NAME

firewall - simple firewalls under FreeBSD

FIREWALL BASICS

A Firewall is most commonly used to protect an internal network from an outside network by preventing the outside network from making arbitrary connections into the internal network. Firewalls are also used to prevent outside entities from spoofing internal IP addresses and to isolate services such as NFS or SMBFS (Windows file sharing) within LAN segments.

The FreeBSD firewalling system also has the capability to limit bandwidth using dummynet(4). This feature can be useful when you need to guarantee a certain amount of bandwidth for a critical purpose. For example, if you are doing video conferencing over the Internet via your office T1 (1.5 MBits/s), you may wish to bandwidth-limit all other T1 traffic to 1 MBit/s in order to reserve at least 0.5 MBits for your video conferencing connections. Similarly if you are running a popular web or ftp site from a colocation facility you might want to limit bandwidth to prevent excessive bandwidth charges from your provider.

Finally, FreeBSD firewalls may be used to divert packets or change the next-hop address for packets to help route them to the correct destination. Packet diversion is most often used to support NAT (network address translation), which allows an internal network using a private IP space to make connections to the outside for browsing or other purposes.

Constructing a firewall may appear to be trivial, but most people get them wrong. The most common mistake is to create an exclusive firewall rather than an inclusive firewall. An exclusive firewall allows all packets through except for those matching a set of rules. An inclusive firewall allows only packets matching the ruleset through. Inclusive firewalls are much, much safer than exclusive firewalls but a tad more difficult to build properly. The second most common mistake is to blackhole everything except the particular port you want to let through. TCP/IP needs to be able to get certain types of ICMP errors to function properly - for example, to implement MTU discovery. Also, a number of common system daemons make reverse connections to the **auth** service in an attempt to authenticate the user making a connection. Auth is rather dangerous but the proper implementation is to return a TCP reset for the connection attempt rather than simply blackholing the packet. We cover these and other quirks involved with constructing a firewall in the sample firewall section below.

IPFW KERNEL CONFIGURATION

You do not need to create a custom kernel to use the IP firewalling features. If you enable firewalling in your */etc/rc.conf* (see below), the ipfw kernel module will be loaded automatically when necessary. However, if you are paranoid you can compile IPFW directly into the FreeBSD kernel by using the **IPFIREWALL** option set. If compiled in the kernel, ipfw denies all packets by default, which means that, if you do not load in a permissive ruleset via */etc/rc.conf*, rebooting into your new kernel will take

the network offline. This can prevent you from being able to access your system if you are not sitting at the console. It is also quite common to update a kernel to a new release and reboot before updating the binaries. This can result in an incompatibility between the ipfw(8) program and the kernel which prevents it from running in the boot sequence, also resulting in an inaccessible machine. Because of these problems the **IPFIREWALL_DEFAULT_TO_ACCEPT** kernel option is also available which changes the default firewall to pass through all packets. Note, however, that using this option may open a small window of opportunity during booting where your firewall passes all packets. Still, it is a good option to use while getting up to speed with FreeBSD firewalling. Get rid of it once you understand how it all works to close the loophole, though. There is a third option called **IPDIVERT** which allows you to use the firewall to divert packets to a user program and is necessary if you wish to use natd(8) to give private internal networks access to the outside world. If you want to be able to limit the bandwidth used by certain types of traffic, the **DUMMYNET** option must be used to enable *ipfw pipe* rules.

SAMPLE IPFW-BASED FIREWALL

Here is an example ipfw-based firewall taken from a machine with three interface cards. fxp0 is connected to the 'exposed' LAN. Machines on this LAN are dual-homed with both internal 10. IP addresses and Internet-routed IP addresses. In our example, 192.100.5.x represents the Internet-routed IP block while 10.x.x.x represents the internal networks. While it is not relevant to the example, 10.0.1.x is assigned as the internal address block for the LAN on fxp0, 10.0.2.x for the LAN on fxp1, and 10.0.3.x for the LAN on fxp2.

In this example we want to isolate all three LANs from the Internet as well as isolate them from each other, and we want to give all internal addresses access to the Internet through a NAT gateway running on this machine. To make the NAT gateway work, the firewall machine is given two Internet-exposed addresses on fxp0 in addition to an internal 10. address on fxp0: one exposed address (not shown) represents the machine's official address, and the second exposed address (192.100.5.5 in our example) represents the NAT gateway rendezvous IP. We make the example more complex by giving the machines on the exposed LAN internal 10.0.0.x addresses as well as exposed addresses. The idea here is that you can bind internal services to internal addresses even on exposed machines and still protect those services from the Internet. The only services you run on exposed IP addresses would be the ones you wish to expose to the Internet.

