14.9 Firewalls

Contributed by Joseph J. Barbish. Converted to SGML and updated by Brad Davis.

14.9.1 Introduction

All software-based firewalls provide some way to filter incoming and outgoing traffic that flows through your system. The firewall uses one or more sets of ``rules'' to inspect the network packets as they come in or go out of your network connections and either allows the traffic through or blocks it. The rules of the firewall can inspect one or more characteristics of the packets, including but not limited to the protocol type, the source or destination host address and the source or destination port.

Firewalls greatly enhance the security of your network, your applications and services. They can be used to do one of more of the following things:

- To protect and insulate the applications, services and machines of your internal network from unwanted traffic coming in from the public Internet.
- To limit or disable access from hosts of the internal network to services of the public Internet.
- To support network address translation (*NAT*), which allows your internal network to use private *IP* addresses and share a single connection to the public Internet (either with a single *IP* address or by a shared pool of automatically assigned public addresses).

14.9.2 Firewall Rule Set Types

Constructing a software application firewall rule set may seem to be trivial, but most people get it wrong. The most common mistake is to create an ``exclusive'' firewall rather than an ``inclusive'' firewall.

An exclusive firewall allows all services through except for those matching a set of rules that block certain services.

An inclusive firewall does the reverse. It only allows services matching the rules through and blocks everything else. This way you can control what services can originate behind the firewall destined for the public Internet and also control which services originating from the public Internet may access your network. Inclusive firewalls are much, much safer than exclusive firewalls.

When you use your browser to access a web site there are many internal functions that happen before your screen fills with the data from the target web site. Your browser does not receive one large file containing all the data and display format instructions at one time. Each internal function accesses the public Internet in multiple send/receive cycles of packets of information. When all the packets containing the data finally arrive, the data contained in the packets is combined together to fill your screen. Each service (*DNS*, *HTTP*, etc) has its own port number. The port number 80 is for *HTTP* services. So you can code your firewall to only allow web page session start requests originating from your *LAN* to pass through the firewall out to the public Internet.

Security can be tightened further by telling the firewall to monitor the send/receive cycles of all the packets making up that session until the session completes. These are called stateful capabilities and provides the maximum level of protection.

A firewall rule set that does not implement stateful capabilities on all the services being authorized is an insecure firewall that is still open to many of the most common methods of attack.

14.9.3 Firewall Software Applications

FreeBSD has two different firewall software products built into the base system. They are IPFILTER (i.e. also known as IPF) and IPFIREWALL (i.e. also known as IPFW). IPFIREWALL has the built in DUMMYNET traffic shaper facilities for controlling bandwidth usage. IPFILTER does not have a built in traffic shaper facility for controlling bandwidth usage, but the ALTQ port application can be used to accomplish the same function. The DUMMYNET feature and *ALTQ* is generally useful only to

and from your system, although they go about it different ways and have different rule syntaxes.

The IPFW sample rule set (found in `/etc/rc.firewall`) delivered in the basic install is outdated, complicated and does not use stateful rules on the interface facing the public Internet. It exclusively uses legacy stateless rules which only have the ability to open or close the service ports. The IPFW example stateful rules sets presented here supercede the `/etc/rc.firewall` file distributed with the system.

Stateful rules have technically advanced interrogation abilities capable of defending against the flood of different methods currently employed by attackers.

Both of these firewall software solutions IPF and IPFW still maintain their legacy heritage of their original rule processing order and reliance on non-stateful rules. These outdated concepts are not covered here, only the new, modern stateful rule construct and rule processing order is presented.

You should read about both of them and make your own decision on which one best fits your needs.

The author prefers IPFILTER because its stateful rules are much less complicated to use in a *NAT* environment and it has a built in ftp proxy that simplifies the rules to allow secure outbound FTP usage. If is also more appropriate to the knowledge level of the inexperienced firewall user.

Since all firewalls are based on interrogating the values of selected packet control fields, the creator of the firewall rules must have an understanding of how *TCP*/IP works, what the different values in the packet control fields are and how these values are used in a normal session conversation. For a good explanation go to: http://www.ipprimer.com/overview.cfm.

