**Pf Firewall Tutorial**

**FreeBSD and OpenBSD ( pf.conf )**

The default firewall for OpenBSD as of v3.0 is called "packet filter" or more commonly referred to as pf. Pf is a BSD licensed stateful packet filter written by Daniel Hartmeier.

**History of PF**

PF was originally designed as replacement for Darren Reed's IPFilter, from which it derives much of its rule syntax. IPFilter was removed from OpenBSD's CVS tree due to OpenBSD developers' problems with its license. Specifically, Reed distributed some versions of his software with the license clause, "Derivative or modified works are not permitted without the author's prior consent." Due to this, the OpenBSD team decided to replace the software. This decision became the subject of wrangling among the parties involved, degenerating into a discussion that failed to reach mutual understanding. On the subject, OpenBSD project leader Theo de Raadt wrote, "Software which OpenBSD uses and redistributes must be free to all... for any purpose including... modification."

PF has since evolved quickly and now has several advantages over other available firewalls. Network Address Translation (NAT) and Quality of Service (QoS) have been integrated into PF, QoS by importing the ALTQ queuing software and linking it with PF's configuration. Features such as pfsync and CARP for failover and redundancy, authpf for session authentication, and ftp-proxy to ease firewalling the difficult FTP protocol, have also extended PF.

One of the many innovative feature is PF's logging. Logging is configurable per rule within the pf.conf and logs are provided from PF by a pseudo-network interface called pflog. Logs may be monitored using standard utilities such as tcpdump, which in OpenBSD has been extended especially for the purpose, or saved to disk in a modified tcpdump/pcap binary format using the pflogd daemon.[Wikipedia "History of pf"](http://en.wikipedia.org/wiki/PF_%28firewall%29)

Pf is an extremely powerful firewall. If you are interested in setting up a secure OS with an equally secure firewall then lets get started. First, we will go over the basics of getting the default [calomel.org](https://calomel.org/) pf.conf example file working. Then, we can talk about the specific options in the example file you may want to take a detailed look at. Options you may be interested in include the quality of service (QOS) called [HFSC](https://calomel.org/pf_hfsc.html) (Hierarchical Fair Service Curve Packet Scheduler) and stateful tracking options (STO). After you have pf setup and working you may also want to explore the possibility of setting up a [pf CARP firewall failover system](https://calomel.org/pf_carp.html) or the [relayd proxy server.](https://calomel.org/relayd.html)

For the purposes of this "how to" we will be working with the latest version of OpenBSD v5.1 stable (GENERIC kernel). The example file will use most of the advanced tools in the pf arsenal so you can get an understanding of how they would work in a fully operational pf.conf file. If you decided to not use some of the options, you can take them out. We just want to give you all the information we can in a functional config file so you can decide what you want to use.

**Pf for FreeBSD or OpenBSD ?**

Below you will find two scroll-able text boxes. The first contains the version of pf.conf for FreeBSD 9.1 which also works for OpenBSD 4.6 and earlier. The second scroll-able text box contains the new syntax for pf.conf started in OpenBSD 4.8. Both formats are available to make it easier for you to review the code. These are fully working config files with the exception of setting up a few variables for your environment.

**FreeBSD 9.1 and 10 CURRENT pf.conf (OpenBSD v4.6 and earlier)**

These rules are for FreeBSD as well as OpenBSD v4.6 and earlier revision of PF. Note that in OpenBSD v4.7 the syntax has been changed. Please look at the next text box below for the newer PF syntax.

This pf.conf is a collection of common connections you may want to support. Take a look through the configuration and add and delete what you need. This configuration will work fine on FreeBSD 9.1 and 10 CURRENT. FreeBSD 10 has SMP friendly pf of which we are a huge fan. With FreeBSD, a Myricom 10Gbit (10G-PCIE2-8B2-2S) network card and a pf.conf just like this example we are able to move 9.98Gbit/sec through a firewall. If you have the time please also take a look at our [FreeBSD Network Performance Tuning](https://calomel.org/freebsd_network_tuning.html) guide.

#

### Calomel.org pf.conf

#

################ FreeBSD pf.conf ##########################

# Required order: options, normalization, queueing, translation, filtering.

# Note: translation rules are first match while filter rules are last match.

################ Macros ###################################

### Interfaces ###

ExtIf ="mxge0"

IntIf ="mxge1"

### Hosts ###

Windows ="10.10.10.3"

Xbox360 ="10.10.10.4"

phone ="10.10.10.5"

WorkSsh ="123.123.123.123"

### Queues, States and Types ###

IcmpPing ="icmp-type 8 code 0"

SshQueue ="(ssh\_bulk, ssh\_login)"

SynState ="flags S/UAPRSF synproxy state"

TcpState ="flags S/UAPRSF modulate state"

UdpState ="keep state"

### Stateful Tracking Options (STO) ###

OpenSTO ="(max 90000, source-track rule, max-src-conn 1000, max-src-nodes 256)"

SmtpSTO ="(max 200, source-track rule, max-src-conn 10, max-src-nodes 256, max-src-conn-rate 200/30)"

SshSTO ="(max 100, source-track rule, max-src-conn 10, max-src-nodes 100, max-src-conn-rate 100/30, overload <BLOCKTEMP> flush global)"

WebSTO ="(max 4096, source-track rule, max-src-conn 64, max-src-nodes 512, max-src-conn-rate 500/100, overload <BLOCKTEMP> flush global)"

### Tables ###

table <BLOCKTEMP> counters

table <BLOCKPERM> counters file "/somedir/block\_permanent"

table <spamd-white>

################ Options ######################################################

### Misc Options

set skip on lo

set debug urgent

set block-policy drop

set loginterface $ExtIf

set state-policy if-bound

set fingerprints "/etc/pf.os"

set ruleset-optimization none

### Timeout Options

set optimization normal

set timeout { tcp.closing 60, tcp.established 7200}

################ Queueing ####################################################

# no quality of service (QOS) since it is not supported by the myricom 10gig

# mxge0 interface drivers and we would lose as much as 10% bandwidth anyways.

# for more information: https://calomel.org/pf\_hfsc.html

################ Normalization ###############################################

# set-tos 0x2E is Maximize-Reliability + Minimize-Delay + Maximize-Throughput

#scrub out log on $ExtIf all random-id min-ttl 15 set-tos 0x2E fragment reassemble

#scrub log on $ExtIf all reassemble tcp fragment reassemble

# NOTE: scrub is disabled because it scrambles rfc1323 time stamps.

no scrub

################ Translation #################################################

### NAT and Redirection rules are first match

# NAT with static NAT for the XBOX360

nat on $ExtIf from $Xbox360 to any -> ($ExtIf) static-port

nat on $ExtIf from $IntIf:network to any -> ($ExtIf)

# Apache or Nginx (external users to an internal server?)

rdr on $ExtIf inet proto tcp from !($ExtIf) to ($ExtIf) port https -> 10.10.10.100

rdr on $ExtIf inet proto tcp from !($ExtIf) to ($ExtIf) port http -> 10.10.10.100

# OpenSMTPD or Postfix with Spamd

rdr on $ExtIf inet proto tcp from !<spamd-white> to ($ExtIf) port smtp -> 10.10.10.200 port spamd

rdr on $ExtIf inet proto tcp from <spamd-white> to ($ExtIf) port smtp -> 10.10.10.250

# Openssh

rdr on $ExtIf inet proto tcp from $WorkSsh to ($ExtIf) port ssh -> lo0

rdr on $IntIf inet proto tcp from $Windows to $IntIf port ssh -> lo0

# Apache or Nginx (internal webserver for the LAN to localhost?)

rdr on $IntIf inet proto tcp from !($IntIf) to ($IntIf) port http -> lo0

rdr on $IntIf inet proto tcp from !($IntIf) to ($IntIf) port https -> lo0

# Bind or Unbound DNS for LAN machines

rdr on $IntIf inet proto udp from $IntIf:network to $IntIf port domain -> lo0

# Ntpd time server for the LAN

rdr on $IntIf inet proto udp from $IntIf:network to $IntIf port ntp -> lo0

# Anchors

rdr-anchor "games"

# Ftp ( secure ftp-proxy for the internal LAN )

nat-anchor "ftp-proxy/\*"

rdr-anchor "ftp-proxy/\*"

rdr pass on $IntIf proto tcp from $IntIf:network to any port 21 -> 127.0.0.1 port 8021

# DENY rogue redirection

no rdr

################ Filtering ###################################################

# Rules are best (closest) match. Notice we optimized the rules so external

# interface parsing is first followed by the internal interface.

### $ExtIf block abusive hosts in temp and perm tables

block drop in log quick on $ExtIf from <BLOCKPERM> to any

block drop in log quick on $ExtIf proto udp from <BLOCKTEMP> to any

block drop in log quick on $ExtIf proto tcp from <BLOCKTEMP> to any port != ssh

### $ExtIf default block with drop

block drop in log on $ExtIf

### $ExtIf inbound

pass in log on $ExtIf inet proto tcp from !($ExtIf) to 10.10.10.100 port https $TcpState $WebSTO

pass in log on $ExtIf inet proto tcp from !($ExtIf) to 10.10.10.100 port www $TcpState $WebSTO

pass in log on $ExtIf inet proto tcp from <spamd-white> to 10.10.10.250 port smtp $TcpState $SmtpSTO

pass in log on $ExtIf inet proto tcp from !<spamd-white> to 10.10.10.200 port spamd $TcpState $SmtpSTO

pass in log on $ExtIf inet proto tcp from $WorkSsh to lo0 port ssh $TcpState $SshSTO

### $ExtIf outbound

pass out log on $ExtIf inet proto tcp from ($ExtIf) to !($ExtIf) $TcpState $OpenSTO

pass out log on $ExtIf inet proto udp from ($ExtIf) to !($ExtIf) $UdpState $OpenSTO

pass out log on $ExtIf inet proto icmp from ($ExtIf) to !($ExtIf) $UdpState $OpenSTO

### $IntIf default block with return (TCP reset)

block return in log on $IntIf inet

### $IntIf inbound (restrict LAN clients to external machines here)

pass in log on $IntIf inet proto tcp from $IntIf:network to any port https $TcpState $OpenSTO

pass in log on $IntIf inet proto tcp from $IntIf:network to any port www $TcpState $OpenSTO

pass in log on $IntIf inet proto tcp from $Windows to lo0 port ssh $TcpState $OpenSTO

pass in log on $IntIf inet proto udp from $IntIf:network to lo0 port domain $UdpState $OpenSTO

pass in log on $IntIf inet proto udp from $IntIf:network to lo0 port ntp $UdpState $OpenSTO

pass in log on $IntIf inet proto icmp from $IntIf:network to $IntIf $IcmpPing $UdpState $OpenSTO

### $IntIf ftp secure secure proxy for LAN

anchor "ftp-proxy/\*" in on $IntIf inet proto tcp

### $IntIf outbound

pass out log on $IntIf inet proto tcp from $IntIf to $IntIf:network $TcpState

pass out log on $IntIf inet proto udp from $IntIf to $IntIf:network $UdpState

pass out log on $IntIf inet proto icmp from $IntIf to $IntIf:network $UdpState

### Games ( Xbox 360, Xbox ONE, PS3, PS4 and PC )

anchor "games"

############# END of FreeBSD pf.conf https://calomel.org #######################

**OpenBSD v5.1 pf.conf**

This pf.conf is for OpenBSD v4.7 or later including OpenBSD 5.1. These rules will \_not\_ work on any earlier versions of PF. The syntax has been changed to add the "match" and rdr-to / nat-to directive rule sets. These rules work similarly to the old rules above. You should be able to cut-paste this into your /etc/pf.conf and, with a little tweaking, get the firewall to do what you need to. We tried to put in all of the different services and redirections we could thing of to show you what you can do with Pf.