It is important to note that the 10.0.0.x network in our example is not protected by our firewall. You must make sure that your Internet router protects this network from outside spoofing. Also, in our example, we pretty much give the exposed hosts free reign on our internal network when operating services through internal IP addresses (10.0.0.x). This is somewhat of security risk: what if an exposed host is compromised? To remove the risk and force everything coming in via LAN0 to go through the firewall, remove rules 01010 and 01011.

Finally, note that the use of internal addresses represents a big piece of our firewall protection

mechanism. With proper spoofing safeguards in place, nothing outside can directly access an internal (LAN1 or LAN2) host.

```
# /etc/rc.conf
#
firewall_enable="YES"
firewall_type="/etc/ipfw.conf"

# temporary port binding range let
# through the firewall.
#
# NOTE: heavily loaded services running through the firewall may require
# a larger port range for local-size binding.  4000-10000 or 4000-30000
# might be a better choice.
ip_portrange_first=4000
ip_portrange_last=5000
...

# /etc/ipfw.conf
#
# FIREWALL: the firewall machine / nat gateway
# LAN0      10.0.0.X and 192.100.5.X (dual homed)
# LAN1      10.0.1.X
# LAN2      10.0.2.X
# sw:       ethernet switch (unmanaged)
#
# 192.100.5.x represents IP addresses exposed to the Internet
# (i.e. Internet routeable).  10.x.x.x represent internal IPs
# (not exposed)
#
#   [LAN1]
#     ^
#     |
#   FIREWALL -->[LAN2]
#     |
#   [LAN0]
#     |
#     +--> exposed host A
#     +--> exposed host B
#     +--> exposed host C
```

```
#     |
#   INTERNET (secondary firewall)
#    ROUTER
#     |
#    [Internet]
#
# NOT SHOWN:  The INTERNET ROUTER must contain rules to disallow
# all packets with source IP addresses in the 10. block in order
# to protect the dual-homed 10.0.0.x block.  Exposed hosts are
# not otherwise protected in this example - they should only bind
# exposed services to exposed IPs but can safely bind internal
# services to internal IPs.
#
# The NAT gateway works by taking packets sent from internal
# IP addresses to external IP addresses and routing them to natd, which
# is listening on port 8668.   This is handled by rule 00300.  Data coming
# back to natd from the outside world must also be routed to natd using
# rule 00301.  To make the example interesting, we note that we do
# NOT have to run internal requests to exposed hosts through natd
# (rule 00290) because those exposed hosts know about our
# 10. network.  This can reduce the load on natd.  Also note that we
# of course do not have to route internal<->internal traffic through
# natd since those hosts know how to route our 10. internal network.
# The natd command we run from /etc/rc.local is shown below.  See
# also the in-kernel version of natd, ipnat.
#
#             natd -s -u -a 208.161.114.67
#
#
add 00290 skipto 1000 ip from 10.0.0.0/8 to 192.100.5.0/24
add 00300 divert 8668 ip from 10.0.0.0/8 to not 10.0.0.0/8
add 00301 divert 8668 ip from not 10.0.0.0/8 to 192.100.5.5

# Short cut the rules to avoid running high bandwidths through
# the entire rule set.  Allow established tcp connections through,
# and shortcut all outgoing packets under the assumption that
# we need only firewall incoming packets.
#
# Allowing established tcp connections through creates a small
# hole but may be necessary to avoid overloading your firewall.
```

```
# If you are worried, you can move the rule to after the spoof
# checks.
#
add 01000 allow tcp from any to any established
add 01001 allow all from any to any out via fxp0
add 01001 allow all from any to any out via fxp1
add 01001 allow all from any to any out via fxp2

# Spoof protection.  This depends on how well you trust your
# internal networks.  Packets received via fxp1 MUST come from
# 10.0.1.x.  Packets received via fxp2 MUST come from 10.0.2.x.
# Packets received via fxp0 cannot come from the LAN1 or LAN2
# blocks.  We cannot protect 10.0.0.x here, the Internet router
# must do that for us.
#
add 01500 deny all from not 10.0.1.0/24 in via fxp1
add 01500 deny all from not 10.0.2.0/24 in via fxp2
add 01501 deny all from 10.0.1.0/24 in via fxp0
add 01501 deny all from 10.0.2.0/24 in via fxp0

# In this example rule set there are no restrictions between
# internal hosts, even those on the exposed LAN (as long as
# they use an internal IP address).  This represents a
# potential security hole (what if an exposed host is
# compromised?).  If you want full restrictions to apply
# between the three LANs, firewalling them off from each
# other for added security, remove these two rules.
#
# If you want to isolate LAN1 and LAN2, but still want
# to give exposed hosts free reign with each other, get
# rid of rule 01010 and keep rule 01011.
#
# (commented out, uncomment for less restrictive firewall)
#add 01010 allow all from 10.0.0.0/8 to 10.0.0.0/8
#add 01011 allow all from 192.100.5.0/24 to 192.100.5.0/24
#

# SPECIFIC SERVICES ALLOWED FROM SPECIFIC LANS
#
# If using a more restrictive firewall, allow specific LANs
```

```
# access to specific services running on the firewall itself.
# In this case we assume LAN1 needs access to filesharing running
# on the firewall.  If using a less restrictive firewall
# (allowing rule 01010), you do not need these rules.
#
add 01012 allow tcp from 10.0.1.0/8 to 10.0.1.1 139
add 01012 allow udp from 10.0.1.0/8 to 10.0.1.1 137,138

# GENERAL SERVICES ALLOWED TO CROSS INTERNAL AND EXPOSED LANS
#
# We allow specific UDP services through: DNS lookups, ntalk, and ntp.
# Note that internal services are protected by virtue of having
# spoof-proof internal IP addresses (10. net), so these rules
# really only apply to services bound to exposed IPs.  We have
# to allow UDP fragments or larger fragmented UDP packets will
# not survive the firewall.
#
# If we want to expose high-numbered temporary service ports
# for things like DNS lookup responses we can use a port range,
# in this example 4000-65535, and we set to /etc/rc.conf variables
# on all exposed machines to make sure they bind temporary ports
# to the exposed port range (see rc.conf example above)
#
add 02000 allow udp from any to any 4000-65535,domain,ntalk,ntp
add 02500 allow udp from any to any frag

# Allow similar services for TCP.  Again, these only apply to
# services bound to exposed addresses.  NOTE: we allow 'auth'
# through but do not actually run an identd server on any exposed
# port.  This allows the machine being authed to respond with a
# TCP RESET.  Throwing the packet away would result in delays
# when connecting to remote services that do reverse ident lookups.
#
# Note that we do not allow tcp fragments through, and that we do
# not allow fragments in general (except for UDP fragments).  We
# expect the TCP mtu discovery protocol to work properly so there
# should be no TCP fragments.
#
add 03000 allow tcp from any to any http,https
add 03000 allow tcp from any to any 4000-65535,ssh,smtp,domain,ntalk
```

```
add 03000 allow tcp from any to any auth,pop3,ftp,ftp-data

# It is important to allow certain ICMP types through, here is a list
# of general ICMP types.  Note that it is important to let ICMP type 3
# through.
#
#       0       Echo Reply
#       3       Destination Unreachable (used by TCP MTU discovery, aka
#                                               packet-too-big)
#       4       Source Quench (typically not allowed)
#       5       Redirect (typically not allowed - can be dangerous!)
#       8       Echo
#       11      Time Exceeded
#       12      Parameter Problem
#       13      Timestamp
#       14      Timestamp Reply
#
# Sometimes people need to allow ICMP REDIRECT packets, which is
# type 5, but if you allow it make sure that your Internet router
# disallows it.

add 04000 allow icmp from any to any icmptypes 0,3,8,11,12,13,14

# log any remaining fragments that get through.  Might be useful,
# otherwise do not bother.  Have a final deny rule as a safety to
# guarantee that your firewall is inclusive no matter how the kernel
# is configured.
#
add 05000 deny log ip from any to any frag
add 06000 deny all from any to any
```

