FreeBSD Firewalls
SS-E 2014

Kevin Chege

ISOC

What's a Firewall?

- Computer network security device to protect devices, or restrict access to or from a network
- Analyzes traffic coming in or going out (or through it) and determines a course of action based on a pre-defined rule set
- Firewalls can be found anywhere:
  - On your laptop OS
  - On routers
  - On server OS
  - On network hardware appliances

Types of firewalls

- Packet Filters – analyze network packets and decide a course of action based on configuration
- Stateful Filters – track network “conversations” and maintain a table of which connections are in an active conversations
- Application layer – aka Layer 7 firewalls are able to detect if an unwanted protocol is attempting to bypass the firewall on an allowed port

Keeping State vs Stateless

- Stateful inspection refers to ability to track the state, or progress, of a network connection
- By storing information about each connection in a state table, a firewall is able to quickly determine if a packet passing through the firewall belongs to an already established connection.
- If it does, it is passed through the firewall without going through ruleset evaluation saving time and avoiding extra processing.

Typical features of a Firewall

- Rule Syntax
- NAT control
- Able to pass, redirect or drop traffic based on the rules
- Logging feature – to allow audit of activities and of traffic
- Stateful inspection - not all and may need to be enabled with extra config options
- Ability to be either inclusive or exclusive - An exclusive firewall allows all traffic through except for the traffic matching the ruleset (default is to allow). Inclusive firewall does the reverse (default is to block)

FreeBSD Firewalls

- FreeBSD ships with 3 Main firewalls:
  - IPFW – IP FireWall is (by default) a stateless firewall. FreeBSD sponsored firewall software application authored and maintained by FreeBSD volunteer staff members.
  - IPF – IP Filter can be configured as stateful or stateless. Open source application and has been ported to FreeBSD, NetBSD, OpenBSD, SunOS™, HP/UX, and Solaris™ operating systems. IPFILTER is actively being supported and maintained, with updated versions being released regularly.
  - PF – Packet Filter can be configured as stateful or stateless. Maintained by OpenBSD Project

PF (Packet Filter)

- Was initially developed for OpenBSD
- Has been successfully ported to many other operating systems including all the other BSDs and Mac OS X
- Written by Daniel Hartmeier
- Derived its rule syntax from IPFilter
- Has many features

Features

- Can do both stateless or state-full firewalling
- Can do Network Address Translation
  - Additionally can do Bidirectional NAT aka One to One NAT
- Combined with ALTQ (ALTernate Queueing framework for BSD) can perform QoS
  - Priority queuing – assign certain traffic a higher priority than others before forwarding
  - Class Based Queuing – assigning bandwidth to certain queues and reducing bandwidth for others
- Can be configured for automatic fail-over between 2 boxes using CARP – Common Address Redundancy Protocol

Features cont'd

- FTP-proxy integration to handle FTP firewalling
- Configurable logging per rule to pflogd
  - Logs can be further monitored with tcpdump
- Simple IP Filter rule syntax
  - Eg: *pass in quick on em0 inet proto tcp all*
- Macro definition – to simplify rule creation
  - Eg identify an interface as "LAN" instead of "em0"
- Support for transparent proxying with SQUID
  - Redirect all traffic destined for a port 80 to the Squid port 8080 for Squid to process
- Among many others

Working with PF

- Installed by default on FreeBSD since FreeBSD 5.3 but is disabled
- Can start in from boot by adding the following to /etc/rc.conf: ***pf_enable=YES***
  - Or by ***kldload pf.ko***
- Start it by doing
  - ***/etc/rc.d/pf start*** OR ***pfctl –e***
- You may want to compile pf support into the kernel to enable:
  - Pfsync seudo device
  - CARP for automatic failover
  - ALTQ – for prioritization, bandwidth throttling

Options in rc.conf

- **pf_enable="YES**" # Enable PF (load module if required)
- **pf_rules="/etc/pf.conf"** # rules definition file for pf
- **pf_flags=""** # additional flags for pfctl startup
- **pflog_enable="YES"** # start pflogd(8)
- **pflog_logfile="/var/log/pflog"** # where pflogd should store the logfile
- **pflog_flags=""** # additional flags for pflogd startup
- You will also want to enable packet forwarding between interfaces and this can be done by
  - **gateway_enable="YES"** in /etc/rc.conf

