Building the Network You Need with OpenBSD's PF

BSDCan, Ottawa May 6th, 2009

Peter N. M. Hansteen
FreeCode AS

peter.hansteen@freecode.no, peter@bsdly.net

Table of Contents

Pitfalls: in, out, on
What is your local network, anyway?
Simple gateway (with NAT if you need to)
Simple gateway with NAT (cont'd.)
Simple gateway with NAT (cont'd.)
Simple gateway with NAT (cont'd.)
Testing your rule set
Domain names and host names?
That old and sad FTP thing
If we have to: ftp-proxy with redirection
This will become historical: pre-3.8 FTP proxies
Other historical ftp solutions: ftpsesame, pftpx
Tables make your life easier
Table commands
Filtering for services
Filtering for services (cont)
Giving spammers a hard time: you're not alone
Giving spammers a hard time (cont'd)
Giving spammers a hard time: The rules
Setting up spamd
Setting up spamd - FreeBSD
Greylisting: See the RFC

Greylisting: My admin told me not to talk to strangers
Setting up spamd
Track real SMTP connections: spamdlogd
Giving spammers a hard time (cont'd)
Giving spammers a hard time (cont'd)
Giving spammers a hard time (cont'd)
Connection lengths
Beating'em up some more: spamdb and greytrapping
spamdb and greytrapping
Greytrapping - the result
Keeping several spamds in sync
Some people really do not get it
Fixing for the people who really do not get it
Giving spammers a hard time: Conclusion
Turning away the brutes
Turning away the brutes: The rules
Turning away the brutes (cont'd)
Turning away the brutes (cont'd)
Expiring table entries with pfctl
expiretable tidies your tables
Advanced state tracking
State tracking (cont)

Physical Separation: The DMZ
DMZ ruleset
DMZ ruleset: tighten
Anchors
Anchors: commands
Anchors: ruleset
Anchors: alternative structure
Anchors - tag and quick
Including files
Wireless networks: background
Wireless networks made easy
Wireless networks: WPA setup
Wireless networks made easy (cont'd)
Wireless networks made easy (cont'd)
authpf: per user rules
Basic authpf setup
Basic authpf setup (cont)
Basic authpf setup (cont)
Per user rules
Wide open but actually shut
Open but shut: pf.conf
Sharing the load: Address pools

relayd
Basic relayd config
Basic relayd config (cont)
relayctl
Filtering for services, the NAT version
Back to the single NATed network
Single NAT, web & mail server on the inside: from the inside
Single NAT, web & mail server on the inside: from the inside
Filtering on interface groups
The power of tags
The filtering bridge
Where does it go?
OpenBSD bridge setup
FreeBSD bridge setup
Bridge PF filtering config
Handling non-routable addresses from elsewhere
Directing traffic with altq
Setting up for ALTQ
Setting up for ALTQ: FreeBSD
Setting up for ALTQ: NetBSD
What is your usable bandwidth?
ALTQ - prioritizing by traffic type

ALTQ - allocation by percentage
Queueing for a DMZ
Queueing for a DMZ: rules part 1
Queueing for a DMZ: rules part 2
overloading to a tiny queue
ALTQ - handling unwanted traffic
CARP and pfsync
CARP: project spec
CARP: project spec cont'd
CARP: project spec cont'd
Is your system CARP ready?
Setting up CARP
CARP: ifconfig
pfsync
What happens to the rule set?
carp config example
Carp ruleset
Making your network troubleshooting friendly
Then, do we let it all through?
The easy way out: The buck stops here
Letting ping through
Helping traceroute

Path MTU discovery
Path MTU discovery (cont'd)
Logging
Taking a peek with tcpdump
tcpdump is your friend
Matching log data to your rule set
Log to syslog
Statistics via labels
$variable label names
$variable label names: example
Keeping an eye on things with pftop
New in 4.5: pflow(4) and pflow state option
Graph your traffic: pfstat
Other log tools you may want to look into
Good logs for good debugging
Getting your setup just right
block-policy
skip
state-policy
state-defaults (new in 4.5)
timeout
limit

debug
ruleset-optimiation
optimization
Hygiene: scrub and antispoof
Testing your setup
Specification (possibly incomplete)
Debugging your setup
Debugging some more
Debug - use tcpdump
Have fun!
If you enjoyed this: Support OpenBSD!
References

This is not a HOWTO

The Pledge of the Network Admin

This is my network.

It is mine
or technically my employer's,
it is my responsibility
and I care for it with all my heart

there are many other networks a lot like mine,

but none are just like it.

I solemnly swear

that I will not mindlessly paste from HOWTOs.

You’re wondering ...

- *PF Looks Really Cool. Can I Run PF on My Linux Machine?*
- *I Know Some Linux, but I Need to Learn Some BSD. Any Pointers?*
- *Can You Recommend a GUI Tool for Managing My PF Rule Set?*
- *Is There a Tool I Can Use to Convert My OtherProduct® Setup to a PF Configuration?*
- *Where Can I Find Out More?*

You're wondering ... Linux?

PF Looks Really Cool. Can I Run PF on My Linux Machine?

You need a BSD. Preferably OpenBSD

Cool, new features: always in OpenBSD first

Porting to non-BSD takes major effort, no reports of completed Linux port (but a few have started)

You're wondering ... Learn BSD?

I Know Some Linux, but I Need to Learn Some BSD. Any Pointers?

- Network interface name equals *driver name* + sequence number, such as `xl0` (3Com), `em0`, `fxp0` (Intel), and so on
- Configuration centralized in `/etc/rc.conf`, on OpenBSD you create, edit `/etc/rc.conf.local` to override `rc.conf` defaults
- PF is configured via `/etc/pf.conf` and command line - pfctl(8)

You're wondering ... GUI tools?

Can You Recommend a GUI Tool for Managing My PF Rule Set?

Yes, GUI tools do exist.

No, this tutorial is taken in without them

Learn to understand `pf.conf` first, then make *informed* decision

Keep your `pf.conf` readable, already!

(`$EDITOR` + *`pf.conf`* is faster anyway)

You’re wondering ... Automatic conversion?

Is There a Tool I Can Use to Convert My OtherProduct® Setup to a PF Configuration?

Some tools claim to do it, DON’T trust them

OtherProduct™ features and approaches probably different from PF

Recommendation: Evaluate your needs, write a good spec, implement as fresh PF config

You’re wondering ... More info?

Where Can I Find Out More?

You’re here.

man pages, PF FAQ

Books, other resources (see references (references.html))

PF - Haiku

Compared to working with iptables, PF is like this haiku:

A breath of fresh air,
floating on white rose petals,
eating strawberries.

Now I'm getting carried away:

Hartmeier codes now,
Henning knows not why it fails,
fails only for n00b.

Tables load my lists,
tarpit for the asshole spammer,
death to his mail store.

CARP due to Cisco,
redundant blessed packets,
licensed free for me.

Jason Dixon, May 20th 2004
(http://marc.info/?l=openbsd-pf&m=108507584013046&w=2)

What PF is

Default packet filter (aka *firewall* with *NAT*) from OpenBSD 3.0 (December 2001)

Replaced IPFilter, which had to be removed due to licensing issues (which in turn lead to a *license audit* of the entire OpenBSD source tree)

Based on new code by Daniel Hartmeier (June 2001 ->), since hacked on by several others

High performance (see *http://www.benzedrine.cx/pf-paper.html*), *low maintenance*

Packet filter? Firewall?

Packet filter:

- Kernel level code (module + admin software) which determines network packets' paths

  'the world consists of packets, protocols, connections and ports'

- Important feature: Stopping undesirable traffic, 'firewall'

  A flexible *tool*, for *taking control* of your network traffic

NAT?

Network Address Translation

- *Why:* Internet commercialization (early 1990s)
- Predicted shortage of official IP-adresses
- Long term solution: IPv6 (128 bit addresses)
- Shorter term solutions: translation logic (RFC 1631, 1994) and private (non routable) adresses (RFC 1918, 1996)

When official addresses are not

- available
- practical

PF today

Packet filter with

- Filtrering on protocol, port, packet type, address, operating system
- Redirection (to other port, local daemon, other machine etc)
- NAT
- traffic shaping (altq)
- Human readable configuration

In the base systems of OpenBSD, FreeBSD, NetBSD and DragonFlyBSD

Simplest possible setup

`/etc/rc.conf[.local]`

```
pf=YES                  # enable pf
pf_rules=/etc/pf.conf # specify which file contains your rules
```

enable with

```
$ sudo pfctl -e
```

Note Rebooting loads the `rc` script's default rule set

Simplest possible setup (FreeBSD)

/etc/rc.conf

```
pf_enable="YES"                 # Enable PF (load module if required)
pf_rules="/etc/pf.conf"         # rules definition file for pf
pf_flags=""                     # additional flags for pfctl startup
pflog_enable="YES"              # start pflogd(8)
pflog_logfile="/var/log/pflog"  # where pflogd should store the logfile
pflog_flags=""                  # additional flags for pflogd startup
```

Fortunately most of these are the defaults, you need only add

```
pf_enable="YES"                 # Enable PF (load module if required)
pflog_enable="YES"              # start pflogd(8)
```

enable with

```
FreeBSD:~$ sudo pfctl -ef /etc/pf.conf
```

or

```
$ sudo /etc/rc.d/pf start
```

Simplest possible setup (NetBSD)

NetBSD from 2.0 - available via packages `security/pflkm`.

From NetBSD 3.0 on, PF is in the base system

Kernel configuration file:

```
pseudo-device  pf # PF packet filter
pseudo-device  pflog # PF log interface
```

`/etc/rc.conf`:

```
pf=YES
pflogd=YES
```

```
NetBSD$ sudo modload /usr/lkm/pf.o;
```

or

```
NetBSD$ sudo pfctl -e
```

```
NetBSD$ sudo /etc/rc.d/pf start ; sudo /etc/rc.d/pflogd start
```

in `/etc/lkm.conf`:

```
/usr/lkm/pf.o - - - - AFTERMOUNT
```

First rule set - single machine

/etc/pf.conf

```
block in all
pass out all keep state
```

OpenBSD 4.1 onwards:

```
# minimal rule set, OpenBSD 4.1 and newer keeps state by default
block in all
pass out all
```

enable with

```
$ sudo pfctl -ef /etc/pf.conf
```

Testing your first rule set

You need to test your work

- Name resolution: `$ host openbsd.org` should work
- Remote login: `$ ssh myotherbox.com` should work
- Surf the web: `$ lynx http://www.openbsd.org` should work

Connections from you to elsewhere should work

Connections from elsewhere to you should *NOT* work

Slightly stricter

/etc/pf.conf

```
tcp_services = "{ ssh, smtp, domain, www, pop3, auth, pop3s }"
udp_services = "{ domain }"

block all
pass out proto tcp to any port $tcp_services
pass proto udp to any port $udp_services
```

NOTE: Default to deny, enable only the stuff we need

- load your new rules

```
$ sudo pfctl -f /etc/pf.conf
```

for syntax check only:

```
$ sudo pfctl -nf /etc/pf.conf
```

Note: only valid rule sets load, flushing rarely makes sense

Testing your rule set

You need to test your work, again

- Name resolution: `$ host netbsd.org` should work (different domain, don't let the cache fool you)
- Remote login: `$ ssh myotherbox.com` should work
- Surf the web: `$ lynx http://www.openbsd.org` should work

Connections from you to **`ssh`**, **`smtp`**, **`domain`**, **`www`**, **`pop3`**, **`auth`**, **`pop3s`** elsewhere should work

All other connections from you should FAIL

Connections from elsewhere to you should *NOT* work

Statistics from pfctl

```
peter@skapet:~$ sudo pfctl -s info
Status: Enabled for 17 days 00:24:58          Debug: Urgent

Interface Stats for ep0                IPv4              IPv6
  Bytes In                       9257508558                 0
  Bytes Out                       551145119               352
  Packets In
    Passed                        7004355                  0
    Blocked                         18975                  0
  Packets Out
    Passed                        5222502                  3
    Blocked                            65                  2

State Table                          Total              Rate
  current entries                       15
  searches                         19620603            13.3/s
  inserts                            173104             0.1/s
  removals                           173089             0.1/s
Counters
  match                              196723             0.1/s
  bad-offset                              0             0.0/s
  fragment                               22             0.0/s
  short                                   0             0.0/s
  normalize                               0             0.0/s
  memory                                  0             0.0/s
```

bad-timestamp	0	0.0/s
congestion	0	0.0/s
ip-option	28	0.0/s
proto-cksum	325	0.0/s
state-mismatch	983	0.0/s
state-insert	0	0.0/s
state-limit	0	0.0/s
src-limit	26	0.0/s
synproxy	0	0.0/s

A gateway

Single machine: *Me vs the network*

- **`in`** from the Internet to me
- **`out`** from me to the Internet

Gateway: *I decide what passes through me*

- **`in`** from the one or more networks to me
- **`out`** from me one or more networks
- *hey, there's a network behind you!*

Pitfalls: in, out, on

If you write

```
pass in inet proto tcp on re1 from re1:network to re0:network \
     port $ports keep state
```

then you also need

```
pass out inet proto tcp on re0 from re1:network to re0:network \
     port $ports keep state
```

but do you actually mean

```
pass inet proto tcp from re1:network to any port $ports keep state
```

What is your local network, anyway?

`interface:network` - the *network* connected to *interface*

Your local net could be

```
localnet = $int_if:network
```

or network as **`192.168.100.0/24`**, or **`fec0:dead:beef::/64`** or a list of networks -

sample filtering rule:

```
pass inet proto tcp from $localnet to any port $ports keep state
```

Simple gateway (with NAT if you need to)

Enable gatewaying

For IPv4: `# sysctl net.inet.ip.forwarding=1`

For IPv6: `# sysctl net.inet6.ip6.forwarding=1`

Make permanent, `/etc/sysctl.conf`

```
net.inet.ip.forwarding=1
net.inet6.ip6.forwarding=1
```

On FreeBSD, `/etc/rc.conf`

```
gateway_enable="YES" #for ipv4
ipv6_gateway_enable="YES" #for ipv6
```

Simple gateway with NAT (cont'd.)