PORT BINDING INTERNAL AND EXTERNAL SERVICES

We have mentioned multi-homing hosts and binding services to internal or external addresses but we have not really explained it. When you have a host with multiple IP addresses assigned to it, you can bind services run on that host to specific IPs or interfaces rather than all IPs. Take the firewall machine for example: with three interfaces and two exposed IP addresses on one of those interfaces, the firewall machine is known by 5 different IP addresses (10.0.0.1, 10.0.1.1, 10.0.2.1, 192.100.5.5, and say 192.100.5.1). If the firewall is providing file sharing services to the windows LAN segment (say it is LAN1), you can use samba's 'bind interfaces' directive to specifically bind it to just the LAN1 IP address. That way the file sharing services will not be made available to other LAN segments. The

same goes for NFS. If LAN2 has your UNIX engineering workstations, you can tell nfsd to bind specifically to 10.0.2.1. You can specify how to bind virtually every service on the machine and you can use a light jail(8) to indirectly bind services that do not otherwise give you the option.

SEE ALSO

dummynet(4), ipnat(5), rc.conf(5), smb.conf(5) (*ports/net/samba*), samba(7) (*ports/net/samba*), config(8), ipfw(8), ipnat(8), jail(8), natd(8), nfsd(8)

ADDITIONAL READING

Ipfilter

ipf(5), ipf(8), ipfstat(8)

Packet Filter

pf.conf(5), pfctl(8), pflogd(8)

HISTORY

The **firewall** manual page was originally written by Matthew Dillon and first appeared in FreeBSD 4.3, May 2001.