14.9.4 The Packet Filter Firewall

As of July 2003 the OpenBSD firewall software application known as *PF* was ported to FreeBSD 5.3. *PF* is a complete, fully featured firewall that contains *ALTQ* for bandwidth usage management in a way similar to the dummynet provides in *IPFW*. The OpenBSD project does an outstanding job of maintaining the PF users' guide that it will not be made part of this handbook firewall section as that would just be duplicated effort.

For older 5.X version of FreeBSD you can find *PF* in the FreeBSD ports collection here: `security/pf`.

More info can be found at the PF for FreeBSD web site: .

The OpenBSD PF user's guide is here: http://www.openbsd.org/faq/pf/index.html.

> **Warning:** PF in FreeBSD 5.X is at the level of OpenBSD version 3.5. The port from the FreeBSD ports collection at the level of OpenBSD version 3.4. Keep that in mind when browsing the user's guide.

14.9.4.1 Enabling PF

PF is included in the basic FreeBSD install for versions newer than 5.3 as a separate run time loadable module. PF will dynamically load its kernel loadable module when the rc.conf statement *pf_enable="YES"* is used. The loadable module was created with pflog(4) logging enabled.

14.9.4.2 Kernel options

It is not a mandatory requirement that you enable PF by compiling the following options into the FreeBSD kernel. It is only presented here as background information. Compiling PF into the kernel causes the loadable module to never be used.

Sample kernel config PF option statements are in the `/usr/src/sys/conf/NOTES` kernel source and are reproduced here:

```
device pf
device pflog
device pfsync
```

device pf tells the compile to include Packet Filter as part of its core kernel.

device pflog enables the optional pflog(4) pseudo network device which can be used to log traffic to a bpf(4) descriptor. The pflogd(8) daemon can be used to store the logging information to disk.

device pfsync enables the optional pfsync(4) pseudo network device that is used to monitor ``state changes''. As this is not part of the loadable module one has to build a custom kernel to use it.

These settings will take affect only after you have built and installed a kernel with them set.

14.9.4.3 Available rc.conf Options

You need the following statements in `/etc/rc.conf` to activate PF at boot time:

```
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
```

If you have a LAN behind this firewall and have to forward packets for the computers in the LAN or want to do NAT you have to enable the following option as well:

```
gateway_enable="YES"            # Enable as Lan gateway
```

14.9.5 The IPFILTER (IPF) Firewall

The author of IPFILTER is Darren Reed. IPFILTER is not operating system dependent. IPFILTER is a 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.

IPFILTER is based on a kernel-side firewall and *NAT* mechanism that can be controlled and monitored by userland interface programs. The firewall rules can be set or deleted with the ipf(8) utility. The *NAT* rules can be set or deleted with the ipnat(1) utility. The ipfstat(8) utility can print run-time statistics for the kernel parts of IPFILTER. The ipmon(8) program can log IPFILTER actions to the system log files.

IPF was originally written using a rule processing logic of ``the last matching rule wins'' and used only stateless type of rules. Over time IPF has been enhanced to include a ``quick'' option and a stateful ``keep state'' option which drastically modernized the rules processing logic. IPF's official documentation covers the legacy rule coding parameters and the legacy rule file processing logic. the modernized functions are only included as additional options, completely understating their benefits in producing a far superior secure firewall.

The instructions contained in this section are based on using rules that contain the ``quick'' option and the stateful ``keep state'' option. This is the basic framework for coding an inclusive firewall rule set.

An inclusive firewall only allows packets matching the rules to pass through. This way you can control what services can originate behind the firewall destine for the public Internet and also control the services which can originate from the public Internet accessing your private network. Everything else is blocked and logged by default design. Inclusive firewalls are much, much more secure than exclusive firewall rule sets and is the only rule set type covered here in.

For detailed explanation of the legacy rules processing method see: http://www.obfuscation.org/ipf/ipf-howto.html#TOC_1 and http://coombs.anu.edu.au/~avalon/ip-filter.html .

The IPF FAQ is at http://www.phildev.net/ipf/index.html.

14.9.5.1 Enabling IPF

IPF is included in the basic FreeBSD install as a separate run time loadable module. IPF will dynamically load its kernel loadable module when the rc.conf statement *ipfilter_enable="YES"* is used. The loadable module was created with logging enabled and the *default pass all* options. You do not need to compile IPF into the FreeBSD kernel just to change the default to *block all* , you can do that by just coding a block all rule at the end of your rule set.