Check interface status with **`ifconfig -a`**

```
$ ifconfig -a
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 33208
        groups: lo
        inet 127.0.0.1 netmask 0xff000000
        inet6 ::1 prefixlen 128
        inet6 fe80::1%lo0 prefixlen 64 scopeid 0x5
xl0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        lladdr 00:50:da:21:e4:af
        groups: egress
        media: Ethernet autoselect (100baseTX full-duplex)
        status: active
        inet 213.187.179.198 netmask 0xfffffffc broadcast 213.187.179.19
        inet6 fe80::250:daff:fe21:e4af%xl0 prefixlen 64 scopeid 0x2
xl1: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        lladdr 00:50:da:3f:b2:86
        media: Ethernet autoselect (100baseTX full-duplex)
        status: active
        inet 192.168.103.1 netmask 0xffffff00 broadcast 192.168.103.255
        inet6 fe80::250:daff:fe3f:b286%xl1 prefixlen 64 scopeid 0x3
enc0: flags=0<> mtu 1536
pflog0: flags=141<UP,RUNNING,PROMISC> mtu 33208
        groups: pflog
```

Simple gateway with NAT (cont'd.)

/etc/pf.conf

```
ext_if = "re0" # macro for external interface - use tun0 for PPPoE
int_if = "re1" # macro for internal interface
# ext_if IP address is (may be) dynamic
nat on $ext_if from $localnet to any -> ($ext_if)
block all
pass inet proto tcp from { lo0, $int_if:network } to any keep state
```

Simple gateway with NAT (cont'd.)

or perhaps

```
client_out = "{ ftp-data, ftp, ssh, domain, pop3, auth, nntp, http, \
             https, 446, cvspserver, 2628, 5999, 8000, 8080 }"
udp_services = "{ domain, ntp }"

pass quick inet proto { tcp, udp } to any port $udp_services keep state

pass inet proto tcp from $int_if:network to any port $client_out \
                flags S/SA keep state

pass in inet proto tcp from any to any port ssh
```

Rule evaluations is top to bottom, *last* matching rule wins.

The *quick* keyword exits rule evaluation when current rule matches (quick rule always wins)

Testing your rule set

You need to test your work, again

- Name resolution: `$ host freebsd.org` should work from the gateway and any host in your local net (check a different domain, don't let the cache fool you)
- Remote login: `$ ssh myotherbox.com` should work from the gateway and any host in your local net
- Surf the web: `$ lynx http://www.openbsd.org` should work from the gateway and any host in your local net

Connections from your net to **`$client_out`** elsewhere should work

All other connections from your net should FAIL

Connections from elsewhere to your net should *NOT* work

Domain names and host names?

Yes, you can use **`pass from self to myotherbox.mydomain.com`**, but -

Your rules will then be valid only after you have name resolution up and running

Possible workaraound: load default rule set, run script from `rc.local` to check DNS, then load real rule set

That old and sad FTP thing

Old and weird protocol (older than TCP/IP, topic of $>$ 50 RFCs!)

- Passwords transferred as clear text (no encryption)
- Uses at least two connections
- Data transferred on randomly chosen port

Recommendation: use something else

If we have to: ftp-proxy with redirection

In the base system, load the daemon if you need it

```
/etc/rc.conf[.local]

ftpproxy_flags=""
```

```
/etc/pf.conf
```

NAT section anchors

```
nat-anchor "ftp-proxy/*"
rdr-anchor "ftp-proxy/*"
```

the redirection

```
rdr pass on $int_if proto tcp from any to any port ftp -> 127.0.0.1 \
        port 8021
```

in your filtering section

```
anchor "ftp-proxy/*"

pass out proto tcp from $proxy to any port 21 keep state
```

This will become historical: pre-3.8 FTP proxies

ftp over nat: ftp-proxy (OpenBSD 3.8 and earlier *equivalents* such as NetBSD, FreeBSD pre-7.0)

`/etc/inetd.conf`

```
127.0.0.1:8021 stream tcp nowait root /usr/libexec/ftp-proxy ftp-proxy -
```

- restart inetd

```
FreeBSD+NetBSD$ sudo /etc/rc.d/inetd restart
```

`/etc/pf.conf`

```
 rdr on $int_if proto tcp from any to any port ftp -> 127.0.0.1 \
                port 8021
 # [...]
 pass in on $ext_if inet proto tcp from port ftp-data to ($ext_if) \
    user proxy flags S/SA keep state
```

- load your new rules

```
$ sudo pfctl -f /etc/pf.conf
```

Other historical ftp solutions: ftpsesame, pftpx

For official addresses on the inside: ftpsesame (OpenBSD + others), fetchable from

http://www.sentia.org/projects/ftpsesame/ (on FreeBSD: - `ftp/ftpsesame`)

configured via an anchor (sub ruleset), cut and paste from **`man ftpsesame`**

pftpx, the next generation ftp-proxy

http://www.sentia.org/downloads/pftpx-0.8.tar.gz, (on FreeBSD - `ftp/pftpx`) cut and paste from **`man pftpx`**

Tables make your life easier

```
table <clients> { 192.168.2.0/24, !192.168.2.5 }
```

the file `/etc/clients`

```
192.168.2.0/24
!192.168.2.5
```

in `/etc/pf.conf`

```
table <clients> persist file /etc/clients

pass inet proto tcp from <clients> to any port $client_out
```

Table commands

Command line tables manipulation:

Add a table entry

```
$ sudo pfctl -t clients -T add 192.168.1/16
```

Delete a table entry

```
$ sudo pfctl -t clients -T delete 192.168.1.116
```

Show table contents

```
$ sudo pfctl -t clients -T show >/etc/clients
```

Replace table contents from a file

```
$ sudo pfctl -t clients -T replace -f /etc/clients
```

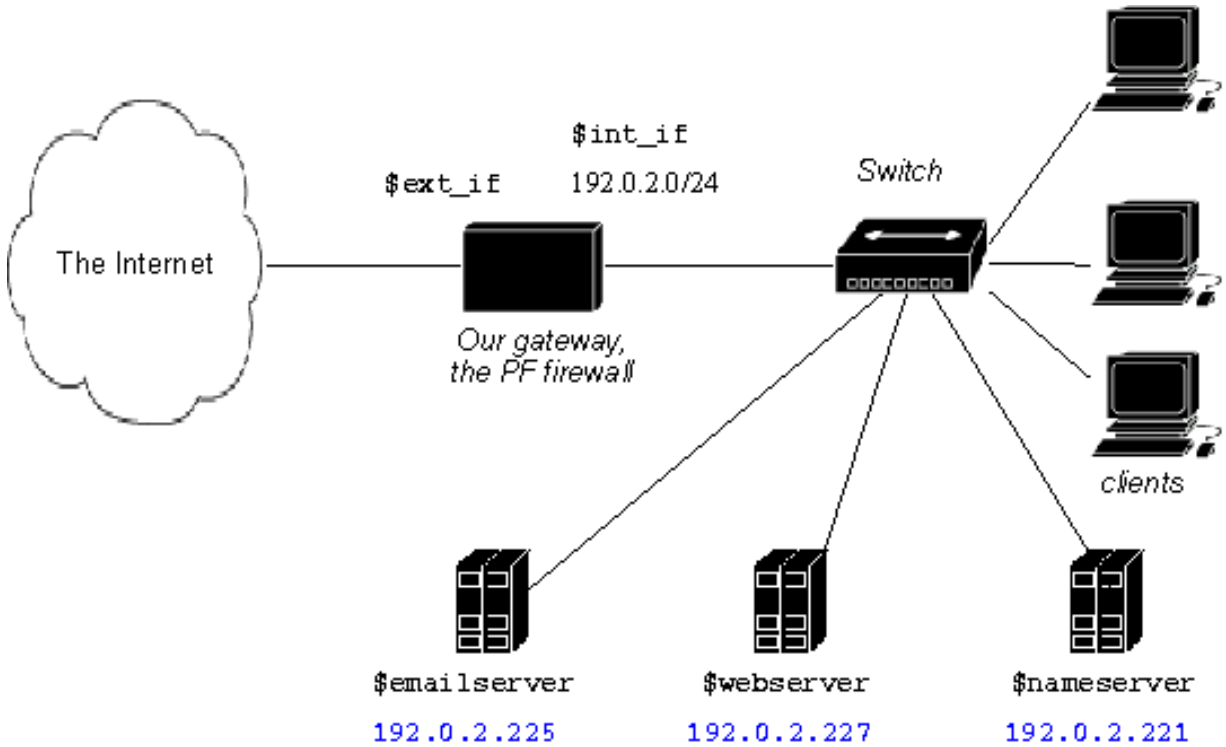
Commonly used commands - likely scripting candidates

Tip: look into cron(8) jobs

Worth noting: several apps including dhcpd and bgpd can interact with PF via tables

Filtering for services

We introduce servers in the network



Filtering for services (cont)

The obvious macros

```
webserver = "192.0.2.227"
webports = "{ http, https }"
emailserver = "192.0.2.225"
email = "{ smtp, pop3, imap, imap3, imaps, pop3s }"
nameservers = "{ 192.0.2.221, 192.0.2.223 }"
```

and rules that use them

```
pass proto tcp from any to $webserver port $webports synproxy state
pass proto tcp from any to $emailserver port $email synproxy state
pass log proto tcp from $emailserver to any port smtp synproxy state
pass inet proto { tcp, udp } from any to $nameservers port domain
```

Giving spammers a hard time: you're not alone

Spammers are a pain. How do we deal?

- Spammers send *large* numbers of messages; you are probably not the first recipient

  - > Known spam senders (hijacked machines) are added to blacklists

- Spammers tend not to obey RFCs (RFC1123, RFC2821/RFC5321)

  - > Greylisting (www.greylisting.org (http://www.greylisting.org)) - with 45n temporary error *still* works!

Giving spammers a hard time (cont'd)

spamd

- tarpit where black listed hosts get stuck - answers 1 byte at the time, tunable interval (-s option)
- Grey-lists unknowns using 45n (temporary local error), well behaved ones are let through within a reasonable time

Introduced in OpenBSD 3.3 (May 2003)

Since OpenBSD 4.1, spamd runs in greylisting mode by default

Giving spammers a hard time: The rules

/etc/pf.conf

```
table <spamd> persist
table <spamd-white> persist
rdr pass on $ext_if inet proto tcp from <spamd> to \
  { $ext_if, $int_if:network } port smtp -> 127.0.0.1 port 8025
rdr pass on $ext_if inet proto tcp from !<spamd-white> to \
  { $ext_if, $int_if:network } port smtp -> 127.0.0.1 port 8025
```

Essential data in the spamd and spamd-white tables

Setting up spamd

Edit `spamd.conf` to specify blacklists and whitelists

```
all:\
:uatraps:whitelist:

uatraps:\
        :black:\
        :msg="SPAM. Your address %A has sent spam within the last 24 hou
        :method=http:\
        :file=www.openbsd.org/spamd/traplist.gz

whitelist:\
        :white:\
        :method=file:\
        :file=/var/mail/whitelist.txt
```

Setting up spamd - FreeBSD

On FreeBSD, spamd is a port, `mail/spamd/`

Needs a fdescfs(5) (http://www.freebsd.org/cgi/man.cgi?query=fdescfs&sektion=5) mounted on `/dev/fd/` (see the package message)

Greylisting: See the RFC

greylisting: you need to behave

RFC5321 (Oct 2008, supersedes RFC2821), section 4.5.4.1, "Sending Strategy":

> "In a typical system, the program that composes a message has some method for requesting immediate attention for a new piece of outgoing mail, while mail that cannot be transmitted immediately MUST be queued and periodically retried by the sender."

and

> "The sender MUST delay retrying a particular destination after one attempt has failed. In general, the retry interval SHOULD be at least 30 minutes; however, more sophisticated and variable strategies will be beneficial when the SMTP client can determine the reason for non-delivery."

RFC5321 goes on to state that

> "Retries continue until the message is transmitted or the sender gives up; the give-up time generally needs to be at least 4-5 days."

Greylisting: My admin told me not to talk to strangers

greylisting: a white lie

“my admin told me not to talk to strangers”

Well intended senders will come back within a reasonable time.

It’s pure pedantry; false positives are rare.

Shifts the load back to the sender

Setting up spamd

Enter spamd lines in `rc.conf[.local]`

```
spamd_flags="-v -G 2:4:864" # for normal use: "" and see spamd-setup(8)
```

NOTE: Greylisting is enabled by default. Use **`spamd_black`** if you want to enable blacklist-only mode.

Maintenance handled with spamd-setup(8), fetches the lists you specified

Tables automatically updated, use spamdb(8)

In addition, you need rules to let legitimate email through

Other interesting options: **`-s`** # of seconds between bytes, **`-w`** window size # of bytes

Track real SMTP connections: spamdlogd

```
emailserver = "192.0.2.225"
pass log proto tcp from any to $emailserver port $email synproxy state
pass log proto tcp from $emailserver to any port smtp synproxy state
```

or log to separate pflog interface

```
pass log (to pflog1) proto tcp from any to $emailserver port $email
pass log (to pflog1) proto tcp from $emailserver to any port smtp
```

Giving spammers a hard time (cont'd)

spamd in use

Typical log entries (with -v):

```
Oct  2 19:55:05 delilah spamd[26905]: (GREY) 83.23.213.115: <gilbert@key
Oct  2 19:55:05 delilah spamd[26905]: 83.23.213.115: disconnected after
Oct  2 19:55:05 delilah spamd[26905]: 83.23.213.115: connected (2/1)
Oct  2 19:55:06 delilah spamd[26905]: (GREY) 83.23.213.115: <gilbert@key
Oct  2 19:55:06 delilah spamd[26905]: 83.23.213.115: disconnected after
Oct  2 19:57:07 delilah spamd[26905]: (BLACK) 65.210.185.131: <bounce-3
Oct  2 19:58:50 delilah spamd[26905]: 65.210.185.131: From: Auto lnsuran
Oct  2 19:58:50 delilah spamd[26905]: 65.210.185.131: Subject: Start SAV
Oct  2 19:58:50 delilah spamd[26905]: 65.210.185.131: To: adm@dataped.nc
Oct  2 20:00:05 delilah spamd[26905]: 65.210.185.131: disconnected after
Oct  2 20:03:48 delilah spamd[26905]: 222.240.6.118: connected (1/0)
Oct  2 20:03:48 delilah spamd[26905]: 222.240.6.118: disconnected after
Oct  2 20:06:51 delilah spamd[26905]: 24.71.110.10: connected (1/1), lis
Oct  2 20:07:00 delilah spamd[26905]: 221.196.37.249: connected (2/1)
Oct  2 20:07:00 delilah spamd[26905]: 221.196.37.249: disconnected after
Oct  2 20:07:12 delilah spamd[26905]: 24.71.110.10: disconnected after 2
```

Stops a lot of spam. Error rate depends on black lists used.