################ OpenBSD pf.conf https://calomel.org ##########################

# Required order: options, queueing, translation and filtering.

#

################ Macros #######################################################

### Interfaces ###

ExtIf ="em0"

IntIf ="em1"

### Hosts ###

Wraith ="10.10.10.3"

Xbox360 ="10.10.10.4"

WorkSsh ="192.168.100.100"

### Queues, States and Types ###

IcmpType ="icmp-type 8 code 0"

IcmpMTUd ="icmp-type 3 code 4"

SshQueue ="(ssh\_bulk, ssh\_login)"

#SynState ="flags S/SA synproxy state"

TcpState ="flags S/SA modulate state"

UdpState ="keep state"

### Ports ###

FtpPort ="8021"

SshPort ="8022"

### Stateful Tracking Options (STO) ###

FtpSTO ="(tcp.established 7200)"

ExtIfSTO ="(max 9000, source-track rule, max-src-conn 2000, max-src-nodes 14)"

IntIfSTO ="(max 150, source-track rule, max-src-conn 50, max-src-nodes 14, max-src-conn-rate 75/20)"

SmtpSTO ="(max 200, source-track rule, max-src-states 50, max-src-nodes 50, max-src-conn-rate 30/10, overload <BLOCKTEMP> flush global)"

SshSTO ="(max 5, source-track rule, max-src-states 5, max-src-nodes 5, max-src-conn-rate 5/60)"

WebSTO ="(max 500, source-track rule, max-src-states 50, max-src-nodes 75, max-src-conn-rate 120/100, overload <BLOCKTEMP> flush global)"

### Tables ###

table <BLOCKTEMP> counters

table <BLOCKPERM> counters file "/tools/pf\_block\_permanent"

table <spamd-white>

################ Options ######################################################

### Misc Options

set skip on lo

set debug urgent

set reassemble yes

set block-policy drop

set loginterface $ExtIf

set state-policy if-bound

set fingerprints "/etc/pf.os"

set ruleset-optimization none

### Timeout Options for normal operations

set optimization normal

set timeout { tcp.established 600, tcp.closing 60 }

### Timeout Options for anti SYN DDoS with short timeouts and increased states

# set optimization aggressive

# set timeout { adaptive.end 120000, interval 2, tcp.tsdiff 5, tcp.first 5, tcp.closing 5, tcp.closed 5, tcp.finwait 5, tcp.established 4200}

# set limit { states 100000, src-nodes 100000 }

################ Queueing ####################################################

### FIOS Upload = 20Mb/s (queue at 97%)

altq on $ExtIf bandwidth 19.40Mb hfsc queue { ack, dns, ssh, web, mail, bulk, bittor, spamd }

queue ack bandwidth 30% priority 8 qlimit 500 hfsc (realtime 20%)

queue dns bandwidth 5% priority 7 qlimit 500 hfsc (realtime 5%)

queue ssh bandwidth 20% priority 6 qlimit 500 hfsc (realtime 20%) {ssh\_login, ssh\_bulk}

queue ssh\_login bandwidth 50% priority 6 qlimit 500 hfsc

queue ssh\_bulk bandwidth 50% priority 5 qlimit 500 hfsc

queue bulk bandwidth 20% priority 5 qlimit 500 hfsc (realtime 20% default)

queue web bandwidth 5% priority 4 qlimit 500 hfsc (realtime (10%, 10000, 5%) )

queue mail bandwidth 5% priority 3 qlimit 500 hfsc (realtime 5%)

queue bittor bandwidth 1% priority 2 qlimit 500 hfsc (upperlimit 95%)

queue spamd bandwidth 1% priority 1 qlimit 500 hfsc (upperlimit 1Kb)

################ Translation and Filtering ###################################

### Blocking spoofed packets: enable "set state-policy if-bound" above

#block drop in log quick on ! $ExtIf inet from (em0:network) to any

### Block to/from illegal sources/destinations

block in quick on $ExtIf inet proto tcp from <BLOCKTEMP> to any port != ssh

block in quick on $ExtIf inet proto tcp from <BLOCKPERM> to any port != ssh

block in quick on $ExtIf inet proto udp from <BLOCKTEMP> to any port != ssh

block in quick on $ExtIf inet proto udp from <BLOCKPERM> to any port != ssh

#block in quick on $ExtIf inet from any to 255.255.255.255

#block in log quick on $ExtIf inet from urpf-failed to any

#block in log quick on $ExtIf inet from no-route to any

### BLOCK all in on external interface by default and log

block log on $ExtIf

### Network Address Translation (NAT with outgoing source port randomization)

match out log on $ExtIf from $Xbox360 to any received-on $IntIf tag EGRESS nat-to ($ExtIf:0) static-port

match out log on $ExtIf from !$Xbox360 to any received-on $IntIf tag EGRESS nat-to ($ExtIf:0)

### Packet normalization ( "scrubbing" )

### remove "min-ttl 64" if you need native traceroute functions or just use "traceroute -I" instead

match log on $ExtIf all scrub (random-id min-ttl 64 set-tos reliability reassemble tcp max-mss 1440)

### $ExtIf inbound

pass in log on $ExtIf inet proto tcp from !($ExtIf) to ($ExtIf) port https $TcpState $WebSTO queue (web, ack) rdr-to lo0

pass in log on $ExtIf inet proto tcp from !($ExtIf) to ($ExtIf) port www $TcpState $WebSTO queue (web, ack) rdr-to lo0

pass in log on $ExtIf inet proto tcp from <spamd-white> to ($ExtIf) port smtp $TcpState $SmtpSTO queue (mail, ack) rdr-to lo0

pass in log on $ExtIf inet proto tcp from !<spamd-white> to ($ExtIf) port smtp $TcpState $SmtpSTO queue (spamd) rdr-to lo0 port spamd

pass in log on $ExtIf inet proto tcp from $WorkSsh to ($ExtIf) port ssh $TcpState $SshSTO queue $SshQueue rdr-to lo0 port $SshPort

#pass in log on $ExtIf inet proto icmp from !($ExtIf) to ($ExtIf) $IcmpType $UdpState

#pass in log on $ExtIf inet proto icmp from !($ExtIf) to ($ExtIf) $IcmpMTUd $UdpState

### $ExtIf BitTorrent ( p2p )

anchor "bittor" in on $ExtIf

### $ExtIf outbound

pass out log on $ExtIf inet proto tcp from ($ExtIf) to !($ExtIf) $TcpState $ExtIfSTO queue (bulk, ack) tagged EGRESS

pass out log on $ExtIf inet proto tcp from ($ExtIf) to !($ExtIf) port ssh $TcpState $ExtIfSTO queue $SshQueue tagged EGRESS

pass out log on $ExtIf inet proto tcp from ($ExtIf) to !($ExtIf) port ftp $TcpState $FtpSTO queue (bulk) tagged EGRESS

pass out log on $ExtIf inet proto udp from ($ExtIf) to !($ExtIf) $UdpState $ExtIfSTO queue (bulk) tagged EGRESS

pass out log on $ExtIf inet proto udp from ($ExtIf) to !($ExtIf) port domain $UdpState $ExtIfSTO queue (dns) tagged EGRESS

pass out log on $ExtIf inet proto icmp from ($ExtIf) to !($ExtIf) $UdpState $ExtIfSTO queue (bulk) tagged EGRESS

### $IntIf return (TCP reset) and log internal traffic

block return log on $IntIf

### $IntIf inbound

pass in log on $IntIf inet proto tcp from $IntIf:network to !$IntIf port www $TcpState $ExtIfSTO

pass in log on $IntIf inet proto tcp from $IntIf:network to !$IntIf port https $TcpState $ExtIfSTO

pass in log on $IntIf inet proto tcp from $IntIf:network to !$IntIf port ftp $TcpState $IntIfSTO rdr-to lo0 port $FtpPort ##obsd 4.7 and earlier

#pass in log on $IntIf inet proto tcp from $IntIf:network to !$IntIf port ftp $TcpState $IntIfSTO divert-to 127.0.0.1 port $FtpPort ##obsd 5.1

pass in log on $IntIf inet proto tcp from $Wraith to $IntIf port ssh $TcpState $IntIfSTO rdr-to lo0 port $SshPort

pass in log on $IntIf inet proto udp from $IntIf:network to $IntIf port domain $UdpState $IntIfSTO rdr-to lo0

pass in log on $IntIf inet proto udp from $IntIf:network to $IntIf port ntp $UdpState $IntIfSTO rdr-to lo0

pass in log on $IntIf inet proto udp from $IntIf:network to $IntIf port bootps $UdpState $IntIfSTO

pass in log on $IntIf inet proto icmp from $IntIf:network to $IntIf $IcmpType $UdpState $IntIfSTO

### $IntIf ftp secure secure proxy for LAN

anchor "ftp-proxy/\*" in on $IntIf inet proto tcp

### $IntIf outbound

pass out log on $IntIf inet proto tcp from $IntIf to $IntIf:network port ssh $TcpState

pass out log on $IntIf inet proto icmp from $IntIf to $IntIf:network $IcmpType $UdpState

### Games ( Xbox 360, PS3 and PC )

anchor "games"

################ END of pf.conf https://calomel.org ###########################

**Setup of the example machine and daemons**

The fictional machine these rules run on is a firewall with an external interface on the Internet (em0 using DHCP) and an internal interface (em1 using network ips 10.10.10/24) on the private lan. The box is running the following services serving the internal lan (em1) only: samba windows share, bind dns, ntp time server, sshd and a ftp proxy. In addition, any machines allowed from the variable $WorkSsh will be allowed to ssh to the box from the Internet (em0).

As an added layer of security all services will be running on localhost and only those clients negotiating the redirect rules (rdr) will be able to connect. The ideology is if the firewall is off or disabled in some way then the services on the firewall are not available to anyone.

**NOTE:** If this box is going to be a firewall and you expect to pass packets from one interface to the other you \_MUST\_ enable packet forwarding. Even if pf is setup correctly for your network, no packets will traverse between your internal and external networks unless packet forwarding is turned on.

**Making sure ip forwarding is on**

You can see if ip.forwarding is set to on=1 or off=0 by typing "sysctl net.inet.ip.forwarding" . If ip.forwarding is off you can manually enable it by typing "sysctl net.inet.ip.forwarding=1". This command will only take effect for this session and ip.forwarding will be set back to its previous setting on reboot.

To make ip.forwarding permanent add the following line into the /etc/sysctl.conf file so packet forwarding will be enabled on boot.

### /etc/sysctl.conf

net.inet.ip.forwarding=1 # 1=Permit forwarding (routing) of packets

**Make sure ACPI v2.0 and APCI are on in the BIOS**

The Advanced Configuration and Power Interface (ACPI) defines common interfaces for hardware recognition, motherboard and device configuration and power management. According to its specification, "ACPI is the key element in Operating System-directed configuration and Power Management (OSPM)". The Advanced Programmable Interrupt Controller (APIC) is a more intricate Programmable Interrupt Controller (PIC) containing a magnitude more outputs, much more complex priority schema, and Advanced IRQ management.

