firewalling with OpenBSD's pf and pfsync

David Gwynne <dlg@openbsd.org>

introduction

- who am i?
- what is openbsd?
- what are pf and pfsync?
- how do i use them?
- ask questions whenever you want

who am i?

- infrastructure architect in EAIT at UQ
  - i do stuff, including run the firewalls
- a core developer in openbsd
  - i generally play with storage
  - but i play with the network stack sometimes

what is openbsd?

- open source general purpose unix-like operating system
- descended from the original UNIX by way of berkeley and netbsd
- aims for “portability, standardization, correctness, proactive security and integrated cryptography.”
- supports various architectures/platforms

what is openbsd?

- one source tree for everything
  - kernel, userland, doco
  - bsd/isc/mit style licenses on all code (with some historical exceptions)
- 6 month dev cycle resulting in a release
- 3rd party software via a ports tree
- emergent focus on network services

what is openbsd?

- it is very aggressive
  - changes up and down the stack (compiler to kernel) to make a harsher, stricter, and less predictable runtime environment
  - minimal or no backward compatibility as things move forward
  - whole tree is checked for new bugs
  - randomise as much as possible all over

what is openbsd?

- it is extremely conservative
  - tree must compile and work at all times
  - big changes go in at the start of the cycle
  - we’re not afraid to back stuff out
  - peer review is necessary
  - we do back away from some tweaks for the sake of usability

what is pf?

- short for packet filter
- the successor to IP Filter (ipf)
  - ipf was removed due to license issues
- the exec summary is that it is a stateful filter for IP (v4 and v6) traffic
  - does a little bit more than that though...
- enabled by default

stateful filtering

- the firewall tracks connections through it
  - src+dst ip, proto, ports, etc
  - red-black tree used for lookups (O(log n))
  - pf states track tcp windows and such
  - each state takes memory, so there is a limit
  - packets without a state fall through to ruleset evaluation

pf rules

- basically a list of things to match on
  - eg, v4/v6, src+dst ip, protocol, ports, interface, direction, tcp flags, socket owner +group, icmp type, probability, and more...
- and what to do
  - pass/block/match, nat/rdr, divert, custom routing, tag, label, short circuit, and more...

pf rules

- last match wins (quick can short circuit)
- implicit keep state (but optional)
  - packets matching states get passed, so rules only have to allow the first packet
- ruleset loads are atomic and do not disturb existing states

pf in the stack

- sits between the traditional network stack (socket layer and forwarding) and interfaces
  - pf is run twice for forwarded packets, once coming into the stack and again going out
  - lots of hooks into other parts of the stack though, and links to itself and other bits

pf in practice

- pfctl(8) and pf.conf(5) for controlling pf

```
pfctl -d disable pf
pfctl -e enable pf
pfctl -si show info
pfctl -ss show states
pfctl -sr show rules
pfctl -nf /etc/pf.conf parse rules
pfctl -f /etc/pf.conf parse and load rules
systat pf watch -si type stats tick over
```

pf in practice: nat at home

- net is on pppoe0, internal is on em0
- sysctl net.inet.ip.forwarding=1

```
block
pass on em0
pass out quick on pppoe0 from (pppoe0)
pass out on pppoe0 from em0:network \
    nat-to (pppoe0)
```

pf in practice: anti-DoS

```
block
pass in on em0 from $mgmt_net to port ssh
pass in on em0 to port www \
  keep state (max-src-states 80 \
  tcp.closed 5) \
  synproxy state
```

pf in practice: remote site

- net: pppoe0, internal: em0, vpn: gif0

```
block
pass on em0
pass in on gif0
pass out on gif0 to $central_net \
  received-on em0
block out quick on pppoe0 to $central_net
pass out quick on pppoe0 from (pppoe0)
pass out on pppoe0 from em0:network \
  nat-to (pppoe0:0)
```

pf in practice: lots of nets

- net: trunk0, internal: vlan0-60, dmz: vlan100
- internal interfaces are in the “staff” ifgroup

```
block

antispoof for { vlan0 vlan1 ... vlan60 }
# block drop in on ! vlanX \
#   from vlanX:network to any...
```

pf in practice: lots of nets

```
pass in on trunk0
pass out on trunk0 received-on staff
```

```
pass out on vlan100 proto tcp \
  to $web port { 80 443 }
pass out on vlan100 proto tcp \
  to $files port { 139 445 } \
  received-on staff
```

pf in practice: ftp

```
# /usr/sbin/ftp-proxy
```

```
anchor "ftp-proxy/*"
pass in quick proto tcp to port ftp \
  rdr-to 127.0.0.1 port 8021
pass out quick user proxy
```

pf.conf

- there are a lot of other useful config bits
  - tables: radix trees instead of single ips
  - macros: foo=192.168.1.1; pass from $foo
  - lists: pass to $foo port { 80 443 }
  - ruleset optimiser and skip steps

failover

- one day your box will fail
  - so buy two!
- but your ruleset only allows connections to start, not continue
  - or you write really bad rulesets
- you need the states on the spare box for failover to work

what is pfsync?