Giving spammers a hard time (cont'd)

This used to be my best log:

```
Dec 11 23:57:24 delilah spamd[32048]: 69.6.40.26: connected (1/1),
lists: spamhaus spews1 spews2
Dec 12 00:30:08 delilah spamd[32048]: 69.6.40.26: disconnected after
1964 seconds. lists: spamhaus spews1 spews2
```

A sender at wholesalebandwidth.com tried a total of 13 times from December 9th through 12th, giving up after one last attempt which lasted 32 minutes, 44 seconds without delivering a message.

Giving spammers a hard time (cont'd)

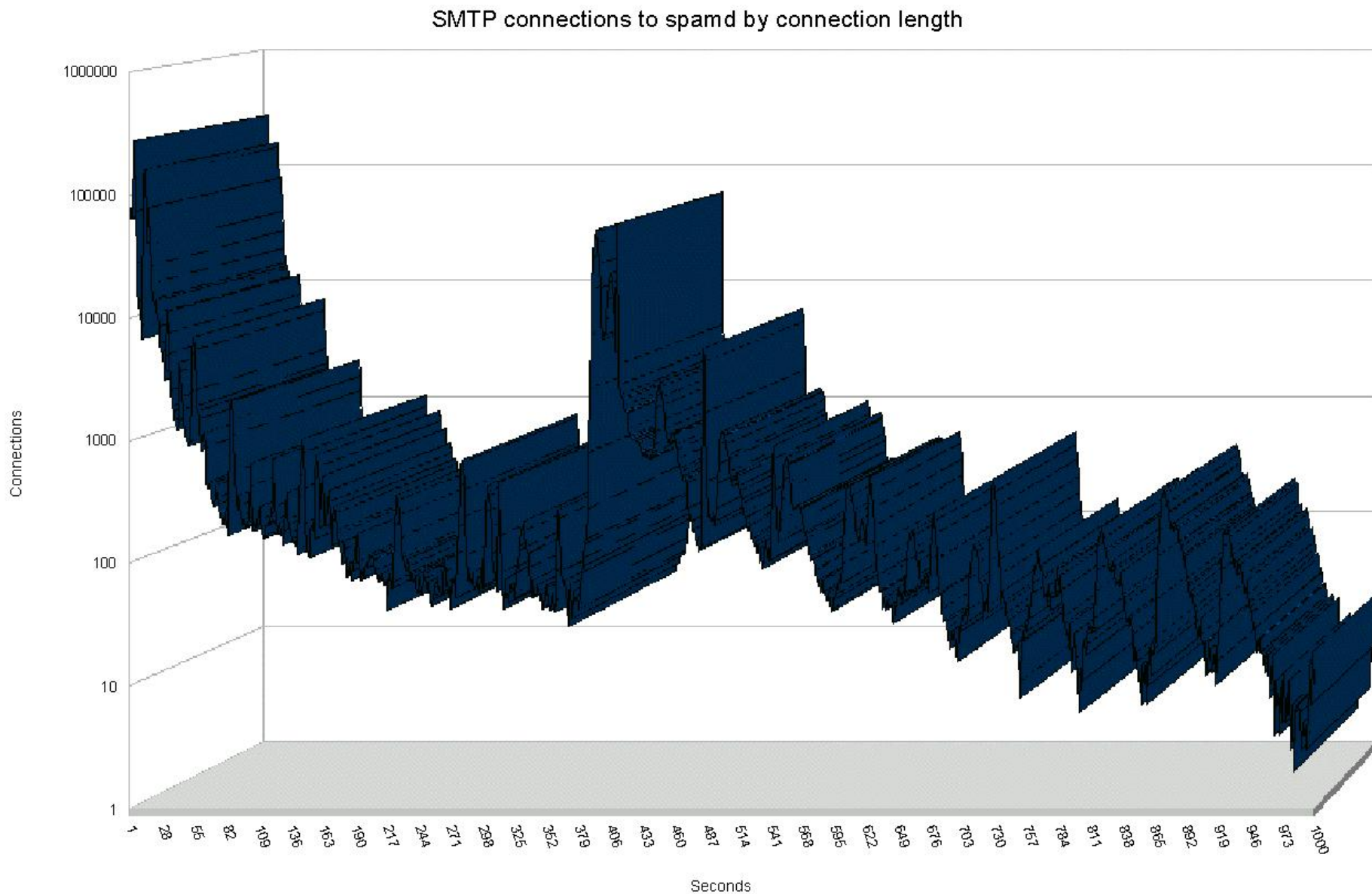
But the record did not stand for long:

```
peter@delilah:~$ grep disconnected /var/log/spamd | awk '{print $9}' | so
   1 42673
   1 36099
   1 14714
   1 10170
   1 5966
   1 5878
   1 5866
   1 5845
   1 5709
   1 5707
```

42673 seconds = almost twelve hours! (others have reported >60K seconds)

```
Dec 21 14:22:44 delilah spamd[29949]: 85.152.224.147: connected (5/2)
Dec 21 14:22:46 delilah spamd[29949]: 85.152.224.147: connected (6/2)
Dec 21 14:22:47 delilah spamd[29949]: 85.152.224.147: disconnected
after 3 seconds.
Dec 22 02:13:59 delilah spamd[29949]: 85.152.224.147: disconnected
after 42673 seconds.
```

Connection lengths



Beating’em up some more: spamdb and greytrapping

spamdb database tool - essential for whitelisting, deleting, etc

spamlogd whitelist updater

- Started and handled automatically from the RC scripts, easy to forget the first time if you’re doing this by hand and don’t want to reboot.

spamdb and greytrapping

Enter obviously bogus addresses into your blacklist with spamdb -T

```
peter@delilah:~$ spamdb -T -a "<wkitp98zpu.fsf@datadok.no>"
```

other useful options: **-t** enter as TRAPPED, **-d** delete entries

Greytrapping - the result

Sure enough, the spammers thought that address was just as usable as almost two years earlier:

```
Nov  6 09:50:25 delilah spamd[23576]: 210.214.12.57: connected (1/0)
Nov  6 09:50:32 delilah spamd[23576]: 210.214.12.57: connected (2/0)
Nov  6 09:50:40 delilah spamd[23576]: (GREY) 210.214.12.57:
<gilbert@keyholes.net> -> <wkitp98zpu.fsf@datadok.no>
Nov  6 09:50:40 delilah spamd[23576]: 210.214.12.57: disconnected
after 15 seconds.
Nov  6 09:50:42 delilah spamd[23576]: 210.214.12.57: connected (2/0)
Nov  6 09:50:45 delilah spamd[23576]: (GREY) 210.214.12.57:
<bounce-3C7E40A4B3@branch15.summer-bargainz.com> ->
<adm@dataped.no>
Nov  6 09:50:45 delilah spamd[23576]: 210.214.12.57: disconnected
after 13 seconds.
Nov  6 09:50:50 delilah spamd[23576]: 210.214.12.57: connected (2/0)
Nov  6 09:51:00 delilah spamd[23576]: (GREY) 210.214.12.57:
<gilbert@keyholes.net> -> <wkitp98zpu.fsf@datadok.no>
Nov  6 09:51:00 delilah spamd[23576]: 210.214.12.57: disconnected
after 18 seconds.
Nov  6 09:51:02 delilah spamd[23576]: 210.214.12.57: connected (2/0)
Nov  6 09:51:02 delilah spamd[23576]: 210.214.12.57: disconnected
after 12 seconds.
Nov  6 09:51:02 delilah spamd[23576]: 210.214.12.57: connected (2/0)
Nov  6 09:51:18 delilah spamd[23576]: (GREY) 210.214.12.57:
```

```
<gilbert@keyholes.net> -> <wkitp98zpu.fsf@datadok.no>
Nov  6 09:51:18 delilah spamd[23576]: 210.214.12.57: disconnected
after 16 seconds.
Nov  6 09:51:18 delilah spamd[23576]: (GREY) 210.214.12.57:
<bounce-3C7E40A4B3@branch15.summer-bargainz.com> ->
<adm@dataped.no>
Nov  6 09:51:18 delilah spamd[23576]: 210.214.12.57: disconnected
after 16 seconds.
Nov  6 09:51:20 delilah spamd[23576]: 210.214.12.57: connected (1/1),
lists: spamd-greytrap
Nov  6 09:51:23 delilah spamd[23576]: 210.214.12.57: connected (2/2),
lists: spamd-greytrap
Nov  6 09:55:33 delilah spamd[23576]: (BLACK) 210.214.12.57:
<gilbert@keyholes.net> -> <wkitp98zpu.fsf@datadok.no>
Nov  6 09:55:34 delilah spamd[23576]: (BLACK) 210.214.12.57:
<bounce-3C7E40A4B3@branch15.summer-bargainz.com> ->
<adm@dataped.no>
```

Keeping several spamds in sync

Starting with OpenBSD 4.1, you can sync greylists

spamd command line flags:

-Y *sync target*

-y *sync listener*

on *mainoffice-gw.example.com*

```
-Y minorbranch-gw.example.com -y re0
```

on *minorbranch-gw.example.com*

```
-Y mainoffice-gw.example.com -y re0
```

Some people really do not get it

Greylisting inevitably means delay for the *initial* message from a new correspondent.

Some sites are misconfigured; do not retry or retry way too fast (seconds)

Others have many outgoing MXes and random selection for retry

Some people are simply too impatient

Fixing for the people who really do not get it

For the impatient:

```
table <localwhite> file "/etc/mail/whitelist.txt"
```

Disable **`rdr`** by putting this at the top of the **`rdr`** block:

```
no rdr proto tcp from <localwhite> to $mailservers port smtp
```

Edit `whitelist.txt`, reload rule set or replace <**`localwhite`**> table contents using pfctl; all table options available.

Giving spammers a hard time: Conclusion

Summing up:

- High precision
- Good effect, minimal load
- Blacklisting is never better than poorest data source used

  We ended up using Bob Beck's traplist (greytrapping generated) at http://www.openbsd.org/spamd/traplist.gz (http://www.openbsd.org/spamd/traplist.gz) - as far as we know no false positives. (graph (hosts-in-uatraps.jpg))

- Prime example: Steve Williams' October 20th, 2006 message to the OpenBSD-misc mailing list (http://marc.info/?l=openbsd-misc&m=116136841831550&w=2) - got rid of more than 95% of spam using spamd in pure greylisting mode
- Greytrapping is fun and effective: See bsdly.blogspot.com (http:/bsdly.blogspot.com/)

Turning away the brutes

You have all seen this:

```
Sep 26 03:12:34 skapet sshd[25771]: Failed password for root from
200.72.41.31 port 40992 ssh2
Sep 26 03:12:34 skapet sshd[5279]: Failed password for root from
200.72.41.31 port 40992 ssh2
Sep 26 03:12:35 skapet sshd[5279]: Received disconnect from
200.72.41.31: 11: Bye Bye
Sep 26 03:12:44 skapet sshd[29635]: Invalid user admin from
200.72.41.31
Sep 26 03:12:44 skapet sshd[24703]: input_userauth_request:
invalid user admin
Sep 26 03:12:44 skapet sshd[24703]: Failed password for invalid user
admin from 200.72.41.31 port 41484 ssh2
Sep 26 03:12:44 skapet sshd[29635]: Failed password for invalid user
admin from 200.72.41.31 port 41484 ssh2
Sep 26 03:12:45 skapet sshd[24703]: Connection closed by 200.72.41.31
Sep 26 03:13:10 skapet sshd[11459]: Failed password for root from
200.72.41.31 port 43344 ssh2
```

Turning away the brutes: The rules

/etc/pf.conf

```
table <bruteforce> persist

block quick from <bruteforce>

pass inet proto tcp from any to $int_if:network port $tcp_services \
        flags S/SA keep state \
(max-src-conn 100, max-src-conn-rate 15/5, \
         overload <bruteforce> flush global)
```

max-src-conn: # of connections from one host

max-src-conn-rate: rate of new connections: 15 connections per 5 seconds.

overload <bruteforce>: offenders go to the blocked table

flush global : kill all connections

Turning away the brutes (cont'd)

Tighten a bit for ssh, differentiate:

/etc/pf.conf

```
table <bruteforce> persist

block quick from <bruteforce>

# tighter for ssh
pass quick proto tcp from any to any port ssh \
        flags S/SA keep state \
        (max-src-conn 15, max-src-conn-rate 5/3, \
        overload <bruteforce> flush global)

pass inet proto tcp from any to $int_if:network port $tcp_services \
        flags S/SA keep state \
(max-src-conn 100, max-src-conn-rate 15/5, \
         overload <bruteforce> flush global)
```

Turning away the brutes (cont'd)

You may not need to **`block`** all of your **`overload`**ers

Eg mail or web service -

- put overloaders in a minimal-bandwidth queue (ALTQ)
- rdr overloaders to specific site

Expiring table entries with pfctl

These tables grow; waste memory

Table contents could become less useful over time (DHCP leases expire, etc)

In *OpenBSD 4.1 and newer*, pfctl can expire table entries:

```
# pfctl -t bruteforce -T expire 86400
```

expiretable tidies your tables

Henrik Gustafsson's expiretable

Remove <**bruteforce**> table entries older than 24 hours - add to `/etc/rc.local`:

```
/usr/local/sbin/expiretable -v -d -t 24h bruteforce
```

Get it from ports/packages or http://expiretable.fnord.se/

Advanced state tracking

`source-track rule` - your limits apply per rule

vs

`source-track global` this rule counts towards the global values

State tracking (cont)

Prevent floods:

```
pass inet proto tcp from any to $webserver port www \
        flags S/SA keep state \
(max-src-conn-rate 15/5, \
         max-src-nodes 250, max-src-states 100, source-track rule)
```