Both ACPI v2.0 and APCI allow the firewall to work more efficiently. Go into your BIOS and make sure that both are enabled.

**What does it all mean?**

To use this config file you need to edit a few variables that pertain to your environment. We will \_not\_ be going over every rule in detail, but explaining what you need to do to get this example pf.conf working in your environment.

**Interfaces**These will be internal (em1) and external (em0) interfaces of your machine. Depending on the manufacture you will have different interface names. An easy way to look for the interfaces on your machine is executing an "ifconfig". The "Carp" interface name is a place holder.

**Hosts**The hosts section of the config shows the internal ip HomeSsh and the external ip WorkSsh we are allowing to ssh to the box.

**States & Queues**These are the connection options and state types each rule will use.

**flags S/SAFR :** Only tcp connections use the "flag" directive, udp and icmp connections can not. For stateful connections, the default in PF is flags set to S/SA. This means, out of SYN and ACK, exactly SYN may be set. SYN, SYN+PSH and SYN+RST do match, but SYN+ACK, ACK and ACK+RST do not. The safer flag settings are S/SAFR as we have in the example. This will deny packets who have the SYN+FIN and SYN+RST flags set since they are generally illegal combinations. The flag options, which we highly recommend understanding, are defined as the following:

**S = SYN** ... request to start a connection with the remote server. Having the SYN packet set is how a client and server will start the negotiation of a connection.

**A = ACK** ... acknowledge the payload or another datagram including a connection request. The ACK packet is a packet sent from the server back o the client. If the SYN+ACK bit is set on a new connection, a malicious client might be trying to pass something by a less advanced packet filter. You want to filter on this flag since a SYN+ACK is not part of a valid, initial connection request.

**F = FIN** ... end or finish a connection. Specifically the client or server is telling the other side that they have nothing more to say. If SYN+FIN is set this can be thought of as, "I want to talk, but have nothing to say." Those machines who set it can be safely ignored. You want to filter on this flag for new connections because it is invalid during a new connection request.

**R = RST** ... is a reset or refusal of a connection. SYN+RST is like answering a phone call by hanging up, which really does not make any sense. You want to filter on this flag since it will not be used in any real connection request. You might see this type of flag set on a Nmap scan for example.

**P = PSH** ... push this packet up the TCP stack ASAP and do not buffer. The PSH flag is used by telnet and SSH, for example, to cause the payload to be processed by the application right away. Normally, the PSH flag is not set on the initial connection, but after a connection is made and the client/server wants to make sure packets are handled quickly. One could filter SYN+PSH packets without a problem. Our example does not filter PSH as we have not found a valid reason to do so. If you wanted to filter PSH bits then add a "P" like so: S/SARFP.

**U = URG** ... simply tries to tell the receiving machine the other machine considers this data payload to be very important (urgent) and to process it ASAP. Urgent data should take precedence over any other data. For example, a Ctrl-C to terminate a FTP download. Just like the PSH packet, you can filter a SYN+PSH without issue is you want to. Our example does not filter URG as we have not found a valid reason to do so. If you wanted to filter URG bits then add a "U" like so: S/SARFU.

**E = ECE or ECN** ... is Explicit Congestion Notification Echo. ECN allows end-to-end notification of network congestion without dropping packets. It is an optional feature, and is only used when both endpoints signal that they want to use it. Traditionally, TCP/IP networks signal congestion by dropping packets. When ECN is successfully negotiated, an ECN-aware router may set a bit in the IP header instead of dropping a packet in order to signal the beginning of congestion. The receiver of the packet echoes the congestion indication to the sender, which must react as though a packet drop were detected. There is no real reason to filter the ECE bit.

**W = CWR** ... is Congestion Window Reduced. In addition to the two ECN bits in the IP header, TCP uses two flags in the TCP header to signal the sender to reduce the amount of information it sends. These are the ECN-echo and Congestion Window Reduced bits. Use of ECN on a TCP connection is optional; for ECN to be used, it must be negotiated at connection establishment by including suitable options in the SYN and SYN-ACK segments. When ECN has been negotiated on a TCP connection, the sender marks all data segments with the ECN-capable code point. A router that detects impending congestion may choose to mark an ECN-capable packet with the congestion experienced code point rather than dropping it outright. Upon receiving a TCP segment with the Congestion Experienced code point, the TCP receiver sends an acknowledgment with the ECN-echo flag set. The ECN-echo bit indicates congestion to the sender, which reduces its congestion window as for a packet drop. It then acknowledges the congestion indication by sending a segment with the Congestion Window Reduced code point. There is no real reason to filter the CWR bit.

**IMPORTANT NOTE:** While S/SAFR is practical and safe, it is also unnecessary to check the FIN and RST flags if traffic is also being scrubbed. The scrubbing process will cause PF to drop any incoming packets with illegal TCP flag combinations (such as SYN and RST) and to normalize potentially ambiguous combinations (such as SYN and FIN). Either way you can set S/SAFR or just S/SA as long as you also use scrubbing.

**keep state:**Specifies whether state information is kept on packets matching this rule. keep state works with TCP, UDP, and ICMP. In OpenBSD v5.1 and later, this option is the default for all filter rules.

**modulate state:**Much of the security derived from TCP is attributable to how well the initial sequence numbers (ISNs) are chosen. Some popular stack implementations choose very poor ISNs and thus are normally susceptible to ISN prediction exploits. By applying a modulate state rule to a TCP connection, pf(4) will create a high quality random sequence number for each connection endpoint. The modulate state directive implicitly keeps state on the rule and is only applicable to TCP connections. (man pf.conf)

**synproxy state:**By default, pf(4) passes packets that are part of a tcp(4) handshake between the endpoints. The synproxy state option can be used to cause pf(4) itself to complete the handshake with the active endpoint, perform a handshake with the passive endpoint, and then forward packets between the endpoints.

No packets are sent to the passive endpoint before the active endpoint has completed the handshake, hence so-called SYN floods with spoofed source addresses will not reach the passive endpoint, as the sender can't complete the handshake.

The proxy is transparent to both endpoints, they each see a single connection from/to the other endpoint. pf(4) chooses random initial sequence numbers for both handshakes. Once the handshakes are completed, the sequence number modulators (see previous section ISN) are used to translate further packets of the connection. synproxy state includes modulate state. (man pf.conf)

**WARNING:** We recommend you do NOT use synproxy unless you really know what you are doing and you really need it. The modulate state option is what you want to use 99% of the time for TCP connections. The reason is synproxy will NOT pass the SYN packet from the client to the server and this will break TCP Window Size Adjustment and Flow Control. OpenBSD 5.1 and later use dynamic window size adjustment and you can not override this behavior. A result of no window size negotiation is a significantly slower transfer speeds of hundreds of kilobytes per second instead of tens of megabytes per second. Also, synproxy will reset the MSS from the remote machine to the max-mss value you have set on the OpenBSD box. For example if the remote machine has a max-mss of 1360 and synproxy overrides this with 1440. The effect will be packets will get to your box unfragmented and the return packets, due to a MSS of 1440 being too large for thier network path, being fragmented or dropped on the way back to the remote system.

**Ports**The daemons running on our example machine are listening on certain ports. We lists those ports here. For example, openssh is listening on localhost on port 8022 so we set SshPort="8022". The variable names are short, but they are logically named. If you see the beginning of the name start with "IntIf" then this is the **Int**ernal **I**nter**f**ace. The ports listed in the variable AllowOUT are ports the firewall is going to allow clients on the internal lan to pass out to the Internet. For example, we are going to let internal machines connect to web servers on the Internet on ports 80 and 443 only.

Want to graph your Pf traffic statistics to see patterns on your network? Check out our guide to setting up [Pfstat to graph PF logs (pfstat.conf)](https://calomel.org/pfstat.html). With just a bit of time you can have full color graphs representing all of your traffic going though Pf.

**Stateful Tracking Options (STO)**STOs are used to limit ips or connection attempts to the machine's services. For example, if you had a web server open to the public and you knew the daemon could handle 8000 connections in total at a rate of 100 per 60 seconds. You could set the "max" to 8000 and "max-src-conn-rate" to 100/60. If you did not limit access from abusive hosts connecting to your server then your daemon may die or be unavailable to others clients. Basically a DDOS or denial of service. STOs give you the ability to set limits on remote clients on the firewall in front of your services. You can see the amount of packets blocked by a rate limiting rule by typing "pfctl -si" and looking at the "src-limit" entry.

**max** is the maximum amount of ESTABLISHED connections from all ips this rule will accept. If you know your web server can not accept more than 1000 connections then set the limit here. The max limit is the grand total, so 1000 connections from 1 ip would be the same as 1 connection from 1000 ips.

**source-track rule** means this rule will restrict access from each ip address individually. If one ip breaks the rules all the other ips will not be affected. This rule works well in punishing the abusers while the good clients are accepted.

**max-src-states** are the maximum amount of total states that will be created for an ip address. This rule does \_NOT\_ rely on the client completing the 3-way handshake. No matter what the state is this directive will cap them at the limit. For example, if we set the limit to 15 states then each ip can connect, attempt to connect, or connect and close a total of 15 created states. They are limited by the number of states we created for them. If they then disconnect all 15 connections they will have to wait until at least one state times out in order to connect again. This is very client restrictive, but can give you a lot of control denying abusive clients. You might use this to restrict connections to a ssh server. Setting the limit to two(2) will limit clients to two connections in total and they will not be able to connect again until the state expires for at least one of those connections.

**max-src-conn** is the maximum amount of ESTABLISHED (complete the 3-way handshake) states a single ip can have created without being denied. If the limit is set to 10 then each ip can have created 10 ESTABLISHED three-way connection states and no more. NOTE: That same ip which already has 10 ESTABLISHED states can also have an unlimited amount of other states like CLOSED and SYN\_SENT. This limit can be used to allow clients the ability to have a few states open at a time and once the first are closed they can reconnect again with having to wait until the state has expired. It is highly suggested that you use synproxy on all rules with max-src-conn.

**max-src-nodes** is the maximum number of individual ip addresses this rule will allow. Nodes are not the amount of states an ip can have, but the number of actual connecting ip addresses. This is a good way to limit a service that will only serve X ips at a time like a pop server or ssh server.

**max-src-conn-rate** is the limit on the rate of new connections completing the 3-way handshake over a set time interval. Lets say we set the rate to 30/100. This means we will only accept 30 connections per 100 seconds per ip that successfully complete the connection. It is highly suggested that you use synproxy on all rules with max-src-conn-rate.

**overload and flush global:**The overload directive means if a client ever reaches the amount of connections per time period specified in max-src-conn-rate the client will be added to the table specified and all of their current states will be globally flushed or deleted. You can then make a rule to do something to the ips in this table. For example, we are going to limit ssh connections to 20 per 60 seconds "max-src-conn-rate 20/60" and if someone reaches this limit they will be put into the OVERLOAD\_SSH table and all their current states will be cleared. We will then make a rule to block all connections from ips listed in OVERLOAD\_SSH to the ssh port.

