UNIX/Linux-based firewalls

Why use UNIX/Linux?

Firstly, unless a network administrator is familiar with the UNIX/Linux platform, then it is not recommended that a UNIX/Linux firewall is implemented. It will be easier to maintain a secure system if the administrator has skills in maintaining the underlying operating system. UNIX/Linux systems typically rely on a CLI more than a GUI, making navigating the system more difficult for the first time user.

Modern Linux kernels (from versions 2.4 and above) include NetFilter/IPTables which filters incoming, outgoing and forwarded traffic. Earlier Linux kernels provided IPChains (2.2) or the ported IPFirewall code (2.0).

UNIX/Linux systems can be configured easily with a low system footprint, making a small, clean install with the minimum configured services. Creating a system with a small footprint makes it easier to maintain and more secure, as operating system updates are only required for packages that are installed.

When building Linux systems there is the opportunity to build a custom kernel. The ability to create a custom monolithic kernel has advantages and disadvantages. The standard kernel included with distributions has a number of options enabled and others disabled and extensions can be included as loadable modules.

Creating a custom kernel provides the ability to incorporate just the elements of the kernel that are required for the operation of a firewall, which includes networking drivers and enhancements like SELinux, GrSecurity, SMP and Magic SysRq.

SELinux and GRSecurity are extensions to the Linux operating system which provide local security enhancements. These are particularly useful on multi-user interactive systems and can also provide protection against buffer overflow attacks and similar exploits.

SMP extensions to the kernel provide support for multiprocessor systems. Magic SysRq supplies a number of operations which can be accessed using the SysRq key in the event of a system crash. These operations can also be achieved using a serial line connection allowing out-of-band crash recovery, disc sync and reboot.

The main disadvantages in creating tailored kernels are the inability to use the vendorsupplied kernel updates and the limited support provided on most contracts. It is essential to weigh up the pros and cons of both solutions.

Requirements

As the firewall is being built on a PC-based hardware system, it is important to consider the hardware configuration required to run it. Linux distributions will run on most hardware vendor platforms and therefore compatibility with major hardware should not be an issue. The hardware required will be dependent on the following factors:

- wirespeed
- complexity of rules
- number of interfaces
- scalability
- data retention
- policy and support.

For a small organisation with a 10Mbit/s internet connection and a default deny firewall policy, implementing a simple two interface routed NAT firewall storing data for 14 days can be built on a very small footprint, often an old desktop PC.

Larger organisations with 100Mbit/s internet connections, more complex firewall rules, more interfaces and longer data storage will require more powerful platforms.

The firewall is one of the most critical pieces of hardware in the organisation’s network infrastructure, so ensuring a reliable platform which is supported by maintenance is paramount. Most hardware vendors have grades of support from within working hours to 24 hours a day, seven days a week. Support can usually be purchased for three years at the time of procurement and then extended for a further year at a time.

Hardware

When purchasing new hardware, it is important not to procure a machine that is underspecified and will cripple the firewall. It is always worth using a server grade motherboard for a production server, along with a rack mount chassis with dual power supplies.

Memory is not a critical requirement: a firewall servicing 100Mbit/s wirespeed typically uses no more than 512Mb of RAM, although as memory is relatively inexpensive, it is worth purchasing 1Gb as a minimum.

CPU speed is also not a critical requirement: the same firewall will run with a small load average using a twin Pentium® Xeon 2.8Ghz processor. Dual processors are not an absolute requirement, although it helps reduce any lag on a heavily-loaded firewall.

It is recommended that good quality branded network interfaces are used, and are installed with a view to future expansion and dedicated out-of-band management. A common installation is Intel® PRO (S) interfaces, with the PCI-X variants in server boards.

Disc space is required for booting the firewall itself and log file storage. This can be achieved with a RAID mirrored pair of system discs and the remaining discs used for log file storage.

Distribution

As with the operating system on which a firewall is based, the choice of distribution is just as critical and should be dependent on the operator's familiarity with it.

Most modern Linux distributions such as Fedora, SUSE® and Ubuntu will use an implementation of NetFilter/IPTables. NetFilter is the firewalling code in the Linux kernel that provides a packet filter firewall, NAT, Connection Tracking and other features through kernel modules. Sample scripts for building the firewall are included with the distributions and further clues are often found in the **rc directory**. Fedora Linux includes the lokkit tool for easy configuration from the command line or from the GNOME desktop.

OpenBSD distributions use the pf packet filter included in the operating system kernel to provide firewall functionality, while FreeBSD® and Mac OS X use the ipfw packet filter. The latter system is very similar in operation to NetFilter/IPTables.

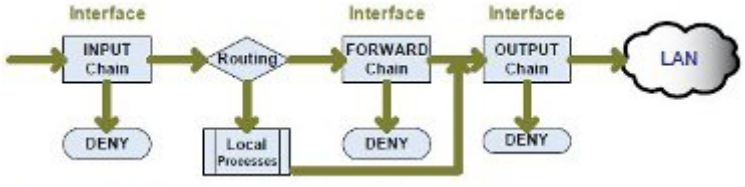
Solaris 10, NetBSD and earlier versions of FreeBSD use the ipf packet filter, which is installed by the operating system. This can provide similar firewall services as a loadable kernel module or be included when compiling a fresh kernel.