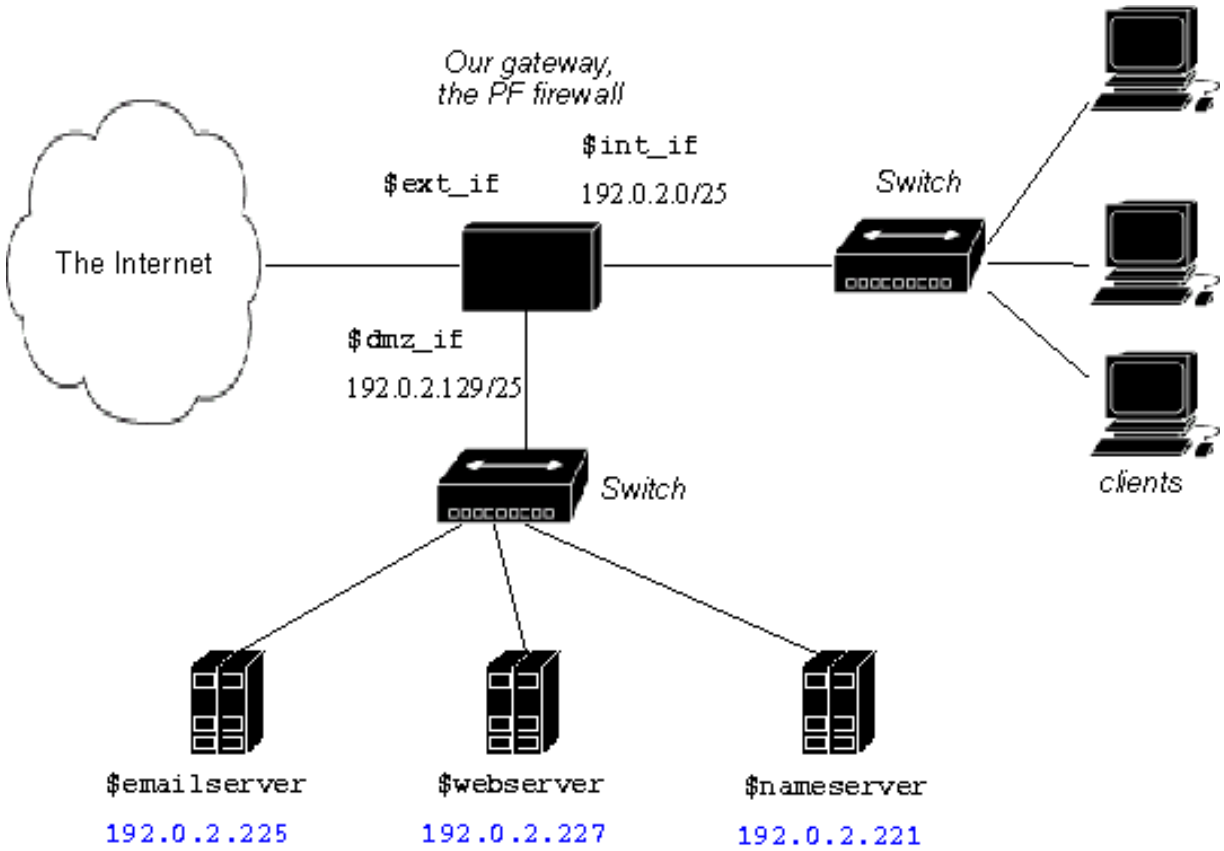
`max-src-nodes`: number of distinct hosts (IP addresses) allowed to have states

`max-src-states`: number of states allowed per host

Others simply dropped

Physical Separation: The DMZ

Put the servers on a separate subnet



DMZ ruleset

How much damage does this do to your rule set?

Not a lot, actually

You may need to edit your **`$webserver`**, **`$emailserver`**, **`$nameservers`**, if their addresses change

DMZ ruleset: tighten

You may want to allow only what's needed:

```
pass in on $ext_if proto { tcp, udp } from any to $nameservers \
     port domain
pass in on $int_if proto { tcp, udp } from $localnet to $nameservers \
     port domain
pass out on $dmz_if proto { tcp, udp } from any to $nameservers \
     port domain
pass in on $ext_if proto tcp from any to $webserver port $webports
pass in on $int_if proto tcp from $localnet to $webserver \
     port $webports
pass out on $dmz_if proto tcp from any to $webserver port $webports
pass in log on $ext_if proto tcp from any to $mailserver port smtp
pass in log on $int_if proto tcp from $localnet to $mailserver \
     port $email
pass out log on $dmz_if proto tcp from any to $mailserver port smtp
pass in on $dmz_if from $mailserver to any port smtp
pass out log on $ext_if proto tcp from $mailserver to any port smtp
```

Anchors

Named sub-rulesets

Used by several apps for inserting on the fly rules (ftp-proxy, relayd, others)

useful for grouping related rules

yes, it's technically possible to populate anchors from the command line:

```
# echo "block drop all" | pfctl -a baddies -f -
```

Anchors: commands

Load from file

```
# pfctl -a baddies -f /etc/anchor-baddies
```

List rules in an anchor

```
# pfctl -a baddies -s rules
```

Anchors: ruleset

Include from file

In your `pf.conf`

```
anchor ssh-good
     load anchor ssh-good from "/etc/anchor-ssh-good"

/etc/anchor-ssh-good

table <sshbuddies> file "/etc/sshbuddies"
     pass inet proto tcp from <sshbuddies> to any port ssh

/etc/sshbuddies

192.168.103.84
10.11.12.13
```

Anchors: alternative structure

alternative structure: common criteria

```
anchor "dmz" on $dmz_if {
        pass out proto { tcp udp } to $nameservers port domain
pass out proto tcp to $webservers port { www https }
pass out proto tcp to $mailserver port smtp
pass in log (all, to pflog1) in proto tcp from $mailserver \
                to any port smtp
}
```

Anchors - tag and quick

`quick` with tags

```
anchor "dmz" on $dmz_if {
        pass out proto { tcp udp } to $nameservers port domain tag GOOD
pass out proto tcp to $webservers port { www https } tag GOOD
pass out proto tcp to $mailserver port smtp tag GOOD
pass in log (all, to pflog1) in proto tcp from $mailserver
                to any port smtp tag GOOD
block log quick ! tagged GOOD
}
```

Including files

Starting with 4.3, you can split `pf.conf` into parts and include subsets:

```
include /etc/pf/subseta.rules
include /etc/pf/subsetb.rules
```

Wireless networks: background

"we're not using wires anymore" – IEEE 802.11 security measures:

- First try: *Wired Equivalent Privacy (WEP)*

  weak link level encryption - broken; deters naive attackers

- Second try: *WiFi Protected Access (WPA)*

  Flexible and complex - better on paper; "constantly improving"; support varies (supported in OpenBSD 4.4)

MAC address filtering: not really useful, but if you want to, see `brconfig`

- > you need SSH & SSL

See eg www.kjhole.com (http://www.kjhole.com/Standards/WiFi/WiFiDownloads.html) for references.

Wireless networks made easy

Check your dmesg

```
ath0 at pci1 dev 4 function 0 "Atheros AR5212" rev 0x01: irq 11
ath0: AR5212 5.6 phy 4.1 rf5111 1.7 rf2111 2.3, ETSI1W, address
00:0d:88:c8:a7:c4
```

Access point, OpenBSD `/etc/hostname.ath0`:

```
up media autoselect mediaopt hostap mode 11b chan 6 nwid unwiredbsd \
 nwkey 0x1deadbeef9
inet 10.168.103.1
```

From OpenBSD 4.4 onwards, we have WPA

Wireless networks: WPA setup

From OpenBSD 4.4 onwards, we have WPA.

For wpa keys, use `wpa-psk`:

```
$ wpa-psk unwiredbsd mylongpassphase
0x7579c38e59faaa3b64bd8372e94f74fe7ae2e4e91af154c956a9bfd0240ac9d0
```

A WPA access point config:

```
up media autoselect mediaopt hostap mode 11b chan 6 nwid unwiredbsd \
 wpapsk $(wpa-psk unwiredbsd mylongpassphase)
```

Wireless networks made easy (cont'd)

Client:

```
$ sudo ifconfig ath0 up mode 11b chan 11 nwid unwiredbsd nwkey 0x1deadbe
$ sudo dhclient ath0
DHCPREQUEST on ath0 to 255.255.255.255 port 67
DHCPREQUEST on ath0 to 255.255.255.255 port 67
DHCPACK from 10.50.90.1
bound to 10.50.90.11 -- renewal in 1800 seconds.
```

Make permanent with `/etc/hostname.ath0`

```
up media autoselect mode 11b chan 6 nwid unwiredbsd nwkey 0x1deadbeef9
dhcp
```

WPA pre-shared

```
up media autoselect mode 11b chan 6 nwid unwiredbsd \
wpapsk $(wpa-psk unwiredbsd mylongpassphase)
```

Wireless networks made easy (cont'd)

Either **`$int_if`** is your wireless interface, or in `/etc/pf.conf`, add:

```
air_if = "ath0"
```

and

```
nat on $ext_if from $air_if:network to any -> ($ext_if) static-port
```

Similar for ftp-proxy config, and include **`$air_if`** in your pass rules.

authpf: per user rules

authpf is a *non-interactive* user shell.

- Authenticates users using any ssh supported method
- After authentication, user's IP address is added to the <**`authpf_users`**> table for general rules to apply; in addition, per user and per user group rules are possible
- Table entry destroyed when ssh session terminates; anchor updated

Useful for wireless networks and wired alike.

Basic authpf setup

You need the table

```
table <authpf_users> persist
```

plus these anchors

```
nat-anchor "authpf/*"
rdr-anchor "authpf/*"
binat-anchor "authpf/*"
anchor "authpf/*"
```

Basic authpf setup (cont)

Let users authenticate

```
pass quick on $int_if inet proto { tcp, udp } to $int_if port ssh
```

Other rules could just as easily go in `authpf.rules`

```
pass quick inet proto { tcp, udp } from <authpf_users> to \
     any port $udp_services
pass inet proto tcp from <authpf_users> to any port $client_out
```

Basic authpf setup (cont)

The macro **`$user_ip`** expands to logged-in user's IP address:

```
client_out = "{ ssh, domain, pop3, auth, nntp, http, https }"
udp_services = "{ domain, ntp }"
pass quick inet proto { tcp, udp } from $user_ip to any \
     port $udp_services
pass inet proto tcp from $user_ip to any port $client_out
```

Per user rules

/etc/authpf/users/peter/authpf.rules

```
client_out = "{ domain, http, https }"
pass inet from $user_ip to 192.168.103.84 port 9000
pass quick inet proto { tcp, udp } from $user_ip to \
     any port $client_out
```

/etc/authpf/users/chris/authpf.rules

```
pass from $user_ip os = "OpenBSD" to any
```

Wide open but actually shut

A wireless network open at network level.

Access only to authenticated users

Create an empty authpf config:

`touch /etc/authpf/authpf.conf`.

Open but shut: pf.conf

/etc/pf.conf

```
ext_if = "re0"
int_if = "ath0"
auth_web="192.168.27.20"
dhcp_services = "{ bootps, bootpc }" # DHCP server + client
table <authpf_users> persist
rdr pass on $int_if proto tcp from ! <authpf_users> to any \
    port http -> $auth_web
nat on $ext_if from $localnet to any -> ($ext_if)
nat-anchor "authpf/*"
rdr-anchor "authpf/*"
binat-anchor "authpf/*"
anchor "authpf/*"
pass quick on $int_if inet proto { tcp, udp } to $int_if \
     port dhcp_services
pass quick inet proto { tcp, udp } from $int_if:network to \
     any port domain
pass quick on $int_if inet proto { tcp, udp } to $int_if port ssh
```

Sharing the load: Address pools

Your users have bookmarked your **`$webserver`**, load goes up.

Solution: more servers and a redirection

```
webpool = "{ 192.0.2.214, 192.0.2.215, 192.0.2.216, 192.0.2.217 }"

rdr on $ext_if from any to $webserver port $webports -> $webpool \
        round-robin sticky-address
```

relayd

Problem: machines go down, rdr doesn't compensate.

`relayd` (pre-4.3: `hoststated`) detects host state (up/down) interacts with *tables*, adds/removes hosts

Configured via `pf.conf` and `relayd.conf`

Introduced in OpenBSD 4.1 (renamed in 4.3), actively developed

Basic relayd config

In your `pf.conf`

```
rdr-anchor "hoststated/*"
anchor "relayd/*"
```

Basic relayd config (cont)

In your `relayd.conf`

```
web1="192.0.2.214"
web2="192.0.2.215"
web3="192.0.2.216"
web4="192.0.2.217"
webserver="192.0.2.227"
sorry_server="192.0.2.200"

interval 5 # check hosts every 5 seconds

table <webpool> { $web1, $web2, $web3, $web4, }
table <sorry> { $sorry_server }

redirect "www" {
 listen on $webserver port http
 forward to <webpool> check http "/status.html" code 200 timeout 300
 forward to <sorry> timeout 300 check icmp
}
```

enable with `rc.conf.local` entry

```
relayd_flags="" # for normal use: ""#
```

relayctl

relayctl for interactive **relayd** control

```
$ sudo relayctl show summary
Type      Id    Name                                Avlblty Status
service    0    www                                         down
table      0    webpool                                     active (2 hosts
host       3    192.0.2.217                         0.00%   down
host       2    192.0.2.216                         100.00% up
host       1    192.0.2.215                         0.00%   down
host       0    192.0.2.214                         100.00% up
table      1    sorry                                       active (1 hosts
host       4    192.0.2.200                         100.00% up
```

enable or disable hosts

```
$ sudo relayctl host disable 192.0.2.216
```

```
$ sudo relayctl host enable 192.0.2.216
```

Filtering for services, the NAT version

If routable addresses are not available, you

- select an appropriate RFC1918 address range
- edit your **`webserver`**, **`emailserver`**
- add appropriate redirections

```
rdr on $ext_if proto tcp from any to $ext_if port \
       $webports -> $webserver
rdr on $ext_if proto tcp from any to $ext_if port \
       $email -> $emailserver
```

segment off your DMZ, introduce address pools

Back to the single NATed network

```
webserver = "192.168.2.7"
webports = "{ http, https }"
emailserver = "192.168.2.5"
email = "{ smtp, pop3, imap, imap3, imaps, pop3s }"

rdr on $ext_if proto tcp from any to $ext_if port \
       $webports -> $webserver
rdr on $ext_if proto tcp from any to $ext_if port \
       $email -> $emailserver

pass in on $ext_if proto tcp from any to $webserver port $webports \
   flags S/SA synproxy state
pass in on $ext_if proto tcp from any to $emailserver port $email \
   flags S/SA synproxy state
pass out on $ext_if proto tcp from $emailserver to any port smtp \
   flags S/SA synproxy state
```

Works with or without a separate dmz, but -

Single NAT, web & mail server on the inside: from the inside

Problem: Traffic from the inside does not reach the internal interface

Solutions:

- ’ Split horizon’ DNS, different answer for LAN and elsewhere
- proxying, such as nc (NetCat)
- Moving your servers to a separate DMZ
- special case of redirection and NAT for the local net

Single NAT, web & mail server on the inside: from the inside

Add to the rule set:

```
rdr on $int_if proto tcp from $int_if:network to $ext_if \
       port $webports -> $webserver
rdr on $int_if proto tcp from $int_if:network to $ext_if \
       port $email -> $emailserver
no nat on $int_if proto tcp from $int_if to $int_if:network
nat on $int_if proto tcp from $int_if:network to $webserver \
       port $webports -> $int_if
nat on $int_if proto tcp from $int_if:network to $emailserver \
       port $email -> $int_if
```