You can reduce the power consumption of your firewall and keep track of system temperatures by using [Power Management with apmd](https://calomel.org/apm_control.html) and [Sensorsd hardware monitor (sensorsd.conf)](https://calomel.org/sensorsd_config.html).

**Tables**Tables are used for large lists of ips and can go into the tens of thousands of entries. We are using tables in the example to define two lists: blacklist and slowqueue. To use a file as the input for the table you simply need to have one ip per line. You can also use the "#" character for comments. Here is an example as well as the definition of what both tables are going to be used for in the example pf.conf .

# /etc/blacklist table example

134.123.12.1

182.43.23.24

# bots

69.34.124.23

**Blacklist:** This is a list of known abusers you never want to talk to again. Perhaps they are web site abusers, mail spammers or something else. It does not matter. The Blacklisted IPS list in conjunction with the block rule in the example pf.conf will deny access to/from those ips.

**Slowqueue:** This list of ip addresses is for people who you consider abusers, but you want to annoy them. Ips in this list will be subjected to 97% packet lose. The 3% of connections that do make it through will come through to your services like normal. In effect, they will not be blocked but have their packets randomly dropped. This is extremely effective against download accelerators, bots, and scanners. It can keep a remote (l)user tied up for weeks wondering why "it works sometimes."

**Misc Options**This is a list of default behaviors for pf. The log level is set to "Urgent" by default. The pf rule sets are going to be required to be in order of options, normalization, queueing, translation, filtering. The default policy is to drop packets. We are going to log on the external interface and the state policy is interface bound. The pf.os fingerprint file is defined and we are going to ask pf \_not\_ to try to optimize the rules as they are already in a good order. The man page of pf has a very good description of all of the miscellaneous options used here.

**Timeout Options**The timeout are how much time do we wait before we drop the state of an idle, open or closed connection. These timings are incredibly aggressive. You may wish to stay with the default timings or make them more aggressive. Your physical connection, type of services, machine specs and traffic load will determine your values.

**Normalization**Scrubbing is the act of combining and auditing a packet to conform to acceptable rules. We can never trust packets from the network and especially not from the Internet. This scrub rule will recombine and de-fragment packets on the external interface so that all packets transversing the firewall will be scrubbed. The minimum time to live (min-ttl) is set to a minimum of 64 to obfuscate packets from different machines from our internal network. NOTE: if you are having problems getting traceroute to work then remove the min-ttl directive from your scrub rules! A TTL (time to live) of 64 hops is the default value on OpenBSD and Ubuntu.

**Type of Service (set-tos)** We will also set the type of service (TOS, set-tos) to "ef" for new versions of OpenBSD or at least "reliability" to try to make the routers on the upload path reduce packet loss and travel time. Packet loss and retransmission are going to cause retransmission times of up to three(3) seconds which is a huge delay. You can also set the TOS to "lowdelay" to try to reduce the latency of the packets on the network for gaming. In gaming you do not care if a packet was lost as it is already to late to worry about it. For general use we highly recomend using a TOS setting of "ef" which signafies "Expedited Forwarding." Normally "ef" is good for VOIP communication, but we find this value works exceptionally well for all traffic. For FreeBSD we suggest using "reliability". Note, instead of using the word "reliability" you can also use "set-tos 0x04".

TOS XORmask Suggested Use

lowdelay 0x10 ftp, telnet, ssh

throughput 0x08 ftp-data, www, data servers

reliability 0x04 snmp, dns, or general

ef 0x2E VOIP, games

af31 0x68 VOIP

af32 0x70 VOIP

cs3 0x60 VOIP

A word about "ef" (0x2E): The IETF defines Expedited Forwarding behavior in RFC 3246. The EF PHB has the characteristics of low delay, low loss and low jitter. These characteristics are suitable for voice, video and other realtime services. EF traffic is often given strict priority queuing above all other traffic classes. Because an overload of EF traffic will cause queuing delays and affect the jitter and delay tolerances within the class, EF traffic is often strictly controlled through admission control, policing and other mechanisms. Typical networks will limit EF traffic to more then 30% of the capacity of the link.

Keep in mind the ACK packets originating on the the BSD machine will still have a TTL of 64 if that is what is defined by your "sysctl net.inet.ip.ttl" or if you do not set that directive at all. Here are some situations you might use each type of service for.

* Minimum delay: Used when the time it takes for a datagram to travel from the source host to destination host (latency) is most important. A network provider might, for example, use both optical fiber and satellite network connections. Data carried across satellite connections has farther to travel and their latency is generally therefore higher than for terrestrial-based network connections between the same endpoints. A network provider might choose to ensure that datagrams with this type of service set are not carried by satellite. This setting is also good for most people who play lots of games and want to make sure their packets arrive within the lest amount of time.
* Maximum throughput: Used when the volume of data transmitted in any period of time is important. There are many types of network applications for which latency is not particularly important but the network throughput is; for example, bulk-file transfers. A network provider might choose to route datagrams with this type of service set via high-latency, high-bandwidth routes, such as satellite connections. This would be recomended setting for a data farm or for a web cluster which needs to move arround a lot of bulk data, but does not need to really worry about when the data arrives.
* Maximum reliability: Used when it is important that you have some certainty that the data will arrive at the destination without retransmission being required. The IP protocol may be carried over any number of underlying transmission mediums. While SLIP and PPP are adequate datalink protocols, they are not as reliable as carrying IP over some other network, such as an X.25 network. A network provider might make an alternate network available, offering high reliability, to carry IP that would be used if this type of service is selected. For very important networks where the packet must make it there no matter the time frame and packet loss is unacceptable. Note that the standard retry time for TCP lost packets is three seconds which should be avoided at all costs.
* Minimum cost: Used when it is important to minimize the cost of data transmission. Leasing bandwidth on a satellite for a transpacific crossing is generally less costly than leasing space on a fiber-optical cable over the same distance, so network providers may choose to provide both and charge differently depending on which you use. In this scenario, your "minimum cost" type of service bit may cause your datagrams to be routed via the lower-cost satellite route.

The Maximum Message Segment Size (max-mss) for most networks can safely be set to 1440 bytes. The Minimum MSS = Maximum datagram size - IP header size - TCP header size. Every IPv4 host is required to be able to handle an MSS of at least 536 octets (= 576 - 20 - 20) and every IPv6 host is required to be able to handle an MSS of at least 1220 octets (= 1280 - 40 - 20). 1500 bytes is going to be the default MTU or Maximum Transmission Unit. If you subtract 20 bytes for the TCP packet headers, 20 bytes for TCP options and allow 20 bytes as a buffer from the MTU then you get the Max-MSS (1500-20-20-20=1440). For efficiency we want to send as much data as possible in that 1500 MTU packet size without ever going over thus we will waste those 20 buffer bytes as a safety zone against fragmentation. If we go over 1500 bytes then the packets will be fragmented by an upstream router and our speeds will severely diminish. The reason you want to manually confirm the MSS of your network is that there may be a misconfigured or obsolete router fragmenting your data. You need to know what the limitations of your network are and if you need to scrub your data to a MSS less than 1440.

**How do I find out what my Max-MSS on my network is using ICMP packets?** We can use the ping tool to test out the maximum payload allowed on our network out to the world. The maximum payload of a tcp packets is referred to as the MSS or maximum segment size. We need to ping a host that is outside of our ISP and somewhere on the Internet. Lets use the OpenBSD.org host since they have a machine and they allow us to ping them. Use ping with the (-D) do not fragment bit set and a (-s 1472) MSS payload size of 1472 bytes and (-c 1) send one ping request.

user@machine$ host openbsd.org

openbsd.org has address 199.185.137.3

user@machine$ ping -c 1 -D -s 1472 199.185.137.3

PING 199.185.137.3 (199.185.137.3): 1472 data bytes

1480 bytes from 199.185.137.3: icmp\_seq=0 ttl=64 time=132.951 ms

--- 199.185.137.3 ping statistics ---

1 packets transmitted, 1 packets received, 0.0% packet loss

round-trip min/avg/max/std-dev = 132.951/132.951/132.951/0.000 ms

As you can see our ping was successful with one(1) packet being sent and 0.0% packet loss. If your ping failed then reduce the payload size by 8 bytes until it does work (1472-8 = 1464 and so on). Our round trip time was 132.951 ms so we know that we can send at least 1472 bytes per packet. Lets try 1473 bytes payload...

user@machine$ ping -c 1 -D -s 1473 199.185.137.3

PING 199.185.137.3 (199.185.137.3): 1473 data bytes

ping: sendto: Message too long

ping: wrote 199.185.137.3 1501 chars, ret=-1

--- 199.185.137.3 ping statistics ---

1 packets transmitted, 0 packets received, 100.0% packet loss

Here we see that a payload for a ICMP packet of 1473 bytes is too large for the network (i.e. 100.0% packet loss) and since we told the routers not to fragment, the packet was dropped.

We now know a mss of 1472 bytes is the largest payload using ICMP we can send over our network according to our ping test. We also know that the MTU or maximum transmission unit according to our network card is 1500. You can use "ifconfig" and look for the external network card's "mtu" setting.

Now, it is important to remember that the ping test has an extra ICMP header that a normal TCP packet will not have. If we take the MSS or 1472 and add 8 bytes for the ICMP header we get the true TCP MSS of 1480. Remember that a IPv4 TCP packet's header is 20 bytes so 1480 plus 20 is 1500; our MTU has been verified.

We need to point out the name maximum segment size is in fact misleading. The value actually refers to the maximum amount of data a segment can hold and does not include the TCP headers. So if the MSS is 100, the actual maximum segment size \_could\_ be 120 (for a regular TCP header) or larger (if the segment includes TCP options).

Should we set our scrub max-mss to 1480? No, we recommend allowing a little bit of a buffer to make sure our packets are never fragmented on the network. We could have a full TCP packet, TCP headers and a bunch of TCP header options which would be fragmented. So, we are going to keep the MSS of 1440 from the ping test and waste the 20 bytes as a safety buffer.

Also, make sure to set the default MSS in /etc/sysctl.conf to the same maximum segment size of 1440. To set the default MSS on the command line use "sysctl net.inet.tcp.mssdflt=1440 . Take a look at our [Network Tuning and Performance Guide (OpenBSD)](https://calomel.org/network_performance.html) for details.

Your firewall is one of the most important machines on the network. Keep the system time up to date with [OpenNTPD "how to" (ntpd.conf)](https://calomel.org/ntpd.html), monitor your hardware with [S.M.A.R.T. - Monitoring hard drive health](https://calomel.org/smart_hd_status.html) and keep track of any changed files with a custom [Intrusion Detection (IDS) using mtree](https://calomel.org/ids_mtree.html). If you need to verify a harddrive for bad sectors check out [Badblocks hard drive validation/wipe](https://calomel.org/badblocks_wipe.html).

**Queueing**HSFC queueing is an excellent choice for quality of service (QOS) on any network. One can use Priority or CBQ, but we find HFSC is significantly more powerful. There are only a few lines in the queueing section, but it can get very complicated to explain. You can find a full discussion here at Calomel.org on the [Hierarchical Fair Service Curve (HFSC) of OpenBSD](https://calomel.org/pf_hfsc.html) page.

