC/Information Systems Institute, std006 edition, August 1980.

[7] J. (ed.) Postel. *Internet Protocol - DARPA Internet Program Protocol Specification.* USC/Information Systems Institute, std005 edition, September 1981.

[8] J. (ed.) Postel. *Transmission Control Protocol - DARPA Internet Program Protocol Specification.* USC/Information Systems Institute, std007 edition, September 1981.

[9] Randy Russell. Linux networking concepts howto. networking-concepts-HOWTO.txt, July 2001.

[10] Randy Russell. Linux 2.4 nat howto. NAT-HOWTO.html, January 2002.

[11] Randy Russell. Linux 2.4 packet filtering howto. packet-filtering-HOWTO.txt, January 2002.

[12] Randy Russell and Harald Welte. Linux netfilter hacking howto. netfilter-hacking-HOWTO.html, July 2002.

[13] Harald Welte. journey of a packet through the linux 2.4 network stack, the. packt-journey-2.4.html, October 2000.

[14] Harald Welte. The netfilter framework in linux 2.4. presentation.html, September 2000.

Addendum A: Firewall Concepts

A firewall is a device which filters traffic between an untrusted network and one or more other hosts or networks. There are four primary technologies used in implementing firewalls; packet filtering, connection tracking, proxying, and Network Address Translation (NAT). Firewalls are typically deployed at the boundary between an untrusted network and a trusted network. Firewalls allow the administrator to restrict network traffic, and to track network activity.

Packet Filtering

When doing packet filtering, the firewall inspects characteristics of the packet being filtered in order to enforce administrative policy. For example, if an administrator chooses to disallow all telnet connections then the packet filter could be configured to drop connections to/from TCP port 23. This will effectively prevent most telnet connections, although a determined user may circumvent this restriction by running an telnet server on a non-standard port. Packet filters typically only look at packet headers from layer 4 and below on the OSI model. Figure 6 contains a conceptual diagram of a packet filter's position in a network connection.

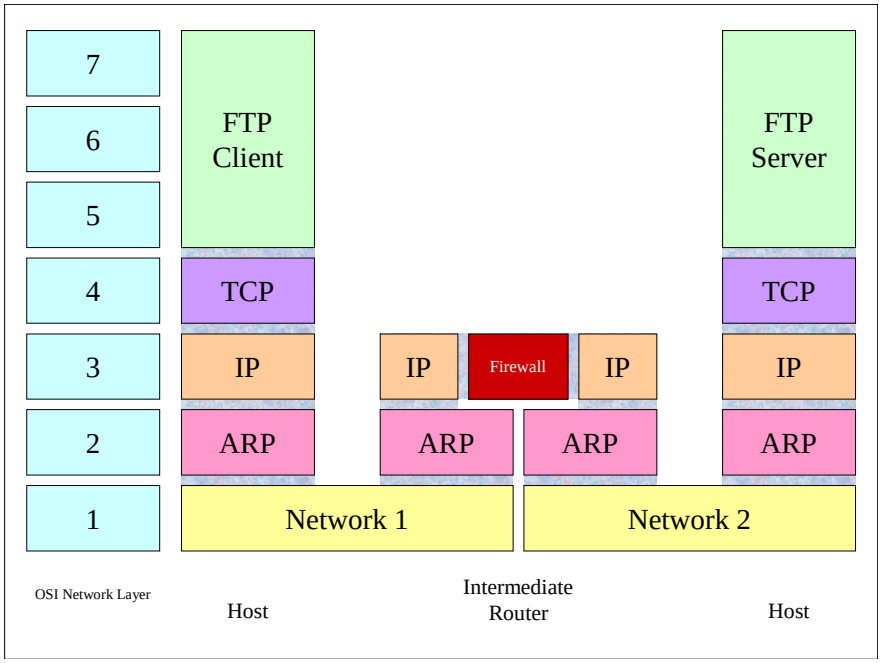

Figure 6: packet filter conceptual diagram

Connection Tracking

Connection tracking is the next logical step in filtering network traffic. Rather than filter using a single packet's information, connection tracking keeps a connection state table for all connections passing through the firewall. Information from the state table is available when making policy decisions about a packet. Connection tracking makes it possible to enforce one-sided permission on connections. For example, the policy may be to allow corporate LAN users to access the Internet freely, but to disallow any access to the corporate LAN by Internet users. With connection tracking a rule to block new connections from the Internet would enforce this policy. Figure 7 contains a conceptual diagram of a connection tracking firewall's position in a network connection.