Filtering on interface groups

You can configure groups of interfaces, filter on them

```
# ifconfig sis2 group untrusted
```

(or `hostname.sis2`)

Use in your `pf.conf`

```
pass in on untrusted to any port $webports
pass out on egress to any port $webports
```

The power of tags

tag packets incoming, block or pass outgoing based on tags

eg in a net with several NATing access points

```
wifi = "{ 10.0.0.115, 10.0.0.125, 10.0.0.135, 10.0.0.145 }"
pass in on $int_if from $wifi to $wifi_allowed port \
     $wifi_ports tag wifigood
     ...
pass out on $ext_if tagged wifigood
```

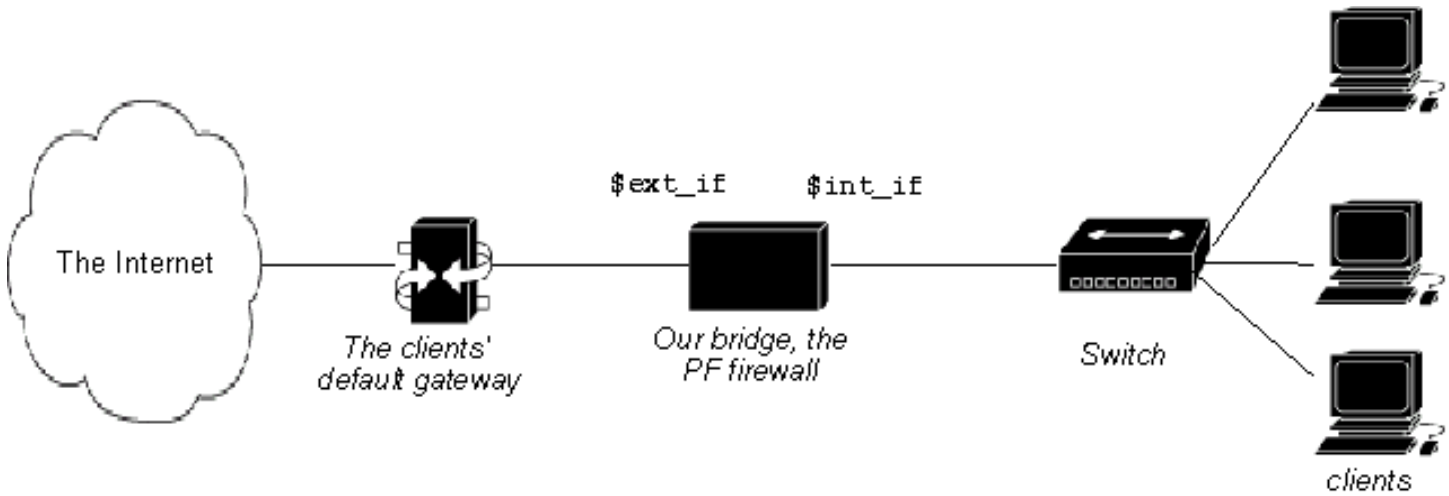
NOTE: tags are sticky - all matching tag rules add their tag

The filtering bridge

Bridge: machine with no IP address of its own, between the Internet and a local network

- Opererates on the Ethernet level
- "Invisible" to the outside world
- Is able to use PF for filtering and nat/rdr

Where does it go?



OpenBSD bridge setup

```
/etc/hostname.ep0

up

/etc/hostname.ep1

up

/etc/bridgename.bridge0

  add ep0 add ep1 blocknonip ep0 blocknonip ep1 up
```

Se also bridge(4), brconfig(8)

FreeBSD bridge setup

Make sure your kernel config has the bridge interface, `/etc/loader.conf`

```
if_bridge_load="YES"
```

Create a bridge:

```
$ sudo ifconfig bridge0 create
```

check your new sysctls

```
$ sudo sysctl net.link.bridge
net.link.bridge.ipfw: 0
net.link.bridge.pfil_member: 1
net.link.bridge.pfil_bridge: 1
net.link.bridge.ipfw_arp: 0
net.link.bridge.pfil_onlyip: 1
```

Check that the interfaces are up, but otherwise unconfigured, then

```
$ sudo ifconfig bridge0 addm ep0 addm ep1 up0
```

make permanent with `/etc/rc.conf` entries:

```
ifconfig_ep0="up"
ifconfig_ep1="up"
cloned_interfaces="bridge0"
ifconfig_bridge0="addm ep0 addm ep1 up"
```

Bridge PF filtering config

/etc/pf.conf

```
ext_if = ep0
int_if = ep1
localnet= "192.0.2.0/24"
webserver = "192.0.2.227"
webports = "{ http, https }"
emailserver = "192.0.2.225"
email = "{ smtp, pop3, imap, imap3, imaps, pop3s }"
nameservers = "{ 192.0.2.221, 192.0.2.223 }"
client_out = "{ ssh, domain, pop3, auth, nntp, http, https, \
                cvspserver, 2628, 5999, 8000, 8080 }"
udp_services = "{ domain, ntp }"
icmp_types = "{ echoreq, unreach }"
set skip on $int_if
block all
pass quick on $ext_if inet proto { tcp, udp } from $localnet to \
     any port $udp_services
pass log on $ext_if inet proto icmp all icmp-type $icmp_types
pass on $ext_if inet proto tcp from $localnet to any port $client_out
pass on $ext_if inet proto { tcp, udp } from any to $nameservers \
     port domain
pass on $ext_if proto tcp from any to $webserver port \
     $webports synproxy state
pass log on $ext_if proto tcp from any to $emailserver port \
     $email synproxy state
```

```
pass log on $ext_if proto tcp from $emailserver to any port \
     smtp synproxy state
```

(knock yourself out)

Handling non-routable addresses from elsewhere

Bar officially unroutable (RFC1918 et al) traffic

```
martians = "{ 127.0.0.0/8, 192.168.0.0/16, 172.16.0.0/12, \
              10.0.0.0/8, 169.254.0.0/16, 192.0.2.0/24, \
              0.0.0.0/8, 240.0.0.0/4 }"

block drop in quick on $ext_if from $martians to any
block drop out quick on $ext_if from any to $martians
```

NOTE: could usefully be rewritten as a table

Directing traffic with altq

ALTQ

ALternate Queueing – uses "queues" for bandwidth allocation, traffic shaping

- class based (cbq) - per cent, (K,M)bytes, *or*
- priority based (PRIQ)
- hierarchical (HFSC)

Syntax:

```
altq on interface type [options ... ] main_queue { sub_q1, sub_q2 ..}
  queue sub_q1 [ options ... ]
  queue sub_q2 [ options ... ]
[...]
pass [ ... ] queue sub_q1
pass [ ... ] queue sub_q2
```

Setting up for ALTQ

OpenBSD: All relevant options are enabled in the GENERIC kernel. No further (non-`pf.conf`) configuration necessary

Setting up for ALTQ: FreeBSD

FreeBSD: The GENERIC kernel does not have ALTQ options enabled. Relevant options are

```
options ALTQ
options ALTQ_CBQ # Class-Based Queuing (CBQ)
options ALTQ_RED # Random Early Detection (RED)
options ALTQ_RIO # RED In/Out
options ALTQ_HFSC # Hierarchical Packet Scheduler (HFSC)
options ALTQ_PRIQ # Priority Queuing (PRIQ)
options ALTQ_NOPCC # Required for SMP buildo
```

Enable the options you need in your kernel config, install the kernel

Setting up for ALTQ: NetBSD

NetBSD: ALTQ available in NetBSD 4.0; The GENERIC kernel does not have ALTQ options enabled. Relevant options are

```
options ALTQ # Manipulate network interfaces' output queues
options ALTQ_CBQ # Class-Based Queuing
options ALTQ_HFSC # Hierarchical Fair Service Curve
options ALTQ_PRIQ # Priority Queuing
options ALTQ_RED # Random Early Detection
```

Enable the options you need in your kernel config, install the kernel

What is your usable bandwidth?

Interfaces report *interface* bandwidth, not *line bandwidth*

Overhead can be single-digit percentage (ethernet) to > 20 % (ADSL)

Set bandwidth to lowest value in the relevant path

ALTQ - prioritizing by traffic type

Daniel Hartmeier's ADSL - prioritizing ACKs to improve up/download over asymmetric link

```
ext_if="kue0"

altq on $ext_if priq bandwidth 100Kb queue { q_pri, q_def }
queue q_pri priority 7
queue q_def priority 1 priq(default)

pass out on $ext_if proto tcp from $ext_if to any flags S/SA \
        keep state queue (q_def, q_pri)

pass in  on $ext_if proto tcp from any to $ext_if flags S/SA \
        keep state queue (q_def, q_pri)
```

See http://www.benzedrine.cx/ackpri.html

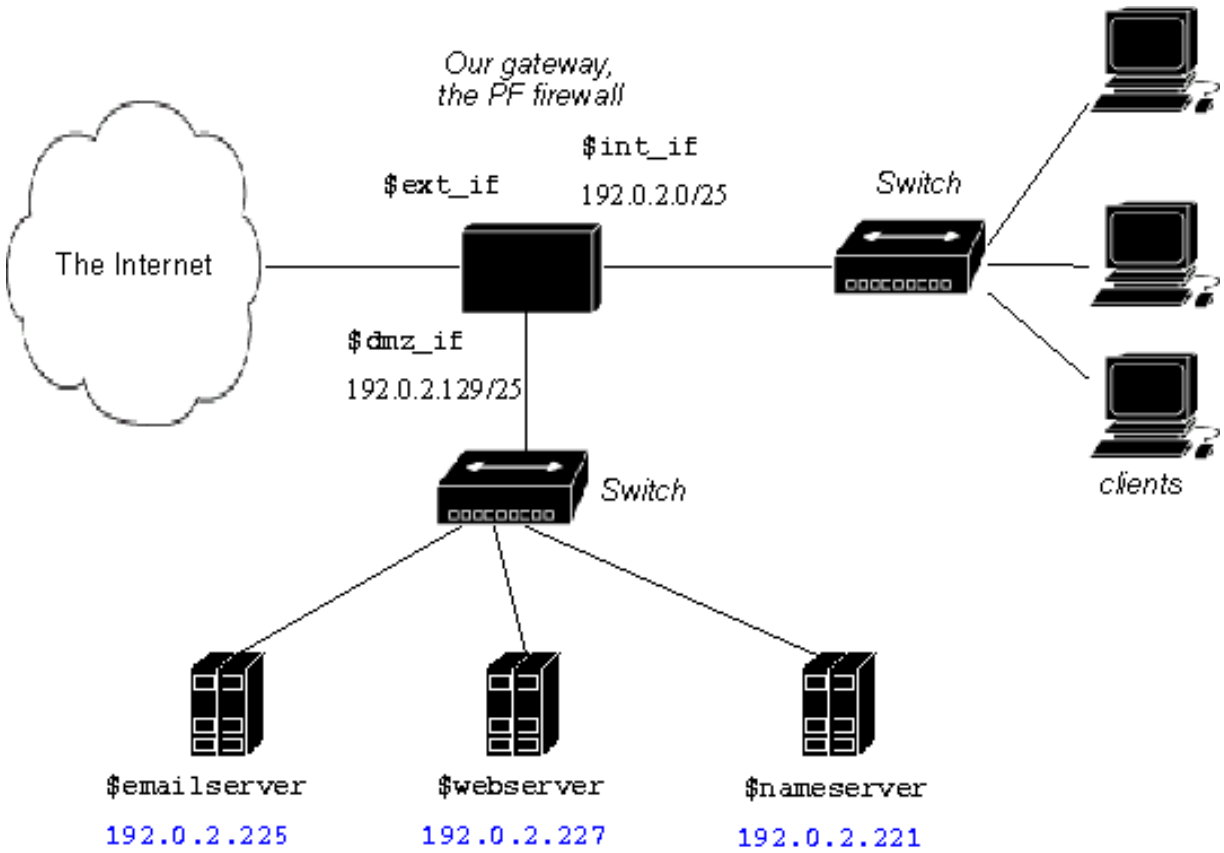
ALTQ - allocation by percentage

example swiped from unix.se:

```
altq on $ext_if cbq bandwidth 640Kb queue { def, ftp, udp, \
    http, ssh, icmp }
    queue def bandwidth 18% cbq(default borrow red)
    queue ftp bandwidth 10% cbq(borrow red)
    queue udp bandwidth 30% cbq(borrow red)
    queue http bandwidth 20% cbq(borrow red)
    queue ssh bandwidth 20% cbq(borrow red) { ssh_interactive, \
          ssh_bulk }
          queue ssh_interactive priority 7 bandwidth 20%
          queue ssh_bulk priority 0 bandwidth 80%
    queue icmp bandwidth 2% cbq

pass log quick on $ext_if proto tcp from any to any port ssh \
    flags S/SA keep state queue (ssh_bulk, ssh_interactive)
pass in quick on $ext_if proto tcp from any to any port ftp \
    flags S/SA keep state queue ftp
pass in quick on $ext_if proto tcp from any to any port www \
    flags S/SA keep state queue http
pass out on $ext_if proto udp all keep state queue udp
pass out on $ext_if proto icmp all keep state queue icmp
```

Queueing for a DMZ



Queueing for a DMZ: rules part 1

```
total_ext = 2Mb
total_dmz = 100Mb
altq on $ext_if cbq bandwidth $total_ext queue { ext_main, ext_web, \
        ext_udp, ext_mail, ext_ssh }
queue ext_main bandwidth 25% cbq(default borrow red) { ext_hi, ext_lo }
     queue ext_hi priority 7 bandwidth 20%
     queue ext_lo priority 0 bandwidth 80%
queue ext_web bandwidth 25% cbq(borrow red)
queue ext_udp bandwidth 20% cbq(borrow red)
queue ext_mail bandwidth 30% cbq(borrow red)
altq on $dmz_if cbq bandwidth $total_dmz queue { ext_dmz, dmz_main, \
      dmz_web, dmz_udp, dmz_mail }
queue ext_dmz bandwidth $total_ext cbq(borrow red) queue { ext_dmz_web,
     ext_dmz_udp, ext_dmz_mail }
     queue ext_dmz_web bandwidth 40% priority 5
     queue ext_dmz_udp bandwidth 10% priority 7
     queue ext_dmz_mail bandwidth 50% priority 3
queue dmz_main bandwidth 25Mb cbq(default borrow red) queue { dmz_main_h
     dmz_main_lo }
     queue dmz_main_hi priority 7 bandwidth 20%
     queue dmz_main_lo priority 0 bandwidth 80%
queue dmz_web bandwidth 25Mb cbq(borrow red)
queue dmz_udp bandwidth 20Mb cbq(borrow red)
queue dmz_mail bandwidth 20Mb cbq(borrow red)
```