**Translation**

**Network Address Translation (NAT)** is the process of modifying network address information in datagram packet headers while in transit across a traffic routing device for the purpose of remapping a given address space into another. NAT is what you use when you want to allow machines on an internal LAN to access the Internet through the firewall.

Our pf example uses the line "nat on egress from $IntIf:network to any -> ($ExtIf:0) port 1024:65535" for this purpose. For security, we also want to make sure that all traffic originating from the box itself is NAT'd when going out the external (egress) interface. You can also use the line "nat on egress from (self) to any -> ($ExtIf:0) port 1024:65535" will take care of all traffic from any ip associated with the firewall itself going out the external (egress) interface.

The option, "port 1024:65535" will allow a larger pool of source ports for NAT. By default PF will use ports 49152 to 65535 which is a range of 16383. By increasing our source ports pool to a range of 64511 we make it significantly harder for the bad guys to spoof return packets coming back to the firewall. Case in point, the latest BIND DNS udp port randomization vulnerability in which the pool of source ports was small and attackers could flood forged UDP packets back to the DNS machine.

**Redirection** rules tell pf to point packets coming in one interface:port pair to another interface:port or machine:port. For example we have openssh listening on local host port 8022, but packets are coming in to our external interface on port 22. Redirection rules redirect the packets to where they are supposed to go. Not shown in the example rules, but shown in the "anchor" example below, is redirecting packets coming in on one interface to another machine on the network. For example, instead of allowing connections to ssh to the external interface you could instead redirect those packets to another machine inside the firewall.

**Filtering**This is where the logic of pf happens. Rules in the filtering area are responsible for accepting connections in and out of the box as well as allowing connections "through" the box from the internal lan to the Internet. Listed in the example are sections for the internal and external interfaces incoming and outgoing. Take some time and take a look at each of the rules in these four(4) blocks. Notice: the rules are in an optimized order of prevalence depending on what interface they control to what protocol they accept.

**Constructing a pf rule**

We are not going to go over every rule individually. Instead lets cover the basics of building a pf rule. With this knowledge you will be able to break down the example pf.conf as learn how to build your own.

This is one of the rules from the example. Lets go over what each directive does.

pass in log on $IntIf inet proto tcp from $IntIf:network to !$IntIf port $AllowOUT $TcpState $ExtIfSTO

**pass**tells pf whether to "pass" or "block" packets that match this rule.

**in**is the direction the packet must pass in to match this rule. You can check packets on the "in" or "out" direction of an interface.

**log**means you want any matching packet to trigger a log entry in pflog. It is important to have logs of everything that happens on the firewall for future reference. Also, if you decide to use the [graphical statistic program "pfstat"](https://calomel.org/pfstat.html) it will look at the logs to determine traffic patterns. Log everything you can.

**on $IntIf**is the the interface. This rule specifies the internal interface according to the variable $IntIf which corresponds to "em1" from the pf.conf.

**inet**simply means that this rule will only match ip version 4 (ipv4) packets. Ip version 6 (ipv6) "inet6" packets will \_not\_ match.

**proto tcp**is the protocol type, we are specifying tcp. You have the choice of tcp, udp and icmp depending on the rule you construct.

**from $IntIf:network**means the packet can originate from any machine on the entire internal network connected to the internal interface ($IntIf = em1). In the example the internal interface has an ip associated with it. All ips in this network are included in the directive "$IntIf:network". For example, if the internal interface has the ip 10.10.10.1 and the netmask is 255.255.255.0 then any machine with the ip 10.10.10.2 to 10.10.10.254 will match (10.10.10/24).

**to !$IntIf**is the destination of the packet. We are saying that the packet can go to any host as long as it is \_not\_ $IntIf. The "!" means not.

**port $AllowOUT**limits the ports a packet is destined for to the ones specified in the variable $AllowOUT. The example pf.conf defines $AllowOUT as the ports 80 and 443.

**$TcpState**are the state options. Our predefined variable $TcpState says "flags S/SA modulate state".

**$ExtIfSTO**is the variable for the stateful tracking options for this rule. $ExtIfSTO is expanded to "(max 9000, source-track rule, max-src-conn 2000, max-src-nodes 254)" when the rule is established.

**tagged**which is not shown in this rule. The "tagged" option is linked to the rdr (redirecting) rule. When a packet is redirected it can have a "tag" which is a string labeling that rule. The rdr rule then needs a pass rule to actually accept the packet. The pass rule can make sure it only accepts the right packet from the rdr line by looking for the "tagged" string. For example if the rdr rule has "tag OPENSSH" then the pass rule can look for "tagged OPENSSH".

**queue**which is not shown in this rule. A queue is only for packets going out of the box or for packets returning back to an external client. You only have control of the speed of packets as they leave your box. The queue will order packets by priority to make sure the packets of the highest priority will go out before lower priority ones.

**What about the rules that do not have all of those options?**

Some rules like the default block rules leave out some of the directives as pf can put them in be default. Lets take a look at the following rule.

block return log on $IntIf

Notice this rule does not have a "in" or "out" entry. This is because if it is not specified then pf automatically assumes that you mean "in and out". The same applies to the protocol that is not defined, pf sees this and assumes you mean "tcp, udp and icmp" packets. This rule expands to say that any packet the internal interface sees that is not allowed by any other rule will be blocked. The directive "return" says to send a "reset" packet back to the ip instead of just dropping the packet out right.

For more information or ideas about on CARP (Common Address Redundancy Protocol) check out our [OpenBSD Pf / CARP Firewall Failover](https://calomel.org/pf_carp.html) page. Calomel.org also has a [OpenSMTPD "how to" (smtpd.conf).](https://calomel.org/opensmtpd.html)

**Anchors (ftp and games)**

An anchor is a collection of filter and/or translation rules, tables, and other anchors that has been assigned a name. When PF comes across an anchor rule in the main ruleset, it will evaluate the rules contained within the anchor point. Processing will then continue in the main ruleset unless the packet matches a filter rule that uses the quick option or a translation rule within the anchor in which case the match will be considered final and will abort the evaluation of rules in both the anchor and the main rule sets.

Anchors are completely separate pf rules and have their own variable names, redirection and pass rules. Make sure your anchors have all the variables defined which are necessary to make the rules work. Rule evaluation of an anchor stays in the anchor. It will not go out to the main pf.conf for anything.

**FTP anchors** These are the nat, rdr and rules for the "forward" ftp-proxy. A "forward" direction proxy allows internal lan clients to connect to external ftp servers. The proxy will make rules and insert them into the ftp anchor on the fly. These are the default anchors necessary to make ftp-proxy work. For more information on ftp-proxy check out the [Calomel.org ftp-proxy "how to".](https://calomel.org/ftp_proxy.html)

**Games anchors** These anchors are for games you may play. The idea is you do not need the redirect rules or the ports open for games if you are not playing them at the time. Remove the rules if they are not needed. You may also want to use anchors if you have children. You can setup a simple cron job to disable the anchor at night enforcing a bedtime deadline. Also, the problem with editing the pf.conf directly are mistakes. If you have ever edited the pf file and saved it after accidentally editing or deleting a line you understand the hazards. We can easily add and delete rules using anchors.

Normally all of the rules are kept in the pf.conf and you edit the rules there. With anchors, we are going to have a separate text file with the rules we need for each separate game we are going to play. When we are ready to play the game the anchor is enabled and all of the rules in the text file are engaged. When we are done playing the game we flush (clear) the anchor rules. The anchor is still listed in the pf.conf, but it is empty and thus not used.

Lets use the game "Supreme Commander: Forged Alliance" as an example. When playing this multi-player game, not only must every client connect to the server machine, but every client must connect to each other. This is called "bus" topology and is used to reduce the upload bandwidth the server would normally need to support game data of this size. This also adds a lot of problems since every computer playing must be able to reach every other player. It only takes one incorrectly configured machine in the group to lose the game.

In the example pf.conf above we have two anchors, rdr-anchor "games" and anchor "games" which are both empty when pf initially loads. Note, anchors can be empty and not be considered an error. Now we need to setup the rules so our game will work.

Note, if you execute the script and add rules into an anchor they will be there till you clear them. If you change the anchor file, for example to change port numbers or something similar, you can execute the script again to replace the contents of the current anchor. This is the normally expected behavior.

BTW, once you get the anchors to work you are welcome to use the following script called **"anchors.sh"**. It is a simple shell script we use to enable anchors, turn them off or show the status.

#!/usr/local/bin/bash

#

## Calomel.org anchors.sh

#

# ./anchors.sh show = display active anchors

# ./anchors.sh off = deactivate all anchors

# ./anchors.sh rtor = turn ON rTorrent anchor

# ./anchors.sh supcom = turn ON Supreme Commander anchor

# ./anchors.sh xbox = turn ON Xbox360 anchor

if [ $1 = "show" ]

then

echo "anchor: games"

pfctl -a games -sr

echo " "

echo "anchor: rTorrent"

pfctl -a bittor -sr

fi

if [ $1 = "off" ]

then

echo "anchor: games"

pfctl -a games -F all

echo " "

echo "anchor: rTorrent"

pfctl -a bittor -F all

fi

if [ $1 = "rtor" ]

then

pfctl -a bittor -f /disk01/tools/pf\_anchor\_rtorrent

echo "anchor: rTorrent"

pfctl -a bittor -sr

fi

if [ $1 = "supcom" ]

then

pfctl -a games -f /disk01/tools/pf\_anchor\_supcom

echo "anchor: games"

pfctl -a games -sr

fi

if [ $1 = "xbox" ]

then

pfctl -a games -f /disk01/tools/pf\_anchor\_xbox360

echo "anchor: games"

pfctl -a games -sr

fi

**Anchor Example #1: Supreme Commander**

First, you need to set the anchor points in your pf.conf. These are the points in which the rules in the following section will be loaded. Think of the anchors as place holders for the rules. When you load the anchor, the rules will be evaluated in the place the anchor line is specified in pf.conf.

NOTE: This anchor uses the new syntax of Pf in OpenBSD v5.1 .

For the Supreme Commander rules we need to add two anchors, one redirect and one rules anchor. Add the following into your pf.conf:

## add to /etc/pf.conf

################ Filtering #################################

# Games (rules anchor)

anchor "games"

Second, the following text file contains the rules to redirect packets from any remote player listed in $SupComHosts and send them to the internal windows machine (Windowz). **Remember** that the anchor will \_not\_ read variable names from the main pf.conf. You will need to tell the anchor about interfaces, like ExtIf="em0", for the anchor to work.