- pfsync was invented to sync states between pf firewalls over the network
- does not concern itself with active/passive roles or directing failover, all peers are equal
- as states change in pf, pfsync is told and builds packets it transmits to peers
- pfsync merges updates from packets into the local state tree

what is pfsync?

- initial versions were rudimentary
- now does ipsec tdb sync for gateway failover
  - plans to sync other flows (ppp things?)
- big rewrite two years ago to allow active-active to work plus free code speedups

pfsync in action

- to use you just create the pfsync0 interface
  - it is an interface so there's something to manage, not as a transport for packets
  - and tell it which network interface to use to tx and rx packets
    - make sure pf allows pfsync packets too...
  - it is your job to keep the rulesets in sync

pfsync in action: carp(4)

- generally use carp(4) to prioritise firewalls
  - Common Address Redundancy Protocol
  - lets hosts share IPs on Ethernet interfaces
  - carp master gets the packets until it fails or the backup assumes higher priority
  - can use ifstated(8) to failover other interfaces based on carp on other nets

pfsync in action

```
# ifconfig pfsync0 create
# ifconfig pfsync0 syncdev bnx0
```

```
# ifconfig pfsync0 maxupd 128
# ifconfig pfsync0 defer

# ifconfig -g carp carpdemote 10
# ifconfig -g carp -carpdemote 10
```

pfsync in action

```
$ ifconfig pfsync0
pfsync0: flags=41<UP,RUNNING> mtu 1500
        priority: 0
        pfsync: syncdev: bnx0 maxupd: 128 defer: on
        groups: carp pfsync
$ ifconfig carp381
carp381: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        lladdr 00:00:5e:00:01:51
        description: staff servers
        priority: 0
        carp: MASTER carpdev vlan381 vhid 81 advbase 1 advskew 192
        groups: carp
        status: master
        inet6 fe80::200:5eff:fe00:151%carp381 prefixlen 64 scopeid 0x73
        inet 130.102.76.62 netmask 0xfffffc0 broadcast 130.102.76.63
$ ifconfig -g carp
carp: carp demote count 10
```

pfsync at home

- two firewalls at home could (should?) be considered overkill
- but pfsync gives you a serialised representation of a pf state which you can now put on disk...
- so you can patch kernels without losing irc

```
on shutdown:
# /sbin/pfctl -S /etc/pf.states
on boot:
# /sbin/pfctl -L /etc/pf.states
```

pfsync at work

- static ips and a single default route
- two firewalls with pfsync between them
- carp(4) on inside and outside
- graceful failover via ifconfig carpdemote
- when the master fails the backup firewalls carp interfaces come up and get the traffic

pfsync at my work

- 3 physical interfaces
  - 10G + 1G in failover trunk
  - 1G dedicated to pfsync traffic
- 60ish internal networks on separate vlans
  - carp interfaces on vlans on trunk
- 2 external links
  - vlans on trunk with ospf

Open Shortest Path First

- openbsd has its own routing daemons
  - ospfd, ospf6d, bgpd, ripd, ldpd...
- ospfd advertises routes on up interfaces
  - carp is up when master, down when backup
- carp changes move route advertisements
- ospf provides upstream failure detection
  - ospf can demote carp if upstreams are gone