Queueing for a DMZ: rules part 2

```
pass in on $ext_if proto { tcp, udp } from any to $nameservers \
     port domain queue ext_udp
pass in on $int_if proto { tcp, udp } from $localnet to $nameservers \
     port domain
pass out on $dmz_if proto { tcp, udp } from any to $nameservers port \
     domain queue ext_dmz_udp
pass out on $dmz_if proto { tcp, udp } from $localnet to $nameservers \
     port domain queue dmz_udp
pass in on $ext_if proto tcp from any to $webserver port $webports \
     queue ext_web
pass in on $int_if proto tcp from $localnet to $webserver port $webports
pass out on $dmz_if proto tcp from any to $webserver port $webports \
     queue ext_dmz_web
pass out on $dmz_if proto tcp from $localnet to $webserver port $webport
     queue dmz_web
pass in log on $ext_if proto tcp from any to $mailserver port smtp
pass in log on $ext_if proto tcp from $localnet to $mailserver port smtp
pass in log on $int_if proto tcp from $localnet to $mailserver port $ema
pass out log on $dmz_if proto tcp from any to $mailserver port smtp \
     queue ext_mail
pass in on $dmz_if from $mailserver to any port smtp queue dmz_mail
pass out log on $ext_if proto tcp from $mailserver to any port smtp \
     queue ext_dmz_mailt
```

overloading to a tiny queue

```
pass log quick on $ext_if proto tcp from any to any port ssh flags S/SA
keep state queue (ssh_bulk, ssh_interactive)
```

becomes

```
pass log quick on $ext_if proto tcp from any to any port ssh flags S/SA
keep state (max-src-conn 15, max-src-conn-rate 5/3, \
     overload <bruteforce> flush global) \
     queue (ssh_bulk, ssh_interactive)
```

where

```
queue smallpipe bandwidth 1% cbq
```

and

```
pass inet proto tcp from <bruteforce> to any \
     port $tcp_services queue smallpipe
```

ALTQ - handling unwanted traffic

```
altq on $ext_if cbq queue { q_default q_web q_mail }

   queue q_default cbq(default)
   queue q_web (...)

    ## all mail limited to 1Mb/sec
    queue q_mail bandwidth 1Mb { q_mail_windows }
    ## windows mail limited to 56Kb/sec
    queue q_mail_windows bandwidth 56Kb

    pass in quick proto tcp from any os "Windows" to $ext_if \
    port 25 keep state queue q_mail_windows
    pass in quick proto tcp from any to $ext_if port 25 \
    label "smtp" keep state queue q_mail
```

" I can't believe I didn't see this earlier. Oh, how sweet. ...
Already a huge difference in my load. Bwa ha ha. "

Randal L. Schwartz, http://use.perl.org/~merlyn/journal/17094

CARP and pfsync

Common Address Redundancy Protocol (CARP)

- Introduced with OpenBSD 3.5
- Patent free alternative to VRRP (RFC 2281, 3768, patent owners: Cisco, IBM, Nokia)
- Firewall/server redundancy
- Virtual network interface for automatic failover

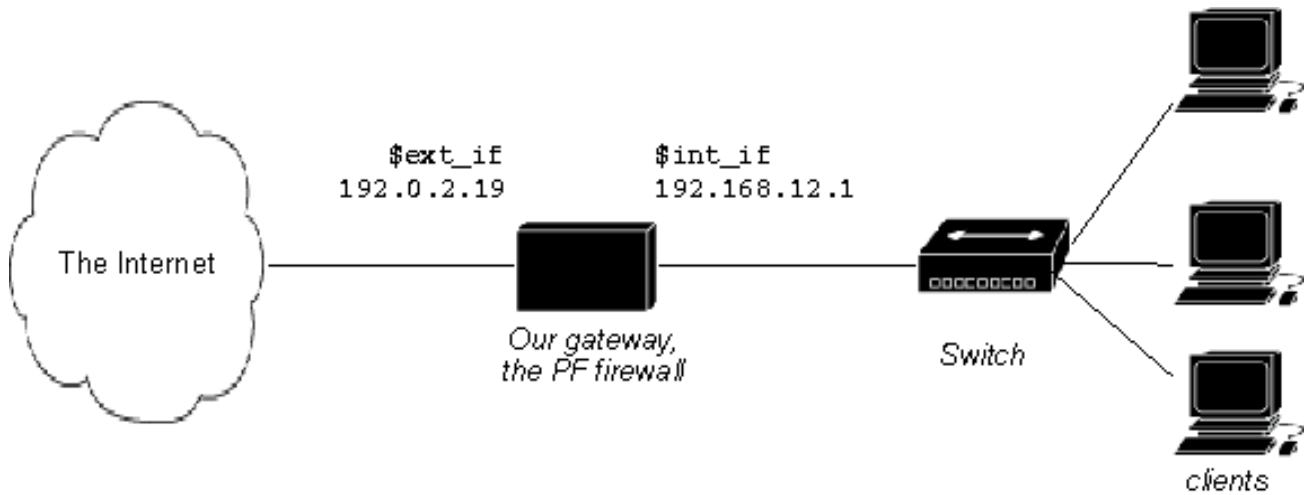
pfsync

- Virtual network interface (assigned to physical interface)
- Handles syncronization between PF firewalls (in advance of failover)

Best reference: http://www.countersiege.com/doc/pfsync-carp/

CARP: project spec

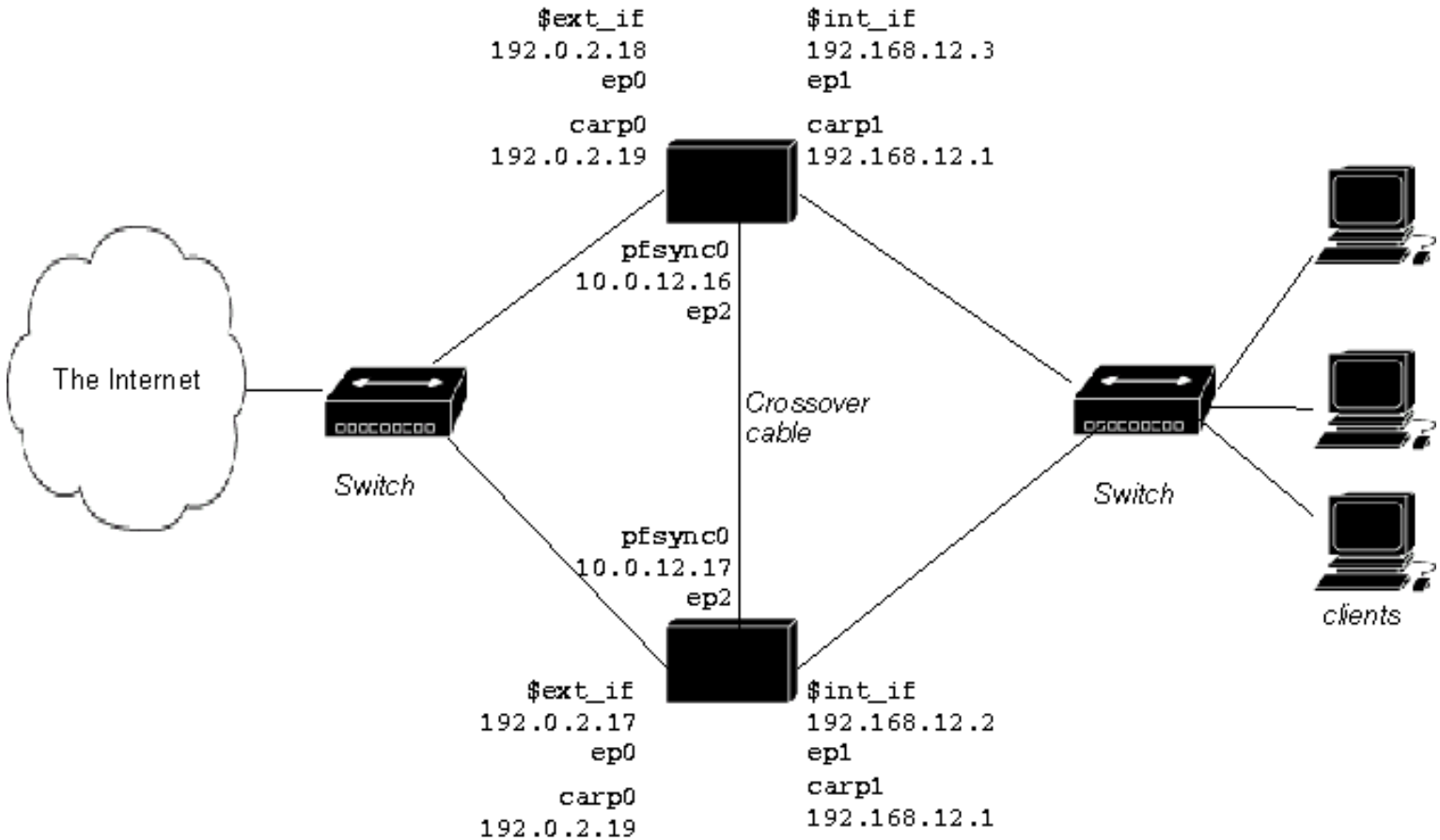
Our network -



becomes (. . .)

CARP: project spec cont'd

Our network becomes



CARP: project spec cont'd

Our network should

- Keep functioning much the same way as it did earlier
- Have better availability with no noticeable downtime
- Experience graceful failover with no interruption of active connections

Tall order, huh?

Is your system CARP ready?

- *OpenBSD*: GENERIC kernel comes with `carp` and `pfsync` devices compiled in
- *FreeBSD*: GENERIC kernel does not have `carp` or `pfsync` devices enabled, must be enabled in kernel config and compiled in
- *NetBSD*: GENERIC kernel does not have `carp`, needs to be enabled and compiled in. NetBSD does not support `pfsync`

Setting up CARP

You need the sysctls

```
$ sysctl net.inet.carp.allow
net.inet.carp.allow=1
```

```
$ sysctl net.inet.carp
net.inet.carp.allow=1
net.inet.carp.preempt=0
net.inet.carp.log=0
net.inet.carp.arpbalance=0
```

to let the magic work, we need

```
$ sudo sysctl net.inet.carp.preempt=1
```

CARP: ifconfig

on the *master*

```
$ sudo ifconfig carp0 192.0.2.19 vhid 1
$ sudo ifconfig carp1 192.168.1.1 vhid 2
```

on the *backup*

```
$ sudo ifconfig carp0 192.0.2.19 vhid 1 advskew 100
$ sudo ifconfig carp1 192.168.1.1 vhid 2 advskew 100
```

the master announces every (1 + 0/256) seconds

the backup announces every (1 + 100/256) seconds

Note Multicast. Use **`carppeer`** option for unicast

Also see Henning Brauer's notes
(http://bulabula.org/carp-and-stp-meet-switch-security.html)

pfsync

Use a physically separate net (or crossover cable)

```
$ sudo ifconfig pfsync0 syncpeer 10.0.12.16 syncdev ep2
```

What happens to the rule set?

Pass CARP traffic on the appropriate interfaces

```
pass on $carpdevs proto carp keep state
```

Pass pfsync traffic on the appropriate interfaces

```
pass on $syncdev proto pfsync
```

Some traffic doesn't make sense to fail over

```
pass in on $int_if from $ssh_allowed to self keep state (no-sync)
```

PF sees the traffic on the physical interface

carp config example

master:

```
sysctl.conf

net.inet.carp.preempt=1

hostname.carp0

pass mekmitasdigoat 192.0.2.19 vhid 1

hostname.carp1

pass mekmitasdigoat 192.168.1.1 vhid 2
```

backup:

```
sysctl.conf

net.inet.carp.preempt=1

hostname.carp0

pass mekmitasdigoat 192.0.2.19 vhid 1 advskew 100

hostname.carp1

pass mekmitasdigoat 192.168.1.1 vhid 2 advskew 100
```

Carp ruleset

```
ext_if=sis0
int_if=sis1
carpdevs="sis0 sis1"
int_carp=carp1
ext_carp=carp0
nat on $ext_if from $int_if:network -> ($ext_carp)
```

Making your network troubleshooting friendly

Mainly concerns the *Internet Control Message Protocol* (*ICMP*)

- *Control Messages* (transmission parameters, packet sizes, routing, *path MTU discovery*)
- Mid 1990s *Ping of Death* scare - ICMP ’just evil’, ’a necessary evil’
- Modern OSes not vulnerable; people are still scared

Then, do we let it all through?

```
pass inet proto icmp from any to any
```

Pro: makes debugging easier

Con: may reveal too much about your network

Note: some ICMP traffic piggybacks on **`keep state`**

The easy way out: The buck stops here

```
pass inet proto icmp icmp-type $icmp_types from $int_if:network \
          to any keep state
pass inet proto icmp icmp-type $icmp_types from any to $ext_if \
          keep state
```

Lets your traffic pass; hides the details of your local net

Letting ping through

ping: icmp - some packet types

```
icmp_types = "echoreq"
 # [...]
pass inet proto icmp all icmp-type $icmp_types keep state
```

Helping traceroute

traceroute needs a bit of help, but uses a fixed formula:

```
# allow out the default range for traceroute(8):
# "base+nhops*nqueries-1" (33434+64*3-1)
pass out on $ext_if inet proto udp from any to any \
     port 33433 >< 33626 keep state
```

This is the stuff you find in list archives - openbsd-misc (e. g. http://marc.info/)

Note: Unix traceroute uses UDP by default; Microsoft uses ICMP ECHO (like unix with -I)

Path MTU discovery

Negotiating critical communication parameters

- Sending progressively larger packets with *"Do not fragment"* flag set
- Expecting ICMP reply *"type 3, code 4"*

  -> "destination unreachable", "fragmentation needed, Do not fragment flag set"

Path MTU discovery (cont'd)

Suggested rule set addition:

```
icmp_types = "{ echoreq, unreach }"

pass inet proto icmp all icmp-type $icmp_types keep state
```

See RFC792, RFC950, RFC1191, RFC1256, RFC2521, RFC2765 (ICMP4),

RFC1885, RFC2463, RFC2466 (ICMP6)

ietf.org (http://www.ietf.org) or faqs.org (http://www.faqs.org) + PF man pages

Logging

Keyword "log" in the rules to be logged

`/etc/pf.conf`

```
pass out log from <client> to any port $email \
     label client-email keep state
```