The ports being forwarded from SupComHosts hosts are listed in $SupComPortsUDP. We also need the windows box to connect out to the GPGnet servers on port 80 (www) and to other players so we will open the ports listed in $GameUdpPorts for this purpose. Finally port 8767 will be used for the Windows box to communicate with a private TeamSpeak server. Here is the file called pf\_anchors\_supcom :

#

## Calomel.org Supreme Commander (pf\_anchors\_supcom)

#

### Interfaces ###

ExtIf="em0"

IntIf="em1"

### Hosts ###

GPGNetHosts="any"

SupComHosts="any"

Windowz="10.10.10.2"

### States & Queues ###

TcpState="flags S/SA modulate state"

UdpState="keep state"

### Stateful Tracking Options ###

ExtIfSTO ="(max 9000, source-track rule, max-src-conn 2000, max-src-nodes 10)"

### SupCom Ports ###

SupComPortsUDP="{6112, 9103}"

GameUdpPorts="{6112, 8767, 9103, 30340:30351}"

# $ExtIf inbound

pass in log on $ExtIf inet proto udp from $SupComHosts to ($ExtIf) port $SupComPortsUDP $UdpState $ExtIfSTO queue (bulk, ack) rdr-to $Windowz

# $IntIf outbound

pass out log on $IntIf inet proto udp from $SupComHosts to $Windowz port $SupComPortsUDP $UdpState

# $IntIf inbound

pass in log on $IntIf inet proto tcp from $Windowz to $GPGNetHosts port www $TcpState $ExtIfSTO

pass in log on $IntIf inet proto udp from $Windowz to !$IntIf port $GameUdpPorts $UdpState $ExtIfSTO

Finally, to enable to the rules you must first reload the pf.conf to enable the anchor place holders, rdr-anchor "games" and anchor "games". You only need to load the pf.conf this one time. Then when you are ready to use the Supreme Commander anchor you can use the following commands or use the script above "anchor.sh".

To re-read the pf.conf file use:

pfctl -f /etc/pf.conf

To enable the lines in the "games" anchor use:

pfctl -a games -f /directory/pf\_anchors\_supreme\_commander

To list out the active redirects and rules of the "games" anchor:

pfctl -a games -sr

To list out all anchor and main rules/redirects together:

pfctl -a '\*' -sr

To flush (clear) \_ALL\_ of the lines currently loaded in the "games" anchor:

pfctl -a games -F all

**Anchor Example #2: Xbox 360 for Xbox Live! Gold (NXE) and online games**

In order to get the Xbox 360 to work online correctly you need to pass three simple tests listed on the "Test Xbox LIVE Connection" page. The value, "Network: Connected" is if the physical network cable or wireless card is connected. The value "Internet: Connected" is if the Xbox has an ip address on the local LAN, but may or may not have access to the Internet. Finally, the "Xbox Live: Connected" is seen when the Xbox can connect to the LIVE service and authenticate your account.

If the page "Test Xbox LIVE Connection" says "Xbox Live:" is Unknown, Failed or that your MTU is less than 1364 then Pf is not setup correctly. Consequently the Live! service will disconnect randomly or you will not be able to play online games or hear people talk to you in the game. Since we are using PF we intend to open only the necessary ports for the xbox without compromising the integrity of our LAN.

BTW, when you get your rules working you are always welcome to look up our Xbox360 gamertag "Calomel org" and mail us. That is "Calomel", single space, "org".

First, lets make sure we understand what each of the "NAT" types the Xbox 360 can see. These can be separated into three(3) NAT classifications:

1. Strict is symmetric NAT.
2. Moderate is cone shaped NAT with port filtering or with UPnP turned off.
3. Open is cone shaped NAT with no port filtering or with UPnP turned on.

When a private address makes a connection outwards, NAT has to assign a state to the connection ,or flow, so return traffic can be sent to the correct private device. What makes the difference for Xbox Live is how these port numbers are chosen for UDP packets.

Symmetric NATs create a new entry (the name for the mapping of an external port to an private address and port) for every outgoing UDP packet if the destination and port are not the same. Cone NATs create a new entry only if the port number changes.

Sending three UDP packets to three different machines (all on the same port) creates three entries on a symmetric NAT and only one on a cone NAT. These two extra entries are what breaks Xbox Live. When you talk to the Live servers they remember the port you connected on and tell other Xboxes to talk to you over that same port. If your NAT is symmetric (default for pf) then it blocks the traffic from the other Xboxes as only the Live server is allowed back on that port. We need to use the "static-port" nat directive and allow traffic using our anchor.

At this point you may be wondering why don't we just use a UPnP program like [MiniUPnP](http://miniupnp.free.fr/). The reason is, we are not going to allow a device on the network, especially a proprietary game console, to make rules on our firewall. That is essentially what UPnP does. It allows a device to open and close ports through the firewall as it needs them and independent of our security mindset. Because of this we will make our own rules and by making our own rules we will better understand "how it works."

NOTE: This anchor uses the new syntax of Pf in OpenBSD v5.1.

First, add a rule in the Translation area of the pf.conf to enable static-port nat for the xbox machine. The line with "static-port" will allow the Xbox360 to connect to any ip. So, if we added the xbox "static-port" rule in our example pf.conf from above it would look like:

## add to /etc/pf.conf

### Network Address Translation (NAT with outgoing source port randomization)

match out log on egress from $Xbox360 to any nat-to ($ExtIf:0) static-port

match out log on egress from !$Xbox360 to any nat-to ($ExtIf:0) port 1024:65535

Second, you need to set the anchor points in your pf.conf. These are the points in which the rules in the following section will be loaded. Think of the anchors as place holders for the rules. When you load the anchor, the rules will be evaluated in the place the anchor line is specified.

For the Xbox 360 we need to add an anchor in pf. Add the following into your pf.conf:

## add to /etc/pf.conf

################ Filtering #################################

# Games (rules anchor)

anchor "games"

Lastly, the xbox360 needs to connect out to all ips on ports 88 udp, 3074 udp and 3074 tcp and allow connections in on port 3074 udp from all ips. These rules are put into a separate file which will be loaded using pfctl as an anchor. **Remember** that the anchor will \_not\_ read variable names from the main pf.conf. You will need to tell the anchor about interfaces, like ExtIf="em0", for the anchor to work.

The following anchor is called **"pf\_anchor\_xbox360"** and works perfectly. We are currently playing Brink online with this anchor. You are welcome to copy/paste these rules. Note we have setup a overload block rule to keep those abusive script kiddies off our console. We have witnessed attempts by angry players (Brink, Call of Duty Black Ops, Battlefield Bad Company 2 and Halo 3 ODST / Reach who seem to be especially sinister) to flood your console with corrupt packets to try to drop you out of an online game. This overload rule is a good solution to block corrupt and abusive (DDOS) packets of incoming UDP traffic in conjunction with antispoof or the urpf-failed block rule in the pf.conf above. An unsporting player should not be able ruin your online experience.

#

## Xbox360 anchor :: https://calomel.org

#

################ Macros ###################################

### Interfaces ###

ExtIf="em0"

IntIf="em1"

### Hosts ###

Xbox360="10.10.10.4"

### Ports ###

XLivePortsOutUdp="{88, 3074}"

XLivePortsOutTcp="{80, 443, 3074}"

XLivePortsInUdp="{3074}"

### States & Queues ###

TcpState="flags S/SA modulate state"

UdpState="keep state"

### Stateful Tracking Options ###

XboxSTOin ="(max 100, source-track rule, max-src-states 10, max-src-nodes 30, max-src-conn-rate 10/180, overload <XBLOCKTEMP> flush global)"

XboxSTOout ="(max 100, source-track rule, max-src-states 75, max-src-nodes 1)"

################ Translation and Filtering ###############################

# block the abusers

block quick on $ExtIf inet from <XBLOCKTEMP> to any

# $ExtIf inbound

pass in log on $ExtIf inet proto udp from !($ExtIf) to ($ExtIf) port $XLivePortsInUdp $UdpState $XboxSTOin tag XBOX360 queue (bulk, ack) rdr-to $Xbox360 label "games"

# $IntIf outbound

pass out log on $IntIf inet proto udp from any to $Xbox360 port $XLivePortsInUdp $UdpState received-on $ExtIf tagged XBOX360 label "games"

# $IntIf inbound

pass in log on $IntIf inet proto udp from $Xbox360 to any port $XLivePortsOutUdp $UdpState $XboxSTOout queue (bulk, ack) label "games"

pass in log on $IntIf inet proto tcp from $Xbox360 to any port $XLivePortsOutTcp $TcpState $XboxSTOout queue (bulk, ack) label "games"

################ END #######################################

Finally, to enable the rules you must first reload the pf.conf to enable the anchor place holders, rdr-anchor "games" and anchor "games". You only need to load the pf.conf this one time. Then when you are ready to use the Xbox360 anchor you can use the following commands or use the script above "anchor.sh".

To re-read the pf.conf file use:

pfctl -f /etc/pf.conf

To enable the lines in the "games" anchor use:

pfctl -a games -f /directory/pf\_anchor\_xbox360

To list out the active redirects and rules of the "games" anchor:

pfctl -a games -sr

To list out all anchor and main rules/redirects together:

pfctl -a '\*' -sr

To flush (clear) \_ALL\_ of the lines currently loaded in the "games" anchor:

pfctl -a games -F all

**Example: Two external ISP connections using route-to and round-robin**

It is possible you work for a company with two ATM/ADSL/DSL lines or you have a home network with a combination of DSL, cable modem and FIOS. If you have two ISP's to connect to then you should be able to load balance your traffic over each as well a control what traffic goes to what external interface. This example is an exercise to show the possible configuration options when using two external internet service providers. This is the theoretical environment we are looking to support:

* Route port 25 and 110 mail traffic from our LAN to the ISP on external interface 1 (ExtIf\_1)
* Load balance port 80 and 443 web traffic from our LAN between both ISP's. If one ISP goes down the other will take on 100% of the web traffic.
* Load balance port 80 external web traffic from the ISP on external interface 2 to our three(3) internal web servers (ExtIf\_2)

This is not a complete pf.conf, you can find that above. Here we list out the rules that are important to completing this example.

################ OpenBSD pf.conf ##########################

## Calomel.org Route-to two ISP's with load balancing

################ Macros ###################################

IntIf = "em1"

IntNet = "10.10.10.0/24"

ExtIf\_1 = "em0"

ExtGw\_1 = "111.111.111.111"

ExtIf\_2 = "em2"

ExtGw\_2 = "222.222.222.222"

Mail\_ports = "{53, 110}"

Web\_ports = "{80, 443}"

Web\_servs = "{ 10.10.10.11, 10.10.10.22, 10.10.10.33 }"

### States & Queues ###

TcpState="flags S/SA modulate state"

################ Normalization #############################

### remove "min-ttl 64" if you need traceroute functions

scrub log on {$ExtIf\_1, $ExtIf\_2} all random-id min-ttl 64 max-mss 1440 set-tos reliability reassemble tcp fragment reassemble

################ Translation ###############################

# External access to web server cluster

rdr on $ExtIf\_2 proto tcp from any to any port 80 tag EXTWEB -> $Web\_servs round-robin sticky-address

################ Filtering #################################

# BLOCK all in/out on all interfaces and log

block log on {$ExtIf\_1, $ExtIf\_2}

block return log on $IntIf

# $ExtIf\_2 inbound

pass in log on $ExtIf\_2 proto tcp from any to ($ExtIf\_2) port 80 $TcpState tagged EXTWEB

# $IntIf inbound

pass in log on $IntIf route-to { ($ExtIf\_1 $ExtGw\_1) } proto tcp from $IntNet to any port $Mail\_ports $TcpState

pass in log on $IntIf route-to { ($ExtIf\_1 $ExtGw\_1), ($ExtIf\_2 $ExtGw\_2) } round-robin from $IntNet to any port $Web\_ports $TcpState

################ END #######################################

**Optimizing PF rule set order to speed up evaluation**

To optimize pf and make it run faster you can order the rules by relevance according to the interface used, protocol and other values. Pf has logic to evaluate the rules listed and ignore rules in bunches that do not match what it is looking for.