Firewall software

The choice of firewall code will probably be determined by the preferred operating system and distribution. All the different firewall systems look very similar on the surface, but they are subtly different underneath. IPChains is not supported by most modern distributions so is only mentioned here in passing. The rest of this section will concentrate on NetFilter/ IPTables and demonstrate how it differs from ipfw. IPTables is the name of the tool which creates the rules to be managed by the overall NetFilter firewall code.

The ipfw code supports three filtering chains: INPUT, FORWARD and OUTPUT. All packets are processed by the INPUT chain, then if accepted, routed to the local computer or forwarded.

Packets forwarded are subjected to the FORWARD chain and finally all packets, including those locally generated, are processed by the OUTPUT chain. Therefore, an ipfw firewall will process forwarded packets against three chains.

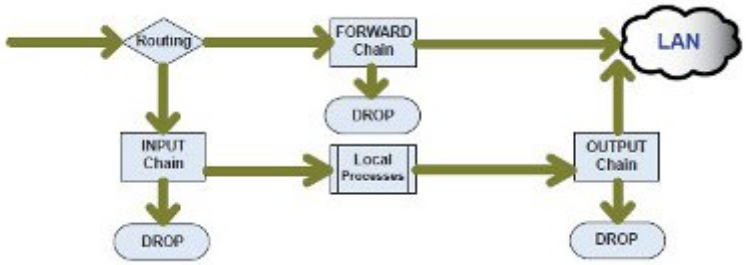


[1]

Figure 13: How ipfw processes a packet through the firewall

NetFilter/IPTables initially uses three chains: INPUT, FORWARD and OUTPUT. All packets are firstly interpreted by the routing element.

If a packet is to be forwarded, it is only processed by the FORWARD chain, before being directed straight out of the appropriate interface. If a packet is destined for the local computer or locally generated then it is processed by the INPUT and OUTPUT chains respectively.



[2]

Figure 14: How NetFilter processes a packet through the firewall

There are many small differences between IPTables and previous firewall code. IPTables is built of many modules. For example, the INPUT and OUTPUT interfaces are identified separately. Translation functions are separated from packet filtering. The logging option is now a rule target and NAT is separate from the packet filtering element.

NetFilter firewalls are constructed using the very powerful IPTables tool which controls the creation and management of all the elements of the firewall.

There are three tables of multiple chains within NetFilter:

- FILTER
- NAT
- MANGLE

The FILTER table is the most commonly used and by default holds the chains FORWARD, INPUT and OUTPUT. The NAT table provides Network Address Translation functions and the MANGLE table is used to alter packets as they are inspected by the firewall code.

The syntax and order of IPTable commands are very strict and follow a standard layout:

iptables <option> <chain> <matching criteria> <target>

The most commonly used **option** is –A to add a rule after the last currently active rule in a chain.

The **chain** entry will be the default INPUT, OUTPUT or FORWARD in the filter table, a user-defined chain, or one contained in the NAT or MANGLE tables.

Matching criteria is the statement which identifies the packets to be acted upon. This could be by source, destination, port number or other supported methods.

Finally the **target** is where the packet is destined, whether it is dropped, logged, allowed or manipulated further.

Implementing a NetFilter/IPTables firewall

Initialising the firewall

The easiest way to create a firewall is from a shell script that is executed when the computer running the firewall starts. On a Fedora or SUSE system, the convention would be to place the script in **/etc/rc.d/** named appropriately as **rc.firewall** or **rc.netfilter**.

The firewall script should be owned and executable by root only. The firewall script can then be started from **rc.local** each time the computer boots.

It is possible to use variables in the shell script for the local network and for specific networks for services. However, the use of variables is down to personal choice. An example of a variable would be:

LOCAL=”192.168.1.0/24”

WEBSERVER=”192.168.1.80”

As with any shell script, it starts by calling the shell in which to execute the script:

#!/bin/sh

As a security measure to avoid starting the computer with no firewall rules, the network configuration for the firewall can be included in a script, which in turn loads the firewall rules.

For a Linux bridging firewall with two interfaces, this script would configure the network before loading the rules.

#!/bin/sh

echo Bringing interfaces up...

ifconfig eth0 up

ifconfig eth1 up

echo Sleeping...

sleep 10

echo Bringing bridging up...

brctl addbr br0

brctl addif br0 eth0

brctl addif br0 eth1

brctl stp br0 off

echo Sleeping...

sleep 10

echo Adding bridge IP address and default route

ifconfig br0 up

ifconfig br0 inet 192.168.1.2 netmask 255.255.255.0 broadcast

192.168.1.255

echo Adding firewall rules

/etc/rc.d/rc.firewall_rules

Customised rulesets

Once a basic firewall script has been created, there are a number of specific services which may still need to be blocked. It is impossible to provide an exhaustive list, but as a starting point some are listed below:

- Net BIOS/SMB/CIFS – TCP/UDP 135-139 and TCP 445
- Microsoft® RPC over HTTP – TCP/UDP 593
- BootP/DHCP – TCP/UDP 67 and 68
- SNMP – TCP/UDP 161and 162
- NFS – TCP/UDP 2049
- Microsoft® SQL – TCP 1433
- MySQL – TCP 3307
- Microsoft® RDP – TCP 3389
- LPD – TCP 515
- TFTP – UDP 69
- Simple UNIX Services (TIME, CHARGEN etc)

Many web sites offer advice on what should be blocked at the firewall, but it is important to tailor the rules to the software and services running at your organisation. When using a particular piece of network-based backup software, it would be prudent to ensure that it is blocked from off-campus access.