Working with PF

- **pfctl –e** *Enable PF*
- **pfctl -d** *Disable PF*
- **pfctl -F all -f /etc/pf.conf** *Flush all rules (nat, filter, state, table, etc.) and reload*
- **pfctl -s [rules | nat | state** *]* *Report on the filter rules, nat rules, or state table*
- **pfctl -vnf /etc/pf.conf** *Check /etc/pf.conf for errors, but do not load ruleset*

Packet Filtering with PF

- Rules are loaded from a file usually **/etc/pf.conf**
- Packets can be passed, redirected or dropped as they pass through an interface
- PF inspects packets based on Layer 3 (IPv4/IPV6) and Layer 4 headers (TCP, UDP, ICMP/v6)
- Can check for source/destination address, protocol (Layer 4) and source/destination port
- Rules evaluated in sequential order – top to bottom of the file

Packet Filtering with PF cont'd

- A packet is evaluated against all the rules UNLESS the key word ***quick*** is specified
- If ***quick*** is not specified then the last rule to match wins and action is taken on the packet
- There is an implicit pass all at the beginning meaning that if a packet does not match any rule then it will be passed
- You are free to circumvent this feature if you want by having a "block all" at the top of the file

Rule Syntax

- *action [direction] [log] [quick] [on interface] [af] [proto protocol] [from src_addr [port src_port]] [to dst_addr [port dst_port]] [flags tcp_flags] [state]*
- **action** – pass or block
- **direction** – in or out
- **log** – should this be logged or not
- **quick** – specified action is taken immediately
- **on interface** – name of the interface
- **inet** – address family, inet6 for ipv6
- **protocol** – tcp, udp, icmp, icmp6 or others in **/etc/protocols**
- **src_addr/dst_addr** – source port or destination address
- **src_port/dst_port** – Number between 1 – 65535 (**/etc/services**
- **tcp_flags** – eg flags S/SA look only for SYN and ACK
- **state** – whether to check state. PF checks state by default

Good practice

- Recommended to have default deny at the beginning of the file so that what you do not specify is denied by default.
  - i.e. to make it an exclusive firewall
- This is to counter the default pass rule
- Done by adding the below at the top of the file
  - **block in all**
- Also good idea to leave out the loopback interface and link local addresses
  - **set skip on lo0**
  - You can set a macro eg: ipv6_ll="fe80::/10"

Keeping state in PF

- Keeping state has many advantages including simpler rulesets and better packet filtering performance.
- PF is able to match packets moving in *either* direction to state table entries meaning that filter rules which pass returning traffic don't need to be written.
- Since packets matching stateful connections don't go through ruleset evaluation, the time PF spends processing those packets can be greatly lessened.

Keeping state cont'd

- When a rule creates state, the first packet matching the rule creates a "state" between the sender and receiver.
- Not only do packets going from the sender to receiver match the state entry and bypass ruleset evaluation, but so do the reply packets from receiver to sender.
- All *pass* rules automatically create a state entry when a packet matches the rule. This can be explicitly disabled by using the no state option.

Understanding the PF Rules

- Rules are red from top to bottom
- A matching pass rule in one direction automatically creates a matching pass rule in the other direction
  - Eg “pass out all” automatically creates a “pass in all”
- PF allows everything in or out by default and you may (or may not) want this
- The last rule to match the packet wins
- If the packet is not matched it is allowed through anyway (default allow – exclusive)
- If you don’t want this, you must set an implicit block in the direction you want to block

What about default deny?

- The recommended practice when setting up a firewall is to take a "default deny" approach.
- That is, to deny *everything* and then selectively allow certain traffic through the firewall.
- This approach is recommended because it errs on the side of caution and also makes writing a ruleset easier. the first two filter rules should be:

  block in all
  block out all

- This will block all traffic on all interfaces in either direction from anywhere to anywhere.
- **HOWEVER**, you may opt to approach your firewall rules differently depending on the scenario

Passing Traffic

```
pass in on dc0 from 192.168.0.0/24 to 192.168.0.1
pass out on dc0 from 192.168.0.1 to 192.168.0.0/24
pass in quick on em0 from 172.16.1.1 to any
pass out on em0 from any to 172.16.1.1
pass in quick on em0 inet proto icmp from any to any
pass in quick on rl0 inet proto udp from any port 53 to 192.168.0.1
block in quick all
block out quick all
```

- What will the above rules do?