For example, the following rule is \_not\_ optimized for pf. The rules for the external and internal interfaces are intermingled and thus pf must evaluate each line before going to the next. This means pf must take five(5) steps to look at all five(5) lines. What we need to do is put the rules into order of Interface (ExtIf), packet flow direction (in/out), address family (ipv4/ip6), protocol (tcp/udp), source address and port then destination address and port. This way pf can look at the rules more efficiently.

pass in on $ExtIf proto tcp from 1.1.1.1 to 2.2.2.2 port 80

pass out on $IntIf proto udp from 1.1.1.1 to 2.2.2.2 port 800

pass in on $ExtIf proto udp from 1.1.1.1 to 3.3.3.3 port 22

pass in on $IntIf proto tcp from 1.1.1.1 to 2.2.2.2 port 80

pass in on $ExtIf proto udp from 1.1.1.1 to 3.3.3.3 port 22

This new ordering below of the same rules is a lot faster for pf to traverse. The reason is the rules are now grouped by similar direction, interface and protocols values.

Pf works like this: Lets say you have a packet that needs to pass out on the internal interface and is udp based. Pf would look at the rules below and see that the first three(3) lines are for the external interface and ignore the whole set of the first three rules. Pf then stops on the fourth(4) line and sees the interface is for the internal interface, but the direction is in so this does not match. Finally, PF hist the last line with is for the internal interface, with the out direction and for the udp protocol. The last line matches.

pass in on $ExtIf proto tcp from 1.1.1.1 to 2.2.2.2 port 80

pass in on $ExtIf proto udp from 1.1.1.1 to 3.3.3.3 port 22

pass in on $ExtIf proto udp from 1.1.1.1 to 3.3.3.3 port 22

pass in on $IntIf proto tcp from 1.1.1.1 to 2.2.2.2 port 80

pass out on $IntIf proto udp from 1.1.1.1 to 2.2.2.2 port 800

When pf runs through the rules above it looks at the first three lines in one step, then the fourth line and then the last line. Three(3) steps in total to get through five(5) lines for our internal interface udp packet. By optimizing the rule order we have reduced the amount of rules pf needed to look at from five(5) to three(3). Just by re-ordering the pf lines we increased the speed of pf's evaluation engine by 160%.

Once you have your working pf.conf make sure to run "systat rules" and look at your rules. The pf resolver compares the rules against the values in the packet in a fixed order one packet at a time (serially). This is the order:

* 1. interface (on ExtIf or IntIf)
* 2. direction (in or out)
* 3. address family (inet or inet6)
* 4. protocol (proto tcp, proto udp or proto icmp)
* 5. source address (from 1.1.1.1)
* 6. source port (from 1.1.1.1 port 1024)
* 7. destination address (to 2.2.2.2)
* 8. destination port (to 2.2.2.2 port 80)

Make sure to group your rules so that pf can skip as many unnecessary rules as possible for the packet in question. Group all of the interface rules together, followed by the direction, the address family, then protocol, etc. Take a look at "systat rules" every time you modify your rules to make sure they run as efficiently as possible.

Also use "pfctl -gsr" and pay special attention to the "Skip steps" values like 'i' for interface, 'd' for direction, 'f' for address family, etc. When pf looks at the rules you want to allow the pf resolver to skip as many unnecessary rules as possible to speed up evaluation time. For example, if we ran "pfctl -gsr" against the five(5) optimized rules above we would see i=4 on the first line. This mean that if the interface the packet came on is from the internal interface (IntIf) Pf is allowed to skip to line 4 where the first IntIf rule is located. Once you have traffic flowing through the firewall use "pfctl -vsr" to to look at the rule evaluations of each line. If you have a rule that is evaluated but never used make sure to remove it.

Keep in mind that Pf is really a serial process which looks at state creation one packet at a time. Even if you only have a dozen rules it make sense to optimize them as best you can; if you have hundreds of rules this is a necessity. Keep in mind the use the "quick" directive to short circuit evaluation and speed up pf like we did in the example with "block" rules. Quick is first match logic while normal pf rules are last match. Also, take a look at any anchors you have and see if you can add conditions to them. If an anchor only deals with the external interface make sure to say so in you pf.conf. The conditional statements will allow the pf resolver to skip over the anchor completely if the packet is only used on the internal interface. Take a look at the pf example at the beginning of the page for ideas.

Finally, Take some time to read a very old post from '06 by the Pf developer called, [PF: Firewall Ruleset Optimization](http://www.undeadly.org/cgi?action=article&sid=20060927091645) . The article has a lot of really good insight into optimizing, using test rules to show the limitations of pf and tools to help you write rules more efficiently like our examples above.

Want more speed? Make sure to also check out the [Network Speed and Performance Guide](https://calomel.org/network_performance.html). With a little time and understanding you could easily double your firewall's throughput.

**Prioritizing empty TCP ACKs**

ALTQ is a framework to manage queueing disciplines on network interfaces. It manipulates output queues to enforce bandwidth limits and prioritize traffic based on classification.

This article presents a simple yet effective example of what the pf/ALTQ combination can be used for. It's meant to illustrate the new configuration syntax and queue assignment. The code used in this example is already available in the -current OpenBSD source branch.

Problem: We are using an asymmetric DSL with 512 kbps downstream and 128 kbps upstream capacity (minus PPPoE overhead). When I download, I get transfer rates of about 50 kB/s. But as soon as I start a concurrent upload, the download rate drops significantly, to about 7 kB/s.

Explanation: Even when a TCP connection is used to send data only in one direction (like when downloading a file through ftp), TCP acknowledgments (ACKs) must be sent in the opposite direction, or the peer will assume that its packets got lost and retransmit them. To keep the peer sending data at the maximum rate, it's important to promptly send the ACKs back.

When the uplink is saturated by other connections (like a concurrent upload), all outgoing packets get delayed equally by default. Hence, a concurrent upload saturating the uplink causes the outgoing ACKs for the download to get delayed, which causes the drop in the download throughput.

Solution: The outgoing ACKs related to the download are small, as they don't contain any data payload.

**Conclusions**

Pf can seem like a complicated firewall and it may take many months to become fluent. Take some time every few days to review one section of pf. Do not try to learn it all at once. When you are comfortable with one pf directive take a look at the next section and read over this page again if you have questions. The answers may become clearer as you see the same information a second time.

HELPFUL HINT: If you are interested in setting up a reverse, forward or redirector proxy then check out our [Relayd proxy "how to" ( relayd.conf )](https://calomel.org/relayd.html).

**Questions?**

**At what point in the TCP connection do each of the Pf timings take effect ?**

In the FreeBSD Pf config at the beginning of the page we mentioned changing the timings of pf to help resist SYN DoS attacks. The following table shows the Pf timings and at wait point in the TCP connection those timings affect the connection. The SYN attack would affect the first two steps, SYN and SYN/ACK. We also added the tcp.tsdiff which is the TCP timestamp difference to no more then 5 seconds.

Pf timings for each part of a TCP connection:

client -> SYN -> server tcp.first 5

server -> SYN/ACK -> client tcp.opening 5

client -> ACK -> server tcp.established 4200

client -> FIN -> server tcp.closing 60

server -> FIN -> Client tcp.finwait 30

RST -> (from either side) tcp.closed 30

Then, in Pf you would set the following:

### Timeout Options for anti SYN DDoS with short timeouts

set optimization aggressive

set timeout { tcp.first 5, tcp.opening 5, tcp.established 4200, tcp.closing 60, tcp.finwait 30, tcp.closed 30, tcp.tsdiff 5}

NOTE: Setting tcp.first and tcp.opening at 5 seconds is quite short and we have noticed those clients who use the Tor anonymity network (originally short for The Onion Router) might take longer then 5 seconds to open a connection to our server. Most Tor users are fine, but sometimes the Tor network is overloaded and can add a significant latency delay to the connection. Keep this mind when tweaking your settings.

**What is the process of Pf deciding on a matching state in the state table ?**

This is a response on the OpenBSD Mailing list from Henning Brauer. He is explaining the processes Pf uses to determine if the packet coming in matches a state we have, a state needs to be created for a pass/match rule or if Pf needs to drop the packet. We have reformatted text for readability.

we have a state table (let me nitpick: it's a tree). a packet comes in. we do a lookup in the table, looking for an entry where the key fields match the packet. keys are:

* protocol
* address family
* src addr
* dst addr
* src port
* dst port
* rdomain

if there is a match we found a state key, [but] not a state yet. so we start to walk the list of states that hangs off the state key to find the right one - there can be multiple with interface bound states.

now we have a state. that doesn't imply passing the packet yet, but at this point we decided for that state and against ruleset evaluation.

now some more checks - there is a bit of timeout handling and for tcp the sequence number checks, and the flags etc. if these all go OK we pass the packet (and apply actions if requested, like NAT, routing etc). if not, we block it.

**What hardware do I need for my PF firewall?**

The majority of the time any old P3 at 1GHz and 512MB or ram will do. It is important to remember that Pf is a single processor process. So, you will want a machine with a single, or at the most, a couple of fast CPU cores. What really matters is fast bus speed and memory access time (DDR1 is slower compared to DDR3 for example), cache size, and integer arithmetic performance if you have a lot of traffic.

Check out the section titled, "What kind of hardware would you recommend for a firewall ?" on our [Network Tuning and Performance Guide (OpenBSD)](https://calomel.org/network_performance.html) for more detailed information and examples.

**Can I setup Pf to listen to port knocking and allow ssh access ?**

Setting up Pf to listen to Port Knocking to allow ssh access should be easy enough. Lets do this, lets say if you hit four ports, 22222 then 12345, then 12345 again and then 34125, in that order will allow ssh authentication from that remote ip.

## Pf Port Knocking

### Ports

# These are the ports to knock in order

portknock1=22222

portknock2=12345

portknock3=34125

### Tables ###

# These are the tables which will hold the ips who are in the processes of port

# knocking or who have successfully port knocked and are allowed to ssh to port

# 22.

table <portknock1> persist

table <portknock2> persist

table <portknock3> persist

### Translation and Filtering ###

# Use Pf with synproxy to keep state on the ports. Add the ip to the overload

# table if the remote ips hits the right port. Once the remote ip hits all the

# ports in order then grant the ability to connect through sshd

pass in on $ExtIf inet proto tcp from any to $ExtIf port $portknock1 synproxy state (max-src-conn 1, overload <portknock1>)

pass in on $ExtIf inet proto tcp from <portknock1> to $ExtIf port $portknock2 synproxy state (max-src-conn 2, overload <portknock2>)

pass in on $ExtIf inet proto tcp from <portknock2> to $ExtIf port $portknock3 synproxy state (max-src-conn 1, overload <portknock3>)

# This is the rule to allow successfully port knock'd ips to the sshd daemon

pass in on $ExtIf inet proto tcp from <portknock3> to $ExtIf port ssh

That should be it. Since the ips in the tables portknock1 and portknock2 are temporary we will clean them out every 5 minutes. Then the ips in portknock3 are cleared out every hour.

