Firewalling with OpenBSD and PF

Written by Kevin Korb

September 5, 2007

1 What is a firewall?

A firewall is a router that has rules defining what kinds of packets are routed and what kinds aren't. It is a device that sits between two different networks (often a private LAN and the Internet) and controls what traffic passes between the two. The firewall is often a dedicated device that provides added security by keeping bad things from getting through. It is of course much more complicated than that but that is the basic idea. The most common use for a firewall is to keep people on the outside of the network from connecting to services that are only intended for internal users.

1.1 What is stateful inspection?

Stateful Inspection is a function of more advanced firewalls that allows the rule set to be more strict while reducing the hassle on users and administrators. The way it works is that instead of firewalling individual packets you firewall complete connections. The rules in a stateful firewall only apply to the initial packet of a connection (like a TCP SYN packet). The rule then creates a state within the firewall for that connection with either a pass or block flag set. Then when further packets come in they are matched to the existing state and follow the same pass or block action. This also means that if an internal IP initiates an outgoing connection which is passed then the incoming packets which are part of the same connection will also be passed even if an externally initiated connection on the same port would be blocked.

2 Why would you need or want a firewall?

2.1 Security

A firewall allows you to prevent incoming (or outgoing) connections to specific services or services that aren't the specific services you want to allow. This is done at a protocol level using port numbers. A properly setup firewall can prevent people on the outside from connecting to services that you do not want to make public or perhaps don't even know are running. If the service can't be contacted then it can't be hacked into either.

2.2 Logging

A firewall can be configured to log the connections that it blocks. This data can then be used to analyze what is going on and adapt as needed. It can also be used as evidence if there is ever a need to prosecute someone.

2.3 NAT

NAT is actually a function of a router not a firewall however since the two are related a firewall often provides NAT functionality. NAT (Network Address Translation) is a method of connecting two networks together while causing the outside network to think that it is talking to a single host. This allows the internal network to use non-routable IP addresses instead of being required to acquire real IP addresses.

3 What does a firewall NOT protect you against?

3.1 Insiders

Any user or device that is on the inside of the firewall is not burdened by it. They don't have to get through the firewall to attack the internal systems. This is primarily a concern for corporate networks where there are employees on the inside that can't be completely trusted however there are also concerns with wireless networks allowing people on the outside act as someone on the inside.

3.2 Allowed protocols

Since the firewall operates on the protocol/port level it can't defend against improper uses of an allowed protocol. A primary example of this would be web exploits. If you allow incoming traffic on TCP port 80 (www) then you also allow all of the various scripting exploit attempts in. There are ways to mitigate this such as an integrated IDS or a filtering proxy however these go beyond the scope of a firewall (though they may be tied to it).

3.3 Attacks against the firewall itself

If there is a vulnerability in the firewall itself the firewall could be compromised and then used as an attack launching platform within your network. OpenBSD's built-in security makes this unlikely however it is possible to open up vulnerabilities by accident (such as providing a service on the firewall system).

4 Why PF/OpenBSD?

4.1 Security.

OpenBSD is a very secure opperating system. It is designed to be secure by default meaning there is no hardening process needed before putting the system on the Internet. Security patches are rare but prompt. OpenBSD also has integrated strong cryptography enabling you to easily use advanced security technologies such as S/KEY authentication. It is also possible to freeze the firewall rules so that they can't be changed without rebooting the system which virtually guarantees that the firewall can't be altered without notice even by someone who manages to gain root access.

4.2 PF tables.

PF supports a system called tables which is really a hashed listing of IP addresses and CIDR net blocks. Searching a table for an IP address is very fast. Because of the hashing algorithm it is almost as fast to search through a list of 50,000 IP addresses as it is with a list of only 5 (though the memory requirements are significantly different). If you are defending against a DDoS attack or similar situation with many different addresses that need to be placed into firewall rules a PF table is the way to go. Tables can be read in from a text file, manipulated with command line tools, or can even be the result of other firewall rules (such as someone making too many connections to your ssh port). This offers great flexibility and scripting capabilities.

4.3 PFLOG

Each firewall rule can be configured to log any packets that match that rule. These logs are actually the complete packet which is copied to a virtual network interface called pflog0. You can then use any packet sniffing or even an IDS program on this virtual interface to inspect only the packets which were specifically logged by PF. There is also a modification to tcpdump to add PF specific information to this logged data such as block/pass and which rule number the packet matched. The command to use this feature is 'tcpdump -n -e -ttt -i pflog0'.