14.9.5.2 Kernel options

It is not a mandatory requirement that you enable IPF by compiling the following options into the FreeBSD kernel. It is only presented here as background information. Compiling IPF into the kernel causes the loadable module to never be used.

Sample kernel config IPF option statements are in the `/usr/src/sys/i386/conf/LINT` kernel source and are reproduced here.

```
options IPFILTER
options IPFILTER_LOG
options IPFILTER_DEFAULT_BLOCK
```

options IPFILTER tells the compile to include IPFILTER as part of its core kernel.

options IPFILTER_LOG enables the option to have IPF log traffic by writing to the ipl packet logging pseudo--device for every rule that has the *log* keyword.

options IPFILTER_DEFAULT_BLOCK changes the default behavior so any packet not matching a firewall *pass* rule gets blocked.

These settings will take affect only after you have built and installed a kernel with them set.

14.9.5.3 Available rc.conf Options

You need the following statements in `/etc/rc.conf` to activate IPF at boot time:

```
ipfilter_enable="YES"             # Start ipf firewall
ipfilter_rules="/etc/ipf.rules"   # loads rules definition text file
ipmon_enable="YES"                # Start IP monitor log
ipmon_flags="-Ds"                # D = start as daemon
                                  # s = log to syslog
                                  # v = log tcp window, ack, seq
                                  # n = map IP & port to names
```

If you have a LAN behind this firewall that uses the reserved private IP address ranges, then you need to add the following to enable *NAT* function.

```
gateway_enable="YES"              # Enable as Lan gateway
ipnat_enable="YES"                # Start ipnat function
ipnat_rules="/etc/ipnat.rules"    # rules definition file for ipnat
```

14.9.5.4 IPF

The ipf command is used to load your rules file. Normally you create a file containing your custom rules and use this command to replace in mass the currently running firewall internal rules.

```
ipf -Fa -f /etc/ipf.rules
```

-Fa means flush all internal rules tables.

-f means this is the file to read for the rules to load.

This gives you the ability to make changes to their custom rules file, run the above IPF command thus updating the running firewall with a fresh copy of all the rules without having to reboot the system. This method is very convenient for testing new rules as the procedure can be executed as many times as needed.

See the ipf(8) man page for details on the other flags available with this command.

The ipf(8) command expects the rules file to be a standard text file. It will not accept a rules file written as a script with symbolic substitution.

There is a way to build IPF rules that utilities the power of script symbolic substitution. See the Building Rule Script section.

14.9.5.5 IPFSTAT

The default behavior of ipfstat(8) is to retrieve and display the totals of the accumulated statistics gathered as a result of applying the user coded rules against packets going in and out of the firewall since it was last started, or since the last time the accumulators were reset to zero by `ipf -z` command.

See the ipfstat(8) manual page for details.

The default ipfstat(8) command output will look something like this:

```
input packets: blocked 99286 passed 1255609 nomatch 14686 counted 0
```

```
input packets logged: blocked 99286 passed 0
output packets logged: blocked 0 passed 0
packets logged: input 0 output 0
log failures: input 3898 output 0
fragment state(in): kept 0 lost 0
fragment state(out): kept 0 lost 0
packet state(in): kept 169364 lost 0
packet state(out): kept 431395 lost 0
ICMP replies: 0 TCP RSTs sent: 0
Result cache hits(in): 1215208 (out): 1098963
IN Pullups succeeded: 2 failed: 0
OUT Pullups succeeded: 0 failed: 0
Fastroute successes: 0 failures: 0
TCP cksum fails(in): 0 (out): 0
Packet log flags set: (0)
```

When supplied with either -i for inbound or -o for outbound, it will retrieve and display the appropriate list of filter rules currently installed and in use by the kernel.

`ipfstat -in` displays the inbound internal rules table with rule number.

`ipfstat -on` displays the outbound internal rules table with the rule number.

The output will look something like this:

```
@1 pass out on xl0 from any to any
@2 block out on dc0 from any to any
@3 pass out quick on dc0 proto tcp/udp from any to any keep state
```

`ipfstat -ih` displays the inbound internal rules table prefixed each rule with count of how many times the rule was matched.

`ipfstat -oh` displays the outbound internal rules table prefixed each rule with count of how many times the rule was matched.