\*/5 \* \* \* \* root /sbin/pfctl -q -t portknock1 -T expire 60

\*/5 \* \* \* \* root /sbin/pfctl -q -t portknock2 -T expire 60

00 \* \* \* \* root /sbin/pfctl -q -t portknock3 -T expire 600

Now, to open up ssh access to your remote ip you will execute "telnet server\_ip 22222", then "telnet server\_ip 12345", "telnet server\_ip 12345" again and finally "telnet server\_ip 34125". The server\_ip is the ip address of your firewall running Pf. If you telnet to all the ports in order Pf will add your remote ip to the portknock3 table. Then just "ssh username@server\_ip" and Pf will allow ssh authentication to your machine. You may to think about adding more ports or even hitting the same port many times to defeat port scanning.

**How can I redirect packets to any ip on port 8080 to that same ip, but now on port 80 ?**Lets say you have an internal machine sending packets to port 8080 of some random internet ip. You can redirect those packets to the same destination ip, but on a different port like port 80. To do this you need to use the bitmask directive on a rdr rule like so:

OpenBSD v4.6 and earlier

rdr on $IntIf proto tcp to any port 8080 -> 0.0.0.0/0 port 80 bitmask

OpenBSD v4.7 and later

pass in on $IntIf proto tcp to any port 8080 rdr-to 0.0.0.0/0 port 80 bitmask

**Can I automatically block port scanners like Nmap, Superscan and Unicornscan on my firewall ?**Sure. A scanner is a remote computer who accesses multiple ports on your machine to see which ones are open. Once they know what ports are open they can deduce what services are being run. Our anti-scanner using Pf will look for access to ports not used for any local services. The theory being that no one should be connecting to our firewall on any port we are not running a service on.

Our example will be looking for connections to ports 23 though 79 and 6000 through 8000 from any remote ip to any local ip including carp addresses. If a remote computer connects to any of these ports they are added to the table BLACKLIST, all of there states are flushed (cleared) and any future connections from that ip will be blocked to our firewall on all ports.

Notice we are using "synproxy state" to accept connections. This will make a connection with the remote machine doing the full tcp handshake and then immediately drop the connection. This is because there are no services on our machine using the ports listed in AntiScanPort. The anti-scanner setup will \_not\_ work for udp or icmp traffic.

If you would like to clear out ips in the BLACKLIST table after a specified amount of time scroll down a bit and look for the question, "How can I expire the ips in the blacklist table?"

These are the lines you can add to your pf.conf in the same format as the example pf.conf above.

################ Macros ###################################

### States & Queues ###

TcpState="flags S/SA synproxy state"

### Ports ###

AntiScanPort="{23:79, 6000:8000}"

### Stateful Tracking Options ###

AntiScanSTO ="(max 60, source-track rule, max-src-conn 1, max-src-nodes 60, max-src-conn-rate 1/60, overload <BLACKLIST> flush global)"

################ Tables ####################################

table <BLACKLIST> persist

################ Filtering #################################

# Block blacklisted

block in quick on $ExtIf from <BLACKLIST> to any

# ExtIf Inbound

pass in log on $ExtIf inet proto tcp from any to any port $AntiScanPort $TcpState $AntiScanSTO

**Can I block ad servers with pf?**Yes, you can. By blocking advertiser's servers you will not have to download the in page HTML ads while browsing. This saves time, bandwidth and will probably make your users quite happy. To start, you need to make a pf table to hold the ad server ips, add a pf rule to block connections to ips in the table and run a script to download the free list of ips.

First, make a persistent table in your pf.conf and then add a block rule. Here we make a block rule on the internal interface which will block the connection and send a tcp reset back to the LAN client. We do this so the client does not pause while accessing a site with ads.

################ Tables ####################################

# Make a table to keep the ad server ips in

table <ad-servers> persist

################ Filtering #################################

# Block to ad servers from our LAN machines on the Internal Interface

block return in quick on $IntIf from any to <ad-servers>

Lastly, run the following script to pull the current list of ad server ips from pgl.yoyo.org and insert them into the ad-servers table. You may also want to make a cron job to run this script every few days to make sure you have the latest list of ips.

#

## Calomel.org pf\_anti\_adserver.sh

#

## get new ad server list

/usr/local/bin/wget -O /tmp/tmp\_ad\_file http://pgl.yoyo.org/adservers/iplist.php?ipformat=plain

## clean html headers out of list

cat /tmp/tmp\_ad\_file | egrep '([[:digit:]]{1,3}\.){3}[[:digit:]]{1,3}' | sort -n > /tmp/tmp\_output

## enter into pf anti-ad server table

pfctl -t ad-servers -T replace -f /tmp/tmp\_output

## clean up and remove temp file

rm -rf /tmp/tmp\_\*

**How can I make sure my network interfaces stay online ?** In order to better keep track of which machine is the master or slave using CARP or just make sure you PPPOE or DHCP devices are online you may want to take a look at our page, [Ifstated interface monitor "how to"](https://calomel.org/ifstated.html). With Ifstated you have the ability to have emails sent to you when the firewalls change state or any other operation you want to add.

**How can I increase the amount of packets per second (pps) my firewall can handle?** If you are pushing a lot of data though the firewall you may be seeing a limit of 10K-20K pps. You can increase the packet switching rate to easily 30K pps or higher by using the following settings in your /etc/sysctl.conf and using the proper kernel and hardware. First, use the GENERIC kernel without SMP. The multiprocessor kernel will actually slow packet forwarding down. Use at least one(1) gigabyte of RAM and solid server network cards like the Intel Pro 1000/MT (em0). The hardware for the firewall should be at least an Intel Xeon 2.5GHz or Amd64 2.4GHz or faster. Check out our [Network Speed and Performance Guide (OpenBSD)](https://calomel.org/network_performance.html).

**How can I find performance bottlenecks and display real time statistics about the firewall hardware?** Run the command **"systat vmstat"** to give you a top like display of memory totals, paging amount, swap numbers, interrupts per second and much more. Systat is incredibly useful to determine where the performance bottleneck is on a machine.

Why is the machine's throughput limited to less than 150Mbit/sec and around 15k pps? Make sure you are using the latest version of OpenBSD. Many improvements have been made which decrease throughput bottlenecks. Second, make sure you are using quality network cards like the Intel Pro/1000 MT or GT series (em4). They are designed to cache data and process the traffic more efficiently. Third, check the options we explain in our [Network Speed and Performance Guide (OpenBSD)](https://calomel.org/network_performance.html). net.inet.ip.ifq.maxlen defines how many packets can be queued in the IP input queue before further packets are dropped. Packets from the network card are first put into this queue and the actual IP packet processing is done later. Gigabit cards with interrupt mitigation may spit out many thousands of packets per interrupt; heavy use of pf can also slowdown packet forwarding. A safe net.inet.ip.ifq.maxlen is 256 times the number of physical interfaces in the machine, but no more than 2500. Finally make sure the sendspace and recvspace are set to 262144 (~32K) The following settings worked very well for a Xeon based firewall with CARP pushing 920Mbits/s at around 37K pps. NOTE: If you want more information about tcp tuning check out the study over at PSU.edu called, ["Enabling High Performance Data Transfers"](http://www.psc.edu/networking/projects/tcptune/).

**How do I list out the pf rules in order with rule numbers?**

pfctl -vvs rules | grep @

**How do I list out all the pf rules and other options in my rules?**

pfctl -sa

**How do I watch the pf logs in real time?**

tcpdump -n -e -ttt -i pflog0

**How do I cat the pf log file?**

tcpdump -n -e -ttt -r pflog0

**How do I tell pf to re-read the pf.conf file after I make a change?**

pfctl -f /etc/pf.conf

**How can I see what ip addresses are in the abusive hosts tables?** You need to use pfctl with "-t" with the name of the table and then "-T show" to list out the contents of a table (i.e. pfctl -t SLOWQUEUE -T show). Here is a shell script called **show\_abusive\_hosts.sh** that will list out all three(3) queues from the example pf.conf. As an added bonus, this shell script will print out the ip address and the hostname of each entry in each of the tables.

#!/usr/local/bin/bash

#

## Calomel.org show\_abusive\_hosts.sh

## Purpose: Display ips and hostnames in the abusive hosts tables

#

total\_slowqueue=`pfctl -t SLOWQUEUE -T show | wc -l`

total\_blacklist=`pfctl -t BLACKLIST -T show | wc -l`

total\_overload=`pfctl -t OVERLOAD\_SSH -T show | wc -l`

#

echo -n "SLOWQUEUE"; echo -n " ("; echo -n $total\_slowqueue; echo ")"

for i in $( pfctl -t SLOWQUEUE -T show );

do

echo -n " "; echo -n $i; echo -n -e "\t" ; echo -n " "; host $i | awk '{print $5}'

done

#

echo " "

echo -n "BLACKLIST"; echo -n " ("; echo -n $total\_blacklist; echo ")"

for i in $( pfctl -t BLACKLIST -T show );

do

echo -n " "; echo -n $i; echo -n -e "\t" ; echo -n " "; host $i | awk '{print $5}'

done

#

echo " "

echo -n "OVERLOAD\_SSH"; echo -n " ("; echo -n $total\_overload; echo ")"

for i in $( pfctl -t OVERLOAD\_SSH -T show );

do

echo -n " "; echo -n $i; echo -n -e "\t" ; echo -n " "; host $i | awk '{print $5}'

done

**How can I expire the ips in the blacklist table?** Pfctl allows you to expire entries after a specified time period. Pfctl can be used to delete addresses which had their statistics cleared more than number seconds ago. For entries which have never had their statistics cleared, number refers to the time they were added to the table. A cron job executing the following every hour will clear out any entries older than 3600 seconds (1 hour) from the table called "blacklist".

#minute (0-59)

#| hour (0-23)

#| | day of the month (1-31)

#| | | month of the year (1-12 or Jan-Dec)

#| | | | day of the week (0-6 with 0=Sun or Sun-Sat)

#| | | | | commands

#| | | | | |

#### Clear entries older than 3600 secs from the table "blacklist"

00 \* \* \* \* pfctl -t BLACKLIST -T expire 3600

**Do you have a script reporting connections to and from the machine using pf logs?**Here is a Perl based reporting tool called ["Pantz PFlog Stats"](http://www.pantz.org/software/pf/pantzpfblockstats.html) which I highly recommend.

**What other OSs has pf been ported to?**Apart from its default platform OpenBSD, PF is also installed by default in FreeBSD starting with version 5.3, in NetBSD from version 3.0, and appeared in DragonFly BSD from version 1.2.

**Why is the Pf firewall able to negotiate DHCP requests even though there are no rules allowing this operation?**DHCP uses BPF (similar to the way tcpdump does) and is below PF and thus not restricted by PF.

**What does the log line "host /bsd: pf: complete: 0xfffffe80d026ad00(1552)" mean ?**If you have enabled the "debug" or lower logging level in Pf then this line will show up in the logs. All this means is a that a packet was successfully reassembled.

**I get an error that "some option" is not supported!**If you are using an older version of OpenBSD than v5.1 or one of the other OSs pf was ported to, make sure the pf directive in question is supported. As pf is developed, new directives are added and thus old distributions may not have all of the functions listed in the example config.

**Can pf use multiple CPUs or cores?**No, it can not. If you find you are having problems with high cpu interrupt usage try the single cpu kernel. The multi-cpu kernel uses more cpu time for little overall gain.

**Is there any more I can do with OpenBSD?**Check on the main page of [calomel.org](https://calomel.org/) for more "how to"s and discussions.