ospfd.conf

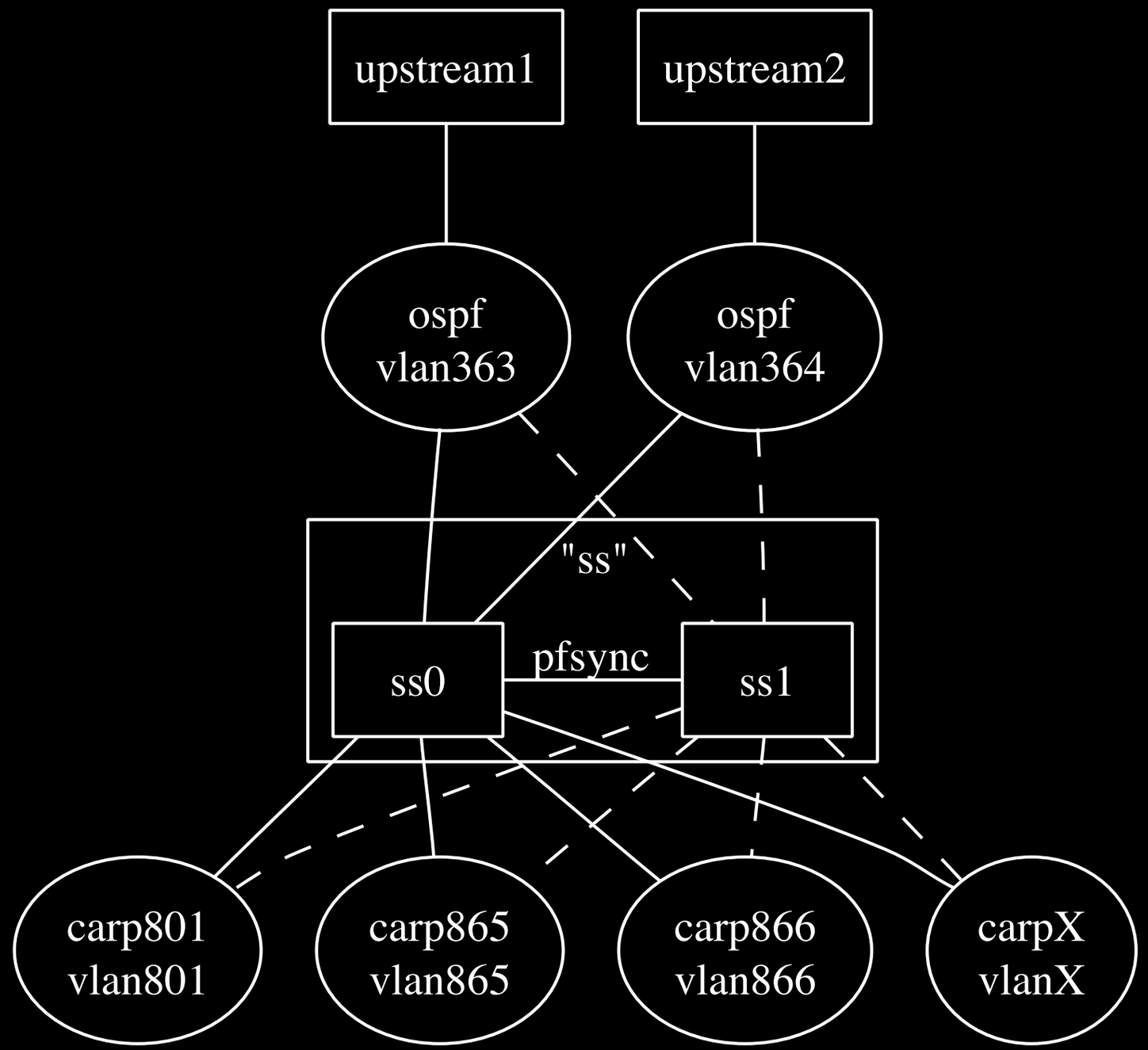
```
area 0.0.0.2 {
        demote carp 10

        interface vlan363 {
                auth-type crypt
                auth-md 1 Ust4ReJ59dnAVogG
                auth-md-keyid 1
        }
        interface vlan364 {
                auth-type crypt
                auth-md 1 r5Sy6ubyyHZaiMDB
                auth-md-keyid 1
        }
        interface carp70 { passive }
        interface carp72 { passive }
}
```

ospfd

- passive interfaces are members of the area, but don’t talk ospf
- ospf default dead time is 30sec with 10sec hello intervals, ie, ~35sec failovers
- we have a hack for ~1sec failovers

```
router-dead-time minimal
fast-hello-interval msec 250
```

pfsync at my work



pfsync caveats

- connections terminating on a firewall cannot be usefully synced because the socket and app state isn't transported
  - sucks for proxies (eg, ftp-proxy)
- high speed connections over two peers are limited because of the pfsync mitigation
- still some newer pf features that aren't represented in the pfsync messages

pf and pfsync and ...

- this is just how we (and others) use it
- there are a lot more tools and ways to mix them
  - bgp, relayd (load balancing/dsr), mpls, vrf, vpn

questions?

- ask away
- http://www.openbsd.org/