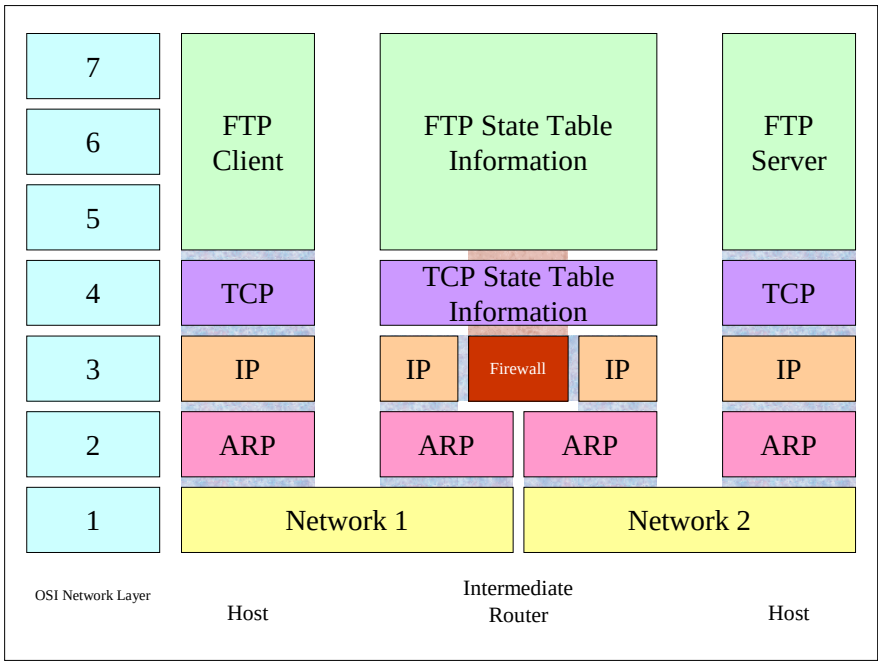

Figure 7: connection tracking conceptual diagram

Proxying

Proxying is not actually a firewall technology, but it is frequently used together with firewall to enforce strict security policy. Proxying involves the use of an application gateway, or 'proxy server' to insulate network applications between trusted and untrusted domains. Consider the policy of restricting all telnet access in/out of a network. We already know that packet filtering alone cannot accomplish this task. One possible solution is to establish proxy servers for all *allowed* services. The firewall can then be configured to *only* permit in/outbound connections to/from the proxy server. This scenario is typical when businesses wish to restrict their employee's activities on the Internet. Figure 8 shows one possible placement of a proxy server and firewall in a network connection.

Network Address Translation (NAT)

Network Address Translation is another technology that is not unique to firewall applications, but is frequently used in conjunction with a firewall. NAT translates source/destination addresses in packet headers (and application payloads where necessary) to map the translation domain addresses to global domain addresses.

The primary use of NAT is IP address conservation. By taking advantage of unused high numbered TCP and UDP port numbers on globally routable IP addresses many hosts in the translation domain can share a few (or one) Internet routable address.

For example a machine in the translation domain may have address 10.10.10.10. This address is not routable on the Internet; it is reserved for private networks. If we want this host to be able to access the Internet, we could either change it's IP address to an Internet routable address, or we could use NAT to let it share an Internet routable address with another machine. Leasing the use of Internet routable addresses costs money. NAT allows you to put more hosts on the Internet for less money.

NAT makes a good deal of of sense for use with firewalls because each new translated host connection