5 Why not Linux/iptables?

5.1 Configuration file based

PF uses a text configuration file (/etc/pf.conf by default) for its configuration. Iptables uses a command line system to push rules into kernel space one at a time. Linux has a way to store the current rule set into a file and then load it back in later (usually after a reboot) however it is designed to be maintained from the iptables command not the configuration file.

5.2 Human readable configuration

The PF configuration file is much easier to read than the iptables save/restore file. With iptables the save/restore file is really just a shell script with the iptables command removed from each line. The pf.conf file uses a very simple syntax that is similar to Cisco's IOS though it is actually even easier.

5.3 Redundancy

PF also includes a system called CARP that provides redundancy across multiple systems. This allows you to setup two firewall systems with the same configuration and network connections that will fail over when needed. The running states and IP addresses will shift to the backup system if the primary fails.

5.4 Performance

Because of the use of tables PF can handle much longer firewall rule sets. This can make a huge difference on a very busy link or if you are a target of a DDoS attack.

6 Why not an appliance firewall/router?

6.1 Flexibility

Because you are running on a fully featured BSD UNIX system you have much greater flexibility beyond simple routing and firewalling. For instance it is possible to write your own scripts or programs to manipulate the firewall to achieve greater automation. It is also possible to install and run intrusion detection systems such as SNORT directly on the firewall.

6.2 Performance

Because OpenBSD runs on many different kinds of computers (not just PCs) it is possible to build a system that can handle almost any amount of traffic. The system can also be upgraded piece by piece as needed by future growth.

6.3 Cost

6.3.1 For home and small business users

For a typical residential or small business Internet connection (such as DSL or cable modem) OpenBSD's hardware requirements will be very small. It will likely work with an old system that is no longer useful for other tasks. My personal OpenBSD firewall that sits in front of my cable modem is an old AMD K6-2 500MHz system with 512MB of RAM (the RAM is more important than the CPU speed). It is more than adequate for my connection which is very heavily used almost 24 hours a day.

6.3.2 For enterprise users

An enterprise level company would likely need a more powerful system. Actually, they would probably want two complete systems setup redundantly. Even with lots of traffic passing through the network PF doesn't need much CPU power. Pretty much any system you get with enough bus speed to handle multiple high speed network interfaces will have more CPU power than you need.

7 Why not a host based firewall?

7.1 Centralized security

A dedicated firewall is much easier to maintain because there is generally only one (or a matched pair) of them. The firewall is installed at an obvious choke point between networks making it a gate keeper. If you use host based firewalling instead you have to maintain rule sets on all your systems some of which will probably have much less flexible firewalling systems than PF. A central firewall also protects devices such as networked printers which often have no firewalling capabilities at all.

7.2 Overhead

Inspecting packets adds latency. If each system on a LAN is doing its own firewalling then each one has added latency. In general there is no need to filter packets that stay within the local LAN so there is no need to add such latency. A little bit of latency added to Internet packets on the other hand will probably not even be noticed.

7.3 Security

Host based firewall products are often the targets of trojans and worms. An infected PC may be running without a firewall even though one is installed.

8 Differences from Linux/iptables

8.1 Configuration file vs. command line

As discussed earlier PF uses a configuration file (/etc/pf.conf) instead of a command line constructed firewall rule set with the option to save and restore.

8.2 Last matching rule vs. first matching rule

In Linux with iptables the first rule that matches a packet is the rule that affects the packet. If no rule matches the packet then the default policy is used. In PF the last rule that matches is the one that affects the packet. This means that you can make a general rule and then follow it up with more exact rules that act as exceptions to the earlier rule. There is also an optional “quick” parameter that forces a rule to be the last rule evaluated whenever it is matched.

8.3 No default policy rule

In Linux with iptables you configure a default policy for each “chain”. In PF there is no such policy. You have to make a rule for every possible case. Of course since you can start with the most general rule and narrow it down from there the obvious first rule is ’block all’ which simply blocks all packets in both directions unless there is a later rule saying otherwise.

8.4 Syntax

Because of the fact that PF uses a text configuration file instead of a command line interface the entries are formatted using keywords instead of parameters such as -A and -j. There is no need for command line style notation with PF which makes things much more readable.