Logs in binary, tcpdump(8) readable format

NOTE: **`log`** logs only initial packet, use **`log (all)`** to log all matching packets

OpenBSD 4.1 onwards: cloneable `pflog`, rules can log to specific interface:

```
pass log (all, to pflog2) inet proto tcp from $mailserver \
     to any port smtp
```

`pflog` interfaces created with ifconfig **`pflogN create`**

Taking a peek with tcpdump

PF logs via the `pflogN` interfaces, pflogd collects data, stores in `/var/log/pflog`.

```
peter@skapet:~$ sudo tcpdump -n -e -ttt -i pflog0
tcpdump: WARNING: pflog0: no IPv4 address assigned
tcpdump: listening on pflog0, link-type PFLOG
Feb 16 16:43:20.152187 rule 0/(match) block in on ep0: 194.54.59.189.2559 >
194.54.107.19.139: [|tcp] (DF)
Feb 16 16:48:26.073244 rule 27/(match) pass in on ep0: 61.213.167.236 >
194.54.107.19: icmp: echo request
Feb 16 16:49:09.563448 rule 0/(match) block in on ep0: 61.152.249.148.80 >
194.54.107.19.55609: [|tcp]
Feb 16 16:49:14.601022 rule 0/(match) block in on ep0: 194.54.59.189.3056 >
194.54.107.19.139: [|tcp] (DF)
Feb 16 16:53:10.110110 rule 0/(match) block in on ep0: 68.194.177.173 >
194.54.107.19: [|icmp]
Feb 16 16:55:54.818549 rule 27/(match) pass in on ep0: 61.213.167.237 >
194.54.107.19: icmp: echo request
Feb 16 16:57:55.577782 rule 27/(match) pass in on ep0: 202.43.202.16 >
194.54.107.19: icmp: echo request
```

tcpdump is your friend

tcpdump has PF smarts for pflog interfaces, such as

```
tcpdump -n -e -ttt -i pflog0 inbound and action block and on wi0
```

or

```
$ sudo tcpdump -n -ttt -i pflog0 port domain
tcpdump: WARNING: pflog0: no IPv4 address assigned
tcpdump: listening on pflog0, link-type PFLOG
Sep 30 14:27:41.260190 212.5.66.14.53 > 194.54.107.19.53:[|domain]
Sep 30 14:27:41.260253 212.5.66.14.53 > 194.54.107.19.53:[|domain]
Sep 30 14:27:41.260267 212.5.66.14.53 > 194.54.107.19.53:[|domain]
Sep 30 14:27:41.260638 194.54.107.19.53 > 212.5.66.14.53:[|domain]
Sep 30 14:27:41.260798 194.54.107.19.53 > 212.5.66.14.53:[|domain]
Sep 30 14:27:41.260923 194.54.107.19.53 > 212.5.66.14.53:[|domain]
```

Matching log data to your rule set

`pflog` log data include rule number matched *in the loaded rule set*

```
$ sudo tcpdump -nettti pflog0
tcpdump: WARNING: pflog0: no IPv4 address assigned
tcpdump: listening on pflog0, link-type PFLOG
Sep 13 15:26:52.122002 rule 17/(match) pass in on epic0: 91.143.126.48.4
Sep 13 15:28:02.771442 rule 12/(match) pass in on epic0: 194.54.107.19.8
Sep 13 15:28:02.773958 rule 10/(match) pass in on epic0: 194.54.107.19.8
Sep 13 15:29:27.882888 rule 10/(match) pass in on epic0: 194.54.107.19.2
Sep 13 15:29:28.394320 rule 12/(match) pass in on epic0: 194.54.107.19.2
```

match to **`pfctl -vvsr`** output

```
$ sudo pfctl -vvsr
@0 scrub in all fragment reassemble
  [ Evaluations: 6116699   Packets: 3069556   Bytes: 646214426   States:
  [ Inserted: uid 0 pid 2006 ]
@0 block return log all
  [ Evaluations: 102723    Packets: 2539      Bytes: 269448      States:
  [ Inserted: uid 0 pid 2006 ]
@1 block return log quick from <bruteforce:1> to any
  [ Evaluations: 102723    Packets: 40        Bytes: 2384        States:
  [ Inserted: uid 0 pid 2006 ]
@2 anchor "ftp-proxy/*" all
  [ Evaluations: 102683    Packets: 28044     Bytes: 22617668    States:
  [ Inserted: uid 0 pid 2006 ]
```

Log to syslog

you can log to syslog, local or remote

NOTE: potential for huge amounts of data

disable local pflog via `rc.conf.local`

```
pflogd_flags="-f /dev/null"
```

define log in `syslog.conf`

```
pflogd_flags="-f /dev/null"
```

start it all (or put in `rc.local`)

```
$ sudo nohup tcpdump -lnettti pflog0 | logger -t pf -p local2.info &
```

Statistics via labels

label creates counters for statistics

```
pass log proto { tcp, udp } from any to $emailserver port smtp \
     label "mail-in"
pass log proto { tcp, udp } from $emailserver to any port smtp \
     label "mail-out"
```

shows up with **`pfctl -vs rules`**, human readable:

```
$ pfctl -vs rules
pass inet proto tcp from any to 192.0.2.225 port = smtp flags S/SA keep
[ Evaluations: 1664158 Packets: 1601986 Bytes: 763762591 States: 0 ]
[ Inserted: uid 0 pid 24490 ]
pass inet proto tcp from 192.0.2.225 to any port = smtp flags S/SA keep
[ Evaluations: 2814933 Packets: 2711211 Bytes: 492510664 States: 0 ]
[ Inserted: uid 0 pid 24490 ]
```

or to feed to a script:

```
$ sudo pfctl -vsl
mail-in 1664158 1601986 763762591 887895 682427415 714091 81335176
mail-out 2814933 2711211 492510664 1407278 239776267 1303933 252734397
```

label, evaluations, packets passed, bytes passed, packets in, bytes in, packets out, bytes out

$variable label names

For even better (finer grained) statistics, you can use variables in label names:

`$if` - The interface.

`$srcaddr` - The source IP address.

`$dstaddr` - The destination IP address.

`$srcport` - The source port specification.

`$dstport` - The destination port specification.

`$proto` - The protocol name.

`$nr` - The rule number.

These expand at ruleset load time

$variable label names: example

```
pass proto tcp from $client1 to $mail_servers port $mail_services \
     label "$srcaddr"
pass proto tcp from $client1 to any port $web_services label "$srcaddr"
```

Accumulate periodically using **`pfctl -vslz`** (the **`z`** reset counters), feed to database

Keeping an eye on things with pftop

```
pfTop: Up State 1-21/67, View: default, Order: none, Cache: 10000

PR   DIR SRC                  DEST                  STATE   AGE   EXP  PK
tcp  Out 194.54.103.89:3847   216.193.211.2:25       9:9     28    67
tcp  In  207.182.140.5:44870  127.0.0.1:8025         4:4     15 86400
tcp  In  207.182.140.5:36469  127.0.0.1:8025        10:10   418    75   8
tcp  In  194.54.107.19:51593  194.54.103.65:22       4:4    146 86395   1
tcp  In  194.54.107.19:64926  194.54.103.65:22       4:4    193 86243   1
tcp  In  194.54.103.76:3010   64.136.25.171:80       9:9    154    59
tcp  In  194.54.103.76:3013   64.136.25.171:80       4:4      4 86397
tcp  In  194.54.103.66:3847   216.193.211.2:25       9:9     28    67
tcp  Out 194.54.103.76:3009   64.136.25.171:80       9:9    214     0
tcp  Out 194.54.103.76:3010   64.136.25.171:80       4:4     64 86337
udp  Out 194.54.107.18:41423  194.54.96.9:53         2:1     36     0
udp  In  194.54.107.19:58732  194.54.103.66:53       1:2     36     0
udp  In  194.54.107.19:54402  194.54.103.66:53       1:2     36     0
udp  In  194.54.107.19:54681  194.54.103.66:53       1:2     36     0
```

New in 4.5: pflow(4) and pflow state option

pflow(4) pseudo-device exports netflow v5 data

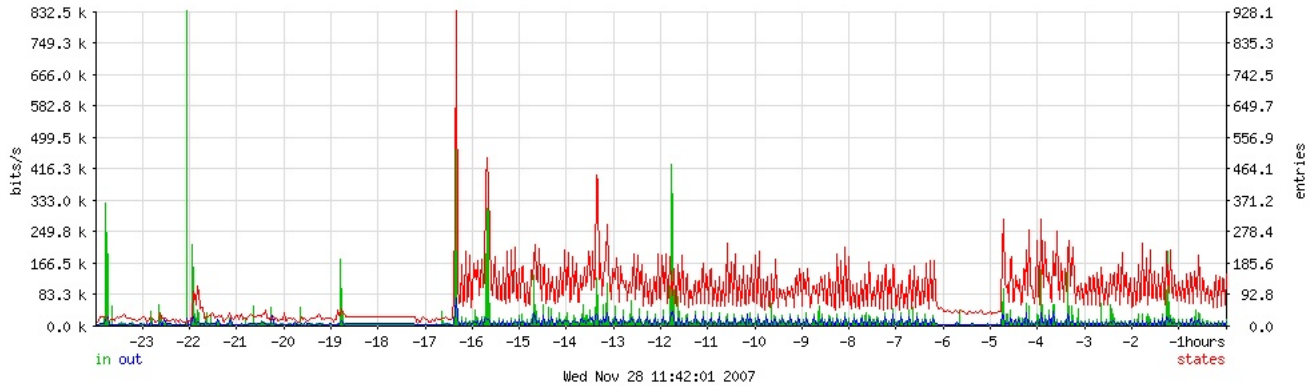
`/etc/pf.conf`

```
pass out log from <client> to any port $email \
label client-email keep state (pflow)
```

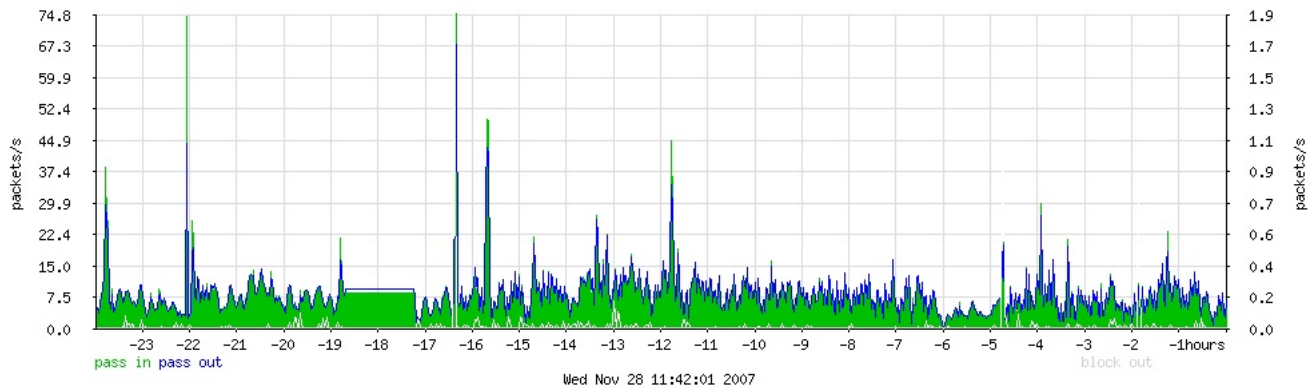
Graph your traffic: pfstat

`pfstat`, collects PF log statistics and graphs (nice pictures for the suits!)

State table entries, last 24 hours:



Packets, last 24 hours:



Other log tools you may want to look into

`net/pfflowd`, collects PF log data, converts to Cisco *NetFlow*™ for further processing (also see `flowd` by the same developer)

SNMP `net/net-snmp`, PF-Related SNMP MIBs from Joel Knight's site, http://www.packetmischief.ca/openbsd/snmp

Good logs for good debugging

Stay in control, understand your logs

Be selective, tweak for debugging

Stay in control

Getting your setup just right

In general *the defaults are sane*

Most tunables can usually be left alone

But you need to know what they do

`man pf.conf` is your friend

block-policy

`block-policy`: the default response to blocked connections

`drop` - drop without return, or

`return` - `Connection refused`, `Destination unreachable`, etc

The default is to drop, to play nicely (ie Go away!):

```
set block-policy return
```

skip

Disable filtering for an interface. Typical use:

```
set skip on lo0
```

(filtering on loopback almost never makes sense)

state-policy

Sets how packets match with state table entries.

possible values: **`floating`** and **`if-bound`**

default is **`floating`**, can be overridden for specific rules, ie

```
pass out on egress inet proto tcp to any port $allowed \
     modulate state (if-bound)
```

Recommendation: leave as is

state-defaults (new in 4.5)

Sets default state options for rules

```
set state-defaults pflow
```

timeout

Cluster of timeout related values for the state table entries

```
$ sudo  pfctl -s timeouts
tcp.first                   120s
tcp.opening                  30s
tcp.established           86400s
tcp.closing                 900s
tcp.finwait                  45s
tcp.closed                   90s
tcp.tsdiff                   30s
udp.first                    60s
udp.single                   30s
udp.multiple                 60s
icmp.first                   20s
icmp.error                   10s
other.first                  60s
other.single                 30s
other.multiple               60s
frag                         30s
interval                     10s
adaptive.start             6000 states
adaptive.end              12000 states
src.track                     0s
```

you can set these individually *if you know what you are doing*.

limit

Set the sizes of memory pools

```
$ sudo pfctl -sm
states        hard limit    10000
src-nodes     hard limit    10000
frags         hard limit     5000
tables        hard limit     1000
table-entries hard limit   200000
```

You can adjust those in `pf.conf`,

```
set limit states 25000
set limit table-entries 300000
```

or

```
set limit { states 25000, src-nodes 25000, table-entries 300000 }
```

Note: limited by available kernel memory

debug

Set the debug info level, possible values **`none`**, **`urgent`** (default), **`misc`**, **`loud`**

useful when debugging, use **`set debug loud`** or use pfctl:

```
$sudo pfctl -x loud
$ tail -f /var/log/messages
Oct 4 11:41:11 skapet /bsd: pf_map_addr: selected address 194.54.107.19
Oct 4 11:41:15 skapet /bsd: pf: loose state match: TCP 194.54.107.19:25
158.36.191.135:62458 [lo=3178647045 high=3178664421 win=33304 modulator=
[lo=3111401744 high=3111468309 win=17376 modulator=0 wscale=0] 9:9 R seq
(3178647044) ack=3111401744 len=0 ackskew=0 pkts=9:12
Oct 4 11:41:15 skapet /bsd: pf: loose state match: TCP 194.54.107.19:25
158.36.191.135:62458 [lo=3178647045 high=3178664421 win=33304 modulator=
[lo=3111401744 high=3111468309 win=17376 modulator=0 wscale=0] 10:10 R s
(3178647044) ack=3111401744 len=0 ackskew=0 pkts=10:12
Oct 4 11:42:24 skapet /bsd: pf_map_addr: selected address 194.54.107.19
```

Lots of info, enough data for auto-DOS if you're not careful

ruleset-optimiation

Enable or tweak the ruleset optimizer, possible values **`none`**, **`basic`** (default), **`profile`**

```
set ruleset-optimization basic
```

or

```
$ sudo pfctl -o basic
```

With the **`basic`** setting the optimizer

- Removes duplicate rules
- Removes rules that are subsets of other rules
- Merges rules into tables if appropriate
- Changes rules order to improve performance

optimization

Choose profile for state-timeout handling, possible values **`normal`** (default), **`high-latency`**, **`satellite`**, **`aggressive`**, **`conservative`**.