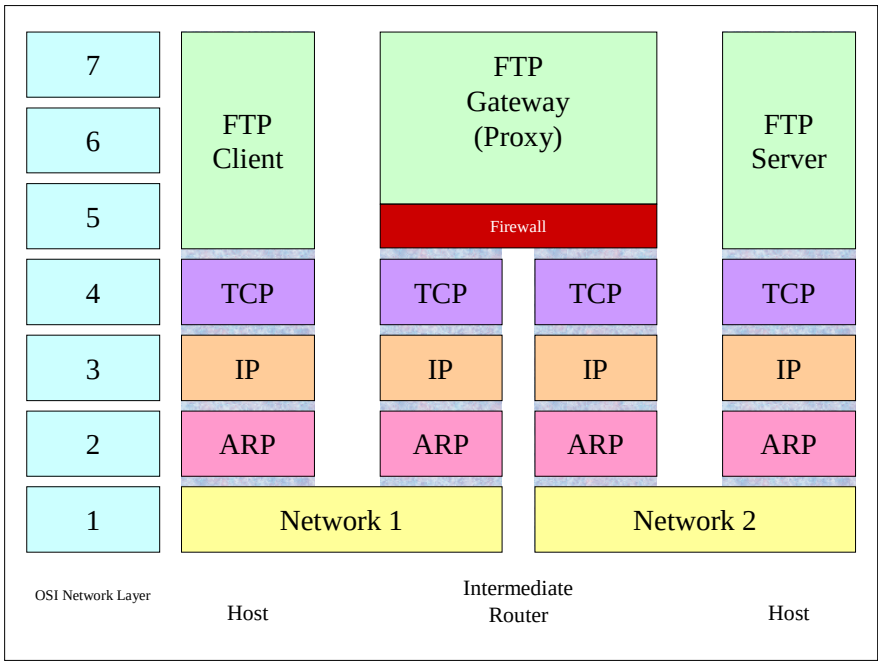

Figure 8: proxying conceptual diagram

keep gets a different high port number in the global address space. This makes it much more difficult for an outside attacker to reach translated hosts, because they are not directly addressable from the Internet.

Addendum B: IP Concepts

The Internet Protocol(IP) and other members of the IP family form the basis for most communication on the Internet. When discussing firewalls, we are typically discussing their application to IP networks. This addendum serves as a *brief* introduction to key concepts of IP networks necessary for a discussion of firewalls.

The first key concept for IP networks is the idea of network layers. The Open Systems Interconnection (OSI) model defines seven layers of network protocols—Physical, Data Link, Network, Transport, Session, Presentation, and Application—numbered 1 thru 7 respectively. The figure 9 shows some of the common Internet protocols and their corresponding OSI model layer.

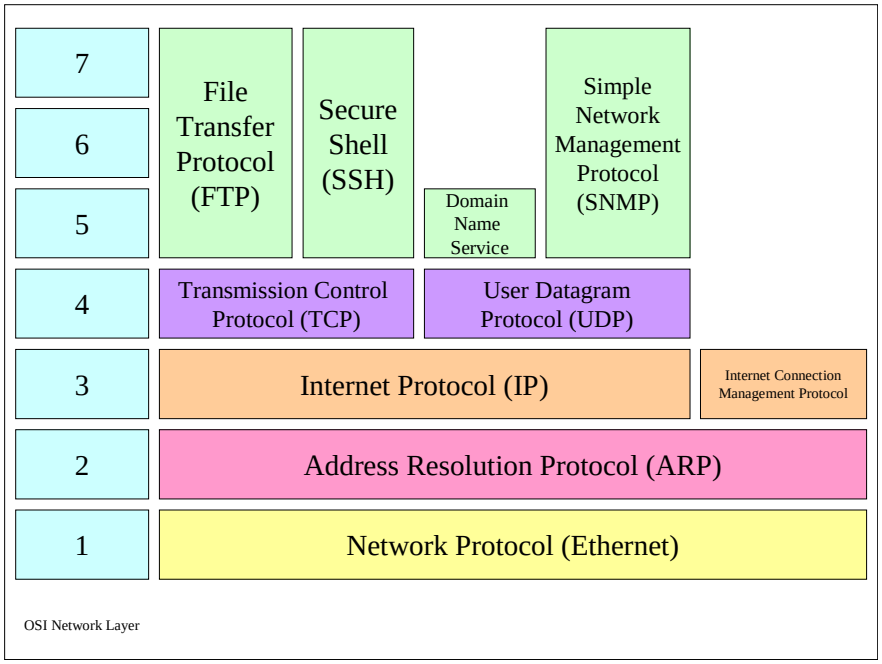

Figure 9: OSI Model layers for IP family protocols

Each layer of the OSI model is built upon the layer below. For example, the Transmission Control Protocol (TCP) requires some kind of layer 3 protocol—typically IP—below it to handle the general end to end routing of packets. TCP in turn is the basis for the File Transfer Protocol (FTP) and the Hyper Text Transport Protocol (HTTP) - the major protocols used to transfer information to your web browser. Most firewall activity is focused on layer 3 & 4 of the OSI model, but occasionally, information from the upper or lower layers is also considered.

The second key concept for IP network is connections. Whether we're talking about protocols based on a connection-oriented protocol such as TCP or a connectionless protocol like the User Datagram Protocol (UDP) we refer to the exchange of data between two machines as a connection. For example, while streaming media services typically use UDP based protocols for content delivery, an entire stream may be considered a single connection. This concept is an important key to how connection tracking is implemented in modern firewalls.

The third and final key concept for IP networks is routing. In IP networks, packets move from client to server or peer to peer by hopping from router to router. Each router reads the information from the IP *header* on the packet and makes a decision about where to forward the packet next. The figure 10 shows the typical header values for a TCP/IP packet.