9 Activating PF on an OpenBSD system

1. Set pf=YES in /etc/rc.conf
2. Set net.inet.ip.forwarding=1 in /etc/sysctl.conf

3. If you are adding enough entries into tables that you need to reserve more memory for them you will have to modify /etc/rc. In /etc/rc there is a block of code that loads a basic default PF rule set into a variable called $RULES. You need to add one to the beginning to reserve your additional memory.
4. This is to setup the system to load PF at boot time so at this point you should reboot.

10 Variables

A simple shortcut that can be used when making firewall rules is to use variables and substitution. The syntax for this is simply:

variable = “value”

The variables are used as $variable within rules. Variables can contain things like host names, interface names, IP addresses, CIDR blocks, or ranges. It is a common practice to use $int_if and $ext_if for the internal and external network interfaces. This also helps if you need to ask a question people understand what a rule with $ext_if in it means but if you used the real interface which might be something like dc1 you would have to post more than that one rule for it to make sense to anyone that doesn’t know your setup.

11 Ranges and groupings

PF supports several ways to group multiple entries or ranges of consecutive entries together in a single rule. For instance if you wanted a rule that affects both ssh and telnet you could use several notations:

- {22, 23} # ports 22 and 23
- 22:23 # ports 22 through 23
- 21><24 # ports between but not including 21 and 24

12 Options

There are options that can be set that control the way PF works. These are settings for things such as timeouts that are not packet specific so they are separate from the rules. The syntax is:

set [option] [value]

Common options to set are:

- optimization - This sets all of the state timeouts to a set of preset values.
- limit table-entries - This defines how much memory is reserved for table entries. If you need to have more than a few thousand entries in a table you will need to increase this setting.
- debug - Level of debugging output

13 Scrub rules

The scrub rules configure PF to normalize packets in various ways. This allows for things such as packet defragmentation and port number randomization (which helps prevent people from guessing random TCP sequence numbers which aren’t so random when generated by some operating systems). A good starting point for scrub rules is:

scrub in all

scrub out on $ext_if all random-id

This will scrub all incoming packets with the standard settings and will re-randomize all packets leaving through the external interface (to the Internet).

14 Tables

As discussed earlier tables are lists of IP addresses or CIDR blocks. They can be read in from a text file, left empty for use with the pfctl command, or populated directly within the pf.conf file. The syntax is:

> table <tablename> [persist | const] [file “filename”] [range]

NOTE that the <> chars around the table name are part of the rule not a document notation. The persist parameter tells PF to leave the table in memory even if there are no rules pointing to it. The const parameter tells PF that the table is read-only after it is defined within the pf.conf file. This is for cases where you don’t want the table to be modifiable by the pfctl command. Tables are accessed within rules by using <tablename> including the angle brackets.

15 NAT

NAT is usually setup with just a single rule. The syntax is:

> nat on $ext_if from $lan to any -> $ext_if

On my system $lan is set to 192.168.100.0/24. This sets up NAT between the private LAN and the outside Internet. Note that if you have multiple real addresses it is possible to NAT using all of them by replacing the $ext_if after the -> to an IP range. This allows better support for protocols that require listeners (FTP and ICQ) by mapping an external IP to an internal IP.

16 Port forwarding

Port forwarding allows you to take a connection that is destined for one location and send it to a different one. The most common usage is to forward connections to a service located within the private network behind a NAT. Another common usage is to deal with a service moving to a different port or IP. The syntax is:

> rdr on [interface] proto [proto] from [source] to [destination] -> [new destination]

example:

> rdr on $ext_if proto tcp from any to $ext_if port 22 -> 192.168.100.202 port 22

The example is a simple redirect rule to forward ssh connections to an internal server instead of the firewall system. Note that a rdr rule does not also add any access rules. There must be a pass rule somewhere that allows the traffic to pass.

17 Access control rules

Access control rules either pass or block packets that match a given rule. There are also many parameters to control how the system passes or blocks the packet. The basic format is:

> pass | block [return] [in | out] [quick] [on interface] [proto] [source] [dest] [flags] [state options] [limits] [queue options]

The portions of that command are:

- pass | block - Defines the rule as a pass or a block rule. One of the two must be specified and one other parameter must be present to specify what you are passing or blocking (even if it is just “all”).
- [return] - An optional parameter to define the block behavior. If return is present on a block rule the connection is refused using the standard mechanism for whichever protocol is in use (ICMP unreachable or TCP reset). Without the return parameter the blocked connections are simply dropped.
- in | out - Specifies that the rule only applies to packets that are entering or leaving the system.

- quick - Specifies that if a packet matches this rule no further rules are evaluated.
- on interface - Specifies that this rule only applies to a specific interface (for instance “on $ext_if”).
- proto - Specifies that this rule only applies to a specific protocol (this is the layer 4 protocol such as TCP, UDP, or ICMP). Note that in some cases you may want a rule to apply to both TCP and UDP and can specify that with “{ tcp, udp }” instead of making two rules.
- source and dest - These are the source and destination. They can be host names, CIDR blocks, ranges, interface names, or tables. They may or may not have port numbers as well. They can also simply be “any”.
- flags - This allows you to specify that the rule only applies to packets with specific TCP flags. This of course only works on rules that also contain “proto tcp”. The default as of OpenBSD 4.1 is “flags S/SA” which means SYN but not SYN+ACK packets. This is important because these are the packets that initiate a new TCP connection (start of a handshake).
- state options - These are options you can set that control how PF retains the state of a running TCP connection. This is how you can make a rule that only matches the initial packet of a connection but treats the subsequent packets the same way. The default under OpenBSD 4.1 is “keep state”. Other options are “modulate state” which adds more randomization to the TCP ISNs and “synproxy state” in which PF acts as a proxy server and does separate TCP handshakes on both sides of the firewall. The advantage of synproxy state is that it removes the possibility of a SYN flood attack however the downside is that you get strange results instead of an error if the destination is down.
- limits - This allows you to set limits such as only allowing a certain number of matching connections from any single IP address. This is how you can prevent (or at least limit) things like ssh brute force password attacks.
- queue options - This allows you to specify that packets matching this rule are to be queued in a specific queue for QoS purposes.

Here are some examples of varying complexity...

```
block all
```

Blocks all connections.

```
pass out all
```

Passes all outgoing connections.

```
block in quick log on $ext_if from { 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16, 255.255.255.255/32,
127.0.0.0/8 } to any
```

Blocks all connections coming in from the Internet claiming to be from non-routable IP blocks. Logs the attempts and doesn’t go any farther in the rules.

```
block return out quick log on $ext_if from any to { 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16, 255.255.255.255/32,
127.0.0.0/8 }
```

Blocks all connections attempting to go out to the Internet destined to non-routable IP blocks. Unlike the incoming rule above this one returns an error to the source (I want to an error when I do something stupid not a long timeout).

```
pass out on $ext_if proto tcp from any to any port 22 queue (high, ssh)
```

Allows outgoing ssh connections (they were already allowed) and queues them. The packets that are marked high priority (interactive sessions) go into the queue named “ssh” while the other packets (such as ssh’s TCP port forwarding go into a queue named “high” which on my system is a lower priority queue.

```
pass out on $ext_if proto tcp from any to any port 25 queue mail
```

Like the above rule except that it puts email connections into a queue called “mail”

> pass in on $ext_if proto tcp from any to any port 22 synproxy state (max-src-conn 20, max-src-conn-rate 10/5, overload <badguys> flush global) queue (high, ssh)

This is like the ssh connection rule above except that it is for incoming connections. Like the above rule it sorts the packets into the high or ssh queue depending on what is in the packet. It uses PF’s TCP proxy system to prevent SYN flood attacks. It also uses connection limits to prevent brute force password guessing attacks. If any single IP address makes more than 20 simultaneous connections or attempts 10 connections within 5 seconds the IP is added to a table called “badguys” and all connections that are already open are flushed (disconnected).

> block in log on $ext_if from <badguys> to any

Blocks all traffic coming in from any IP address that is listed in the “badguys” table and logs the attempt. These two rules together mean that anyone who trips my ssh connection limits has their running connections broken and any further attempts are blocked. The entries for “badguys” are expired elsewhere.

> pass in on $ext_if proto tcp from any to any port {80, 81, 8080, 8888} synproxy state queue web_serv

Passes traffic to my web server on several different ports. Uses the TCP proxy and places the connections into the “web_serv” queue.

18 Prioritizing (queuing for QoS)

OpenBSD also uses a queuing system called ALTQ that is used to prioritize packets so that some connections that require certain amounts of bandwidth can have what they need even on a link that is otherwise busy. It is important to note that this only applies to outgoing packets as you have no control over the order in which packets are received. There are three different queuing schemes that are supported by PF:

- priq - With priq each queue has a priority number. Whenever there is an opportunity to send a packet the one in the queue with the highest priority is the first to go out while the lower priority packets sit and wait. The risk is that if there is a flood of high priority packets the low priority stuff will just stall and eventually timeout. On the other hand this is the easiest scheme to setup as you don’t have to know how much bandwidth is needed by each protocol.
- cbq - With cbq each queue is defined an amount of bandwidth. This allows you to dedicate an amount of bandwidth to a queue for things like VOIP which require a constant flow. You can also allow one queue to borrow bandwidth from another queue if it isn’t being used.
- hsfc - This is sort of a hybrid scheme that has both bandwidth allocations like cbq and priority assignment like priq. This is also the only queuing scheme that supports link sharing and guaranteed real-time services.

Here is my priq setup as an example:

```
altq on $ext_if priq bandwidth 320Kb queue \
{ voip, ssh, high, other, usenet, web_serv, mail, p2p, spam }
queue voip priority 14
queue ssh priority 13
queue high priority 8
queue other priority 6 priq(default)
queue usenet priority 5 priq(red)
queue web_serv priority 4
queue mail priority 3 priq(red)
queue p2p priority 2 priq(red)
queue spam priority 0 priq(red)
```

This sets up my queues in order of my priority. VOIP traffic has highest priority because it requires a constant stream of data or the sound quality becomes horrible. SSH traffic is next because I hate lag time when I type things on remote shell connections (especially when in vi). The high queue is for things that I wanted to be at a higher than normal priority but less than the really high priority stuff. The other queue is the default queue for any connections that haven't been assigned to a queue. The web_serv queue is for when my web server uploads web pages to visitors. The mail queue is for when I send email to someone. The p2p queue is for when one of my p2p clients uploads (or seeds) data. The spam queue is for SMTP connections that are redirected into my spamd tar pit (hopefully they are all spammers). The "priq(red)" setting is a bandwidth reduction setting that essentially tells PF that the packets in this queue can simply be dropped if the queue gets too long.

19 Anchors

Anchors are a method of separating out chunks of rules similar to the way that tables separate out lists of IP addresses. This allows you to store sets of rules in other text files or manipulate them from the command line.

20 The pfctl command

The pfctl command controls all aspects of PF. It has many different uses and options to go with them. The most common usage is 'pfctl -f /etc/pf.conf' which simply reloads the entire PF configuration from the standard configuration file. This command is also used to manipulate tables and rules. If you use a limited firewall rule like my above ssh rule with the badguys table you would probably want to expire old entries from that file with a cron job like "pfctl -t badguys -T expire 86400" which expires any entry that hasn't been active for more than 86400 seconds (24 hours). Other common pfctl options are -s [modifier] which shows the various things stored within the PF system and -k [IP] which kills open sessions originating from an IP address.

21 Redundancy

PF also has a system of redundancy between multiple OpenBSD systems. This system consists of the CARP protocol and the pfsync interface. This allows you to setup two OpenBSD servers connected to the same networks and have one take over the firewall rules and even active states if the other fails. This is similar to Cisco's VRRP but of course it is free and unencumbered.

22 AuthPF

AuthPF is a system that ties ssh logins to PF anchors and tables. It is possible to configure PF so that whenever certain users log into the shell via ssh their IP is added to a different set of firewall rules probably to give them greater access to the network.

23 Examples

23.1 Minimal setup

```
# setup variables
$int="dc0"
$ext="dc1"
$lan="192.168.100.1/24"
# scrub all incoming packets with the default settings
scrub in all
# scrub outgoing packets with more randomness
scrub out on $ext all random-id
# allow the internal LAN to reach the Internet via NAT
```

```
nat on $ext from $lan to any -> $ext
# prevent most IP spoofing tricks from working
antispoof log quick for $ext inet
# block all traffic unless I say otherwise
block log all
# block all incoming IPv6 traffic because it isn't very useful yet
block in quick inet6
# allow all outbound connections
pass out all
# allow all internal connections
pass in on {$int, lo0} all
# allow incoming external ssh connections
pass in on $ext proto tcp from any to ext port 22
```