Sample File

```
block in all #block stuff by default
set skip on lo0
pass in log on em0 inet proto tcp all
pass out on em0 inet proto tcp all
```

- Will UDP traffic pass? Why?
- Will ICMP Traffic pass? Why?

Sample Rules 2

```
block in all #block stuff by default
set skip on lo0
block in quick on em0 inet proto tcp from
   192.168.0.1 80 to any
pass in on em0 inet proto tcp all
```

- What happens here?

Macros

- Use macros to make rules simpler. Macros are usually identified at the top of the file ruleset. Sample Macros:

```
ext_if="em0"
int_if="em1"
LAN="192.168.0.0/24"
good_ports="{ 80, 22, 110}"
bad_ips="{ 172.16.0.0/23, 10.10.0.0/16,}"
my_pc="172.16.1.1"
```

- Macros cannot contain a hyphen however
  - so **ext-if="em0"** will not work

Macros cont'd

- Macros are then called using the $ sign in the ruleset:

```
block in all #block stuff by default
set skip on lo0
pass in on $int_if inet proto tcp from any\
   $good_ports to any
block in quick on $ext_if from any $bad_ips\
```

Sample Web Server 1

```
good_ports="{ 22, 443, 80 }"
me="192.168.0.1" #Web Server IP
set skip on lo0
block in all
pass out all
pass in on em0 inet proto tcp from any to $me port \
$good_ports
```

##This is sufficient to allow any communication that the server initiates (pass out all), allow all incoming tcp traffic to the good ports and block all other incoming traffic. The "pass out all" is needed despite PF having an implicit pass rule. Removing it will mean traffic out will not match any rule but incoming replies to conversations initiated by the server will be matched against the "block in all" rule.

Sample Web Server 2

```
good_ports="{ 22, 443, 80 }"
me="192.168.0.1" #Web server IP
set skip on lo0
pass in quick on em0 inet proto tcp from any to $me \
port $good_ports
block in all
pass out all
```

##These rules will pass the good ports first (due to quick word) then block all incoming traffic (block in all). Outgoing traffic will be allowed as long as the server sent the first packet (pass out all)

Sample Web Server 3

```
good_ports="{ 22, 443, 80 }"
me="192.168.0.1" #Web Server
set skip on lo0
pass in on em0 inet proto tcp from any to $me port \
$good_ports
block in all
pass out all
```

##These rules will block all incoming traffic (block in all) including traffic to the good ports hence this is a badly formatted ruleset. Outgoing traffic will be allowed out but incoming replies will be dropped so this a broken ruleset

Simple Mail Server example 1

```
good_ports="{ 22, 25, 110, 143, 993, 995, 443, 80 }"
good_udp="53"
block in all
pass out all
set skip on lo0
pass in on em0 inet proto tcp from $good_ports to \
any
pass in on em0 inet proto udp from any port \
$good_udp
```

##These rules will pass the good ports on tcp, pass any udp traffic with a source port of 53 then block all incoming traffic (block in all). Outgoing traffic will be allowed as long as the server sent the first packet (pass out all)

Simplest way to organize the ruleset

- Put your Macros at the top
  - If you are on IPv6 add a macro for link local (fe80::/10)
- Skip the lo0 – **set skip on lo0**
- Put the “block in all” at the top so that all traffic that you do not specify is dropped
- Allow traffic that your server initiates “pass out all”
  - However if the machine gets compromised, for example, is hacked and is used to attack other networks, you will have to change this rule
- Allow link local addresses (if you are on an IPv6 network)
- Allow the traffic and ports that you wish

Logging

- When you activate logging, a new virtual interface will be created called ***pflog0***
- Doing a tcpdump on this interface will provide details of all traffic that you have chosen to log
- You can also direct all logs to **/var/logs/pflog** by adding the following to /etc/rc.conf

  pflog_logfile="/var/log/pflog"

Other PF Features

- **Antispoofing** - when a malicious user fakes the source IP address in packets they transmit in order to either hide their real address or to impersonate another node on the network
- **Anti-SPAM –** when used with a software called **spamd** which downloads a list a blacklisted IP Addresses which can be fed to PF to block (spamd is not spamassassin)
- **Gateway Firewall –** add to /etc/rc.conf

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

References and more reading

- http://en.wikipedia.org/wiki/PF_%28firewall%29
- http://www.openbsd.org/faq/pf/filter.html
- http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/firewalls-pf.html
- http://en.wikipedia.org/wiki/Firewall_%28computing%29