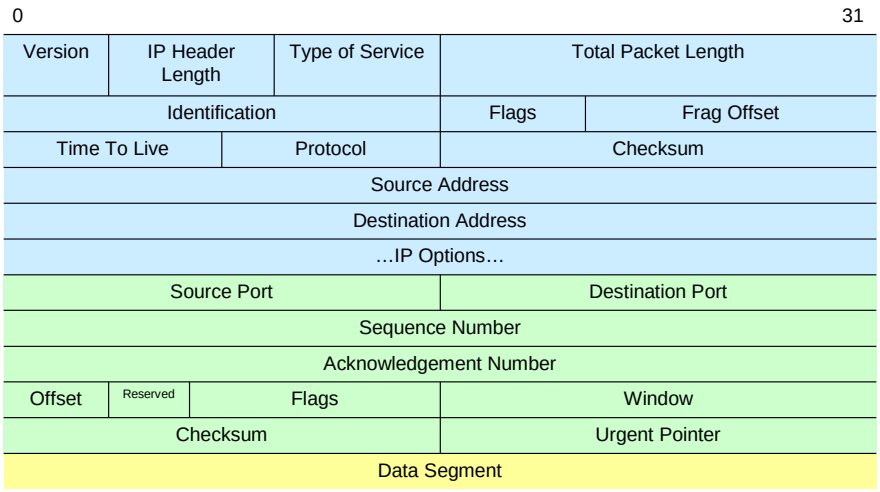

Figure 10: typical headers for TCP/IP packet

The intermediate router typically does not consider information beyond the layer 3 header to make a routing decision (although routers performing Quality of Service policy routing may examine layer 4 headers). This means that strictly speaking, intermediate routers typically need not implement upper layer protocols in order to perform IP routing. This is the model upon which the entire Internet backbone is based. Figure 11 shows a conceptual model for IP routing.

I should also briefly mentions what happens when QOS policy routing is involved. Policy routing means that instead of routing strictly on the basis of destination IP address, packets which match some criteria are given network priority or alternate routing. One example of this would be to priority queue Voice Over IP (VOIP) traffic ahead of FTP traffic, to preserve the responsiveness of the telephone call being carried over the VOIP connection. This all connects with firewalls because firewall packet filters can be used to mark packets with a QOS or Differentiated Services (DiffServ) classification that the QOS routing engine uses for the actual routing decision.

This introduction has only scratched the surface of IP network concepts, but should provide the reader with enough information to grasp the associated discussion of firewalls. Readers desiring a thorough introduction to Internet protocols may wish to read:

Douglas E. Comer, *Internetworking with TCP/IP. Volume 1: Principles, Protocols, and Architecture*, Prentice Hall: Upper Saddle River, NJ. 1995. ISBN 0-13-216987-8

For those who want to get into the dirty details, the website of the Internet Engineering Task Force (http://www.ietf.org) has detailed standards information for most of the protocols used on the Internet

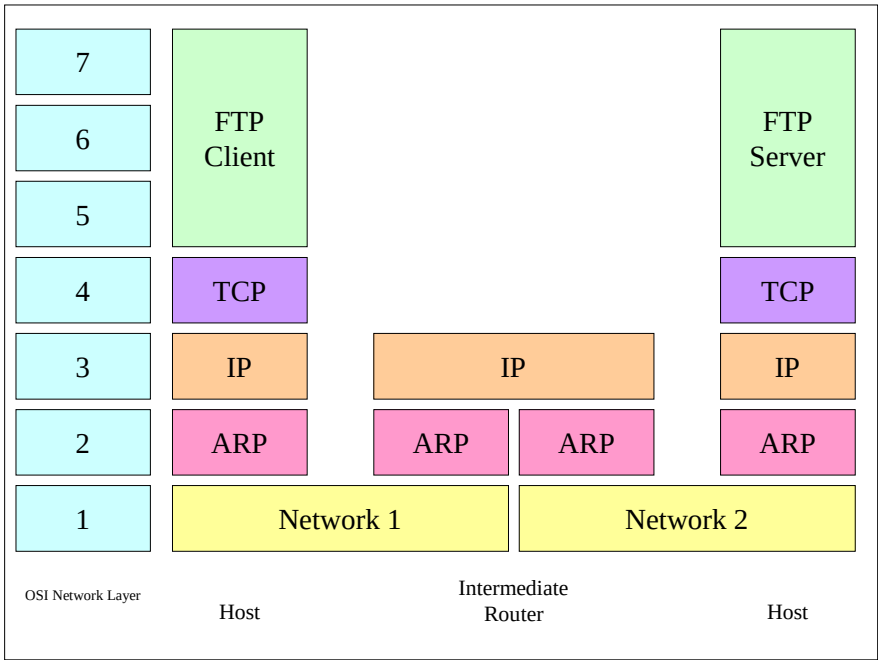

Figure 11: conceptual model of IP routing