The output will look something like this:

```
2451423 pass out on xl0 from any to any
354727 block out on dc0 from any to any
430918 pass out quick on dc0 proto tcp/udp from any to any keep state
```

One of the most important functions of the ipfstat command is the -T flag which activates the display state table in a way similar to the way top(1) shows the FreeBSD running process table. When your firewall is under attack this function gives you the ability to identify, drill down to, and see the attacking packets. The optional sub-flags give the ability to select destination or source IP, port, protocol, you want to monitor in real time. See the ipfstat(8) man page for details.

14.9.5.6 IPMON

In order for `ipmon` to properly work, the kernel option IPFILTER_LOG must be turned on. This command has 2 different modes it can be used in. Native mode is the default mode when you type the command on the command line without the -D flag.

Daemon mode is for when you want to have a continuous system log file available so you can review logging of past events. This is how FreeBSD and IPFILTER are configured to work together. FreeBSD has a built in facility to automatically rotate syslogs. That is why outputting the log information to syslogd is better than the default of outputting to a regular file. In `rc.conf` file you see the ipmon_flags statement uses the "-Ds" flags

```
ipmon_flags="-Ds" # D = start as daemon
                  # s = log to syslog
                  # v = log tcp window, ack, seq
                  # n = map IP & port to names
```

The benefits of logging are obvious. It provides the ability to review, after the fact, information like: what packets had been dropped, what addresses they came from and where they were going. These all give you a significant edge in tracking down attackers.

Even with the logging facility enabled, IPF will not generate any rule logging on its own. The firewall administrator decides what rules in the rule set he wants to log and adds the log keyword to those rules. Normally only deny rules are logged.

Its very customary to include a default deny everything rule with the log keyword included as your

rules in the rule set.

14.9.5.7 IPMON Logging

Syslogd uses its own special method for segregation of log data. It uses special grouping called ``facility'' and ``level.'' IPMON in -Ds mode uses Local0 as the ``facility'' name. All IPMON logged data goes to Local0. The following levels can be used to further segregate the logged data if desired.

```
LOG_INFO - packets logged using the "log" keyword as the action rather than pass or block.
LOG_NOTICE - packets logged which are also passed
LOG_WARNING - packets logged which are also blocked
LOG_ERR - packets which have been logged and which can be considered short
```

To setup IPFILTER to log all data to `/var/log/ipfilter.log`, you will need to create the file. The following command will do that:

```
touch /var/log/ipfilter.log
```

The syslog function is controlled by definition statements in the `/etc/syslog.conf` file. The `syslog.conf` file offers considerable flexibility in how syslog will deal with system messages issued by software applications like IPF.

Add the following statement to `/etc/syslog.conf` :

```
Local0.* /var/log/ipfilter.log
```

The *Local0.** means to write all the logged messages to the coded file location.

To activate the changes to `/etc/syslog.conf` you can reboot or bump the syslog task into re-reading `/etc/syslog.conf` by `kill -HUP <pid>`. You get the pid (i.e. process number) by listing the tasks with the `ps -ax` command. Find syslog in the display and the pid is the number in the left column.

Do not forget to change `/etc/newsyslog.conf` to rotate the new log you just created above.

14.9.5.8 The Format of Logged Messages

Messages generated by ipmon consist of data fields separated by white space. Fields common to all messages are:

1. The date of packet receipt.
2. The time of packet receipt. This is in the form HH:MM:SS.F, for hours, minutes, seconds, and fractions of a second (which can be several digits long).
3. The name of the interface the packet was processed on, e.g. `dc0`.
4. The group and rule number of the rule, e.g. *@0:17*.

These can be viewed with ipfstat -in.

1. The action: p for passed, b for blocked, S for a short packet, n did not match any rules, L for a log rule. The order of precedence in showing flags is: S, p, b, n, L. A capital P or B means that the packet has been logged due to a global logging setting, not a particular rule.
2. The addresses. This is actually three fields: the source address and port (separated by a comma), the -> symbol, and the destination address and port. 209.53.17.22,80 -> 198.73.220.17,1722.
3. PR followed by the protocol name or number, e.g. PR tcp.
4. len followed by the header length and total length of the packet, e.g. len 20 40.

If the packet is a *TCP* packet, there will be an additional field starting with a hyphen followed by letters corresponding to any flags that