When defining the rules, it is essential to specifically allow services which are running on your network which require Internet access. The most important thing is to make the rule as specific as possible. For example, when creating a rule to allow SMTP access to Message

Transfer Agents on your internal network, do not allow port 25 access to the entire network, but ensure it is only allowed to specific hosts.

For example, do not use:

iptables –A FORWARD –d 192.168.1.0/24 --dport 25 –j ACCEPT

Instead, use these rules for the primary and secondary mail routers:

iptables –A FORWARD –d 192.168.1.25 --dport 25 –j ACCEPT

iptables –A FORWARD –d 192.168.1.26 --dport 25 –j ACCEPT

iptables –A FORWARD –d 192.168.1.0/25 --dport 25 –j DENY

The third line, which explicitly denies port 25 traffic to the rest of the netblock, is very useful when first creating firewall rules and also when debugging. Explicit deny rules like this will be shown on screen by the iptables **–L FORWARD –v** command along with hit counters to show how many times each rule has been matched.

When optimising rules for a production environment, if there is a explicit DENY for all traffic at the end of the rule set, some DENY rules may be removed. However, rules can only be removed if there will not be an accidental match further down the rule set.

Multicast

Multicast traffic is often overlooked when setting firewall policies. Multicast is enabled by most RNOs and is an integral tool for many collaboration projects. Internally, multicast is used when deploying computer images using packages like Norton Ghost™, and for locating resources with Service Location Protocol and Rendezvous/Bonjour on Mac OS X. It is advisable to ensure that appropriate rules are in place to protect multicast traffic, as an example of creating a user-defined rule. Multicast traffic can be identified and then examined in further detail in the MCAST chain.

iptables -F MCAST

iptables -X MCAST

iptables -N MCAST

iptables -A FORWARD -i br0 -d 224.0.0.0/4 -j MCAST

 # NTP

iptables -A MCAST -d 224.0.1.1 -j DROP

 # SGI IRIX objectserver/directoryserver

iptables -A MCAST -d 224.0.1.2 -j DROP

 # rwhod

iptables -A MCAST -d 224.0.1.3 -j DROP

 # Service Location Protocol

iptables -A MCAST -d 224.0.1.22 -j DROP

 # Microsoft Active Directory server

iptables -A MCAST -d 224.0.1.24 -j DROP

 # Service Location Protocol Directory Agent

iptables -A MCAST -d 224.0.1.35 -j DROP

 # Cisco’s PIM Auto Rendezvous Point

iptables -A MCAST -d 224.0.1.39 -j DROP

iptables -A MCAST -d 224.0.1.40 -j DROP

 # HP Device Discovery

iptables -A MCAST -d 224.0.1.60 -j DROP

 # Sun RPC

iptables -A MCAST -d 224.0.2.2 -j DROP

 # Altiris Ghosting / Multicast Usenet

iptables -A MCAST -d 225.1.2.3 -j DROP

 # Norton Ghosting

iptables -A MCAST -d 229.55.150.208 -j DROP

 # Source Specific Multicast (SSM) groups

iptables -A MCAST -s 233.80.58.0/24 -j ACCEPT

iptables -A MCAST -d 233.80.58.0/24 -j ACCEPT

iptables -A MCAST -d 232.0.0.0/8 -j DROP

 # ImageCast ghosting

iptables -A MCAST -d 234.42.42.42 -j DROP

 # ImageCast ghosting

iptables -A MCAST -d 234.142.142.142 -j DROP

 # Locally (admin) scoped groups

iptables -A MCAST -d 239.0.0.0/8 -j DROP

Logging

By default, logging from the NetFilter/IPTables firewall code will be logged as **priority 4 kern.warn** messages and written to **/var/log/messages**. The priority can be changed with the **--log-level option**. The creation of the messages is controlled by the syslog process.

Syslog works on a series of facilities and the priority of alerts sent from them. The facilities are categories of event which allow easier diagnostics and log file analysis. The priority of alerts is set by the programmer for the software in question. The alerts are used to determine if and where the messages should be logged. This is all configured in **/etc/syslog.conf**. Firewall logging can be redirected to a new file by adding a line to **/etc/syslog.conf**, but it is important to make sure an empty file exists and other log management software like log rotate is updated.

kern.warn/var/log/firewall

Note: In some older UNIX/Linux distributions the white space in the **/etc/syslog.conf** file requires tab characters instead of simple spaces.

Links
[1] http://community.ja.net/system/files/images/firewalls-tg-13.jpg
[2] http://community.ja.net/system/files/images/firewalls-tg-14.jpg