23.2 More advanced setup

```
$int="dc0" # internal interface is dc0
$ext="dc1" # external interface is dc1
$lan="192.168.100.1/24"
$server="192.168.100.202"
scrub in all
scrub out on $ext all random-id
# setup some basic QoS
altq on $ext priq bandwidth 320Kb queue {high, other, low}
queue high priority 8
queue other priority 5 priq(default)
queue low priority 2 priq(red)
nat on $ext from $lan to any -> $ext
# redirect these ports to the internal server so they are reachable from the outside
rdr on $ext proto tcp from any to $ext port {22, 25, 80} -> $server port {22, 25, 80}
rdr on $ext proto udp from any to $ext port 53 -> $server port 53
antispoof log quick for $ext inet
block log all
block in quick inet6
# prioritize TCP ACKs on all connections. This will speed up large transfers on async
# Internet connections (like cable and ADSL)
pass out on $ext all queue (other, high)
pass in on { $int, lo0 } all queue (other, high)
# this allows non-standard packet flags and options which are used by tools like nmap
pass out on { $int lo0 } all allow-opts
# block connections from non-routable IPs
block in quick log from no-route to any
# allow all ICMP traffic other than redirects which can be used for man-in-the-middle attacks
pass in on $ext inet proto icmp all
block in log on $ext inet proto icmp all icmp-type redir
# prioritize the interactive portion of ssh connections
pass in on $ext proto tcp from any to $server port 22 queue (other, high)
pass in on $int proto tcp from any to any port 22 queue (other, high)
# allow other incoming services
pass in on $ext proto tcp from any to $server port 80
pass in on $ext proto tcp from any to $server port 25 queue low
pass in on $ext proto udp from any to $server port 53
```

23.3 Even more advanced setup

```
$int="dc0"
$ext="dc1"
$lan="192.168.100.1/24"
$server="192.168.100.202"
scrub in all
scrub out on $ext all random-id
altq on $ext priq bandwidth 320Kb queue {high, other, low}
queue high priority 8
queue other priority 5 priq(default)
queue low priority 2 priq(red)
nat on $ext from $lan to any -> $ext
rdr on $ext proto tcp from any to $ext port {22, 25, 80} -> $server port {22, 25, 80}
rdr on $ext proto udp from any to $ext port 53 -> $server port 53
# setup a table to store a list of people who have been caught doing bad things
table <badguys> persist
antispoof log quick for $ext inet
block log all
block in quick inet6
pass out on $ext all queue (other, high)
pass in on { $int, lo0 } all queue (other, high)
pass out on { $int lo0 } all allow-opts
block in quick log from no-route to any
pass in on $ext inet proto icmp all
block in log on $ext inet proto icmp all icmp-type redir
# This will allow ssh connections in but will create limits
# Anyone who exceeds the limits will be added to the badguys table and firewalled
pass in on $ext proto tcp from any to any port 22 synproxy state \
(max-src-conn 20, max-src-conn-rate 10/5, overload <badguys> flush global) \
queue (other, high)
pass in on $int proto tcp from any to any port 22 queue (other, high)
pass in on $ext proto tcp from any to $server port 80
pass in on $ext proto tcp from any to $server port 25 queue low
pass in on $ext proto udp from any to $server port 53
# block the badguys who were caught earlier
block in log on $ext from <badguys> to any
# Add extra block rules for stuff that I want to be blocked differently than the default.
# Mostly this is done because these connections are very common (silly Windows boxes)
# and I don't want my logs flooded with them.
# For identd I want them blocked with a return so the connections fail quickly.
block return in on $ext proto tcp from any to any port 113
block in quick on $ext proto {tcp, udp} from any to any port {135:140, 42, 445}
block return out quick on $ext proto {tcp, udp} from any to any port {135:140, 42, 445}
# Block all web traffic from internal Windows systems.
# Note that they can still use the proxy server.
block return in log on $int proto tcp from any os "Windows" to any port 80
```

24 Other resources

http://www.openbsd.org/faq/index.html The OpenBSD FAQ

ftp://ftp.openbsd.org/pub/OpenBSD/doc/pf-faq.txt The PF User's Guide

http://www.sanitarium.net/unix_stuff/config_files/pf.conf.txt My pf.conf file