Recommendation: keep the default unless you know you need something else

`high-latency`, **`satellite`**: states expire slowly (synonyms)

`aggressive`: states expire faster, saves memory, could drop idle connections early

`conservative`: states expire very slowly, uses more memory

Hygiene: scrub and antispoof

`scrub`:

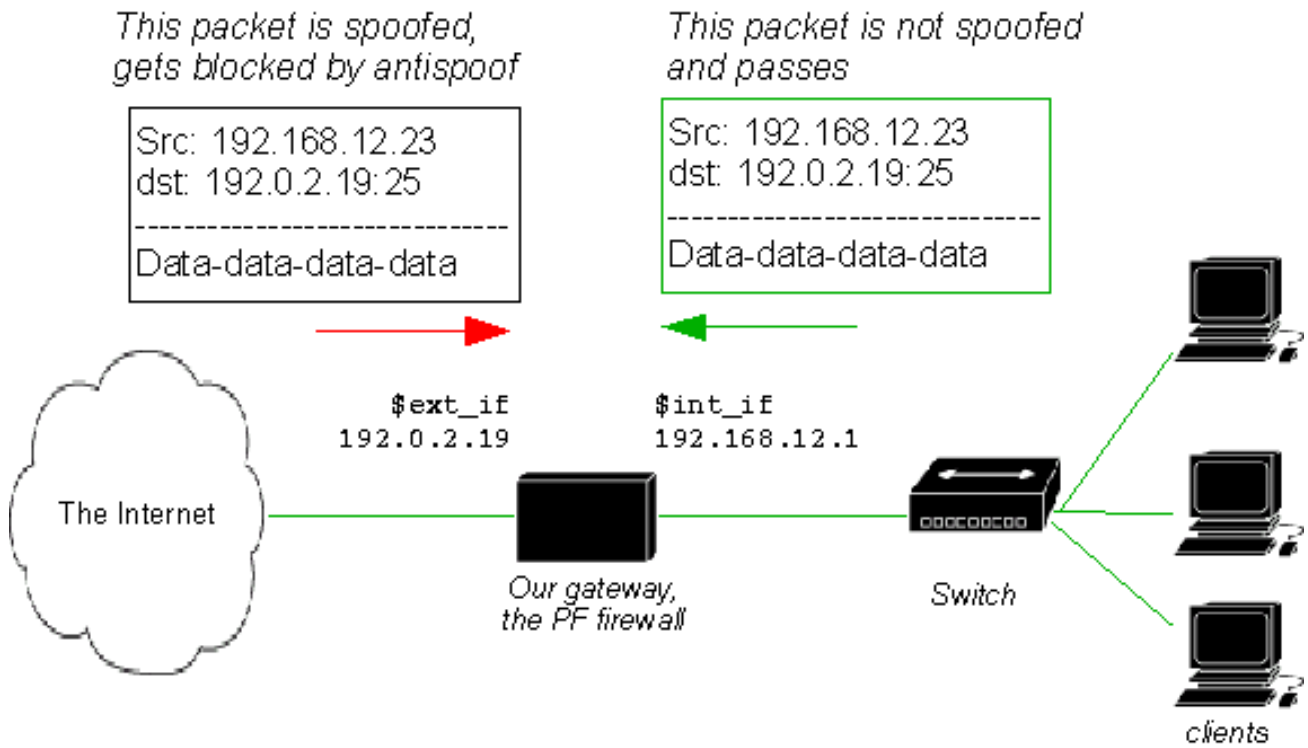
normalization, defragmentation

```
scrub in all
```

`antispoof`:

"this packet should not be here"

drop packets from the wrong network

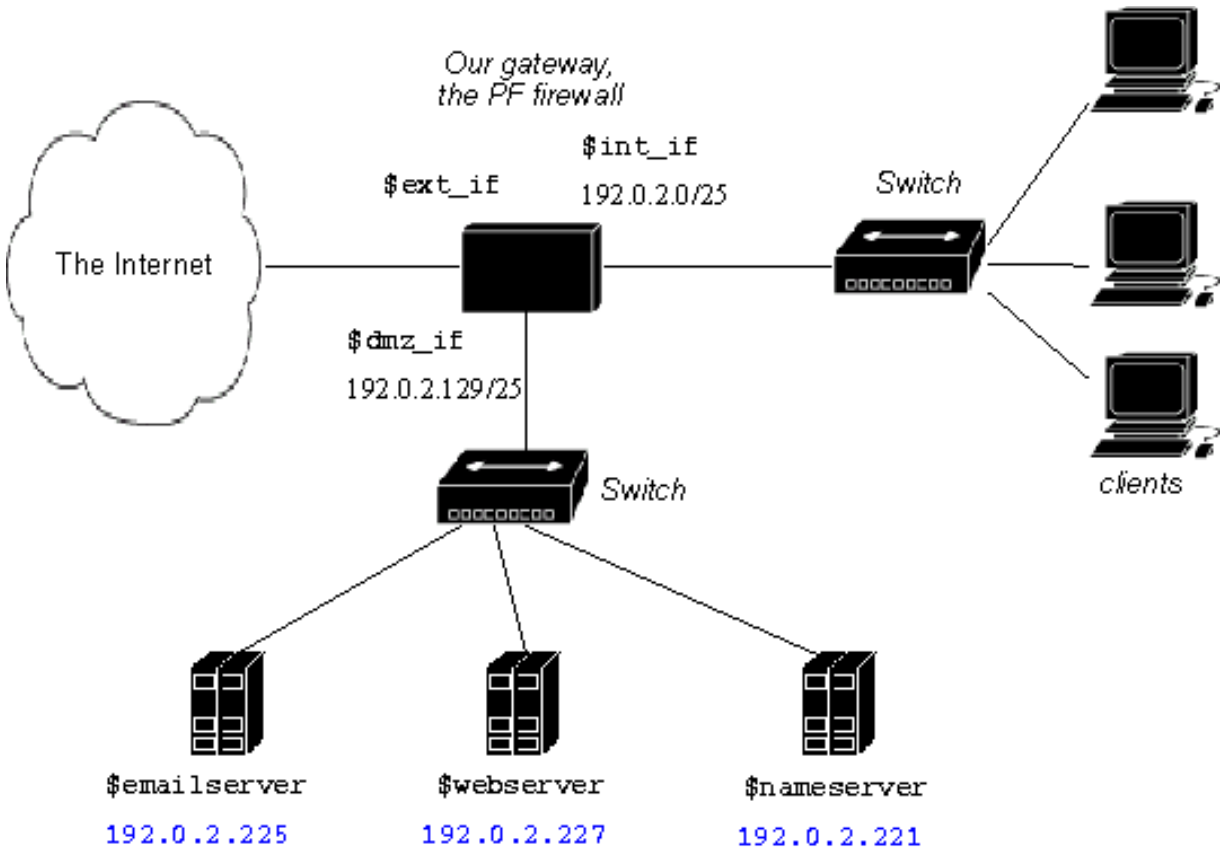



```
antispoof for $ext_if
antispoof for $int_if
```

Testing your setup

Go back to the specification. Does the config do what it's supposed to?

Consider this network again:



Specification (possibly incomplete)

- Default deny (aka **`block all`**)
- Allow access from anywhere to DMZ hosts for certain services
- Allow access from local net to DMZ, local net to anywhere port **`$client_out`**
- Allow access from DMZ to anywhere for some services.

Your task: Test that this works, valid traffic passes.

Test stuff that shouldn't work too, make sure it breaks.

Debugging your setup

If your setup does not conform to spec - *debug*

First thing to check: is PF enabled?

```
$ sudo pfctl -si | grep Status
Status: Enabled for 20 days 06:28:24             Debug: Loud
```

Does enabling/disabling PF make a difference?

Do a ruleset walkthrough, based on **`pfctl -s rules`** output

```
$ sudo pfctl -sr
scrub in all fragment reassemble
block return log all
block return log quick from <bruteforce> to any
anchor "ftp-proxy/*" all
```

Debugging some more

Get working with **`pfctl -vvsr`** output

```
$ sudo pfctl -vvsr
@0 scrub in all fragment reassemble
  [ Evaluations: 6116699   Packets: 3069556   Bytes: 646214426   States:
  [ Inserted: uid 0 pid 2006 ]
@0 block return log all
  [ Evaluations: 102723    Packets: 2539      Bytes: 269448      States:
  [ Inserted: uid 0 pid 2006 ]
@1 block return log quick from <bruteforce:1> to any
  [ Evaluations: 102723    Packets: 40        Bytes: 2384        States:
  [ Inserted: uid 0 pid 2006 ]
@2 anchor "ftp-proxy/*" all
  [ Evaluations: 102683    Packets: 28044     Bytes: 22617668    States:
  [ Inserted: uid 0 pid 2006 ]
```

Trace your packet's path through the logic in the *loaded* rule set.

What's the *last* matching rule? Any **`quick`** rules to watch for?

Debug - use tcpdump

Use tcpdump to check for traffic

```
$ sudo tcpdump -nvvvpi xl0 tcp and not port ssh and not port smtp
tcpdump: listening on xl0, link-type EN10MB
16:23:33.351341 194.54.107.19.58679 > 80.79.54.23.80: S [tcp sum ok] 292
<mss 1460,nop,wscale 1,nop,nop,timestamp 166143589 0,sackOK,eol> (DF) (
16:23:33.434469 80.79.54.23.80 > 194.54.107.19.58679: S [tcp sum ok] 227
win 16384 <mss 1460,nop,nop,sackOK,nop,wscale 0,nop,nop,timestamp 313853
16:23:33.434813 194.54.107.19.58679 > 80.79.54.23.80: . [tcp sum ok] 1:1
166143673 3138530904> (DF) (ttl 63, id 14603, len 52)
16:23:33.436111 194.54.107.19.58679 > 80.79.54.23.80: P 1:242(241) ack 1
166143674 3138530904> (DF) (ttl 63, id 14604, len 293)
16:23:33.698605 80.79.54.23.80 > 194.54.107.19.58679: . 1:1449(1448) ack
3138530905 166143674> (DF) (ttl 48, id 51709, len 1500)
16:23:33.700022 80.79.54.23.80 > 194.54.107.19.58679: P 1449:2533(1084)
3138530905 166143674> (DF) (ttl 48, id 45465, len 1136)
```

Find out exactly where your logic breaks

Have fun!

Conclusion: Know your network, stay in control.

With OpenBSD and PF, you have *the best toolset* for taking control and do things the smart way. We've only scraped the surface here.

This is when the fun part starts, it's up to you!

If you enjoyed this: Support OpenBSD!

Buy OpenBSD CDs and other items, donate!

OpenBSD.org Orders Page: http://www.openbsd.org/orders.html

OpenBSD Donations Page: http://www.openbsd.org/donations.html.

OpenBSD Hardware Wanted Page: http://www.openbsd.org/want.html.

Remember: Free software takes real work and real money to develop and maintain.

If you want to support me, buy the book (http://www.nostarch.com/pf.htm), PDFs are even better!

References

Peter N. M. Hansteen, The Book of PF (http://www.nostarch.com/pf.htm), No Starch Press December 2007

My online PF tutorial, several formats http://home.nuug.no/~peter/pf/

OpenBSDs web http://www.openbsd.org/

OpenBSDs FAQ, http://www.openbsd.org/faq/index.html

PF User Guide http://www.openbsd.org/faq/pf/index.html

Daniel Hartmeier's PF pages, http://www.benzedrine.cx/pf.html

Daniel Hartmeier: Design and Performance of the OpenBSD Stateful Packet Filter (pf), http://www.benzedrine.cx/pf-paper.html (presented at Usenix 2002)

Nate Underwood: HOWTO: Transparent Packet Filtering with OpenBSD, http://ezine.daemonnews.org/200207/transpfobsd.html

Randal L. Schwartz: Monitoring Net Traffic with OpenBSD's Packet Filter, http://www.samag.com/documents/s=9053/sam0403j/0403j.htm

Unix.se: Brandvägg med OpenBSD, http://unix.se/Brandv%E4gg_med_OpenBSD

Randal L. Schwartz: Blog for Thu, Jan 29, 2004, http://use.perl.org/~merlyn/journal/17094

RFC 1631, "The IP Network Address Translator (NAT)", May 1994
http://www.ietf.org/rfc/rfc1631.txt?number=1631

RFC 1918, "Address Allocation for Private Internets", February 1996
http://www.ietf.org/rfc/rfc1918.txt?number=1918

The FreeBSD PF home page, http://pf4freebsd.love2party.net/

Peter Postma's PF on NetBSD pages, http://nedbsd.nl/~ppostma/pf/

Marcus Ranum: The Six Dumbest Ideas in Computer Security
(http://www.ranum.com/security/computer_security/editorials/dumb/index.html),
September 1, 2005

Kjell Jørgen Hole WiFi courseware,
http://www.kjhole.com/Standards/WiFi/WiFiDownloads.html, also see wifinetnews.com
(http://wifinetnews.com/archives/cat_security.html); also The Unofficial 802.11 Security
Web Page (http://www.drizzle.com/~aboba/IEEE/) comes higly recommended.

Greylisting.org greylisting.org (http://www.greylisting.org/)

Evan Harris: The Next Step in the Spam Control War: Greylisting
(http://greylisting.org/articles/whitepaper.shtml) (the original greylisting paper)

Mark Uemura: What's New in 4.3: authpf-noip
(http://undeadly.org/cgi?action=article&sid=20080324141004)

Henning Brauer: Carp and STP meet switch security
(http://bulabula.org/carp-and-stp-meet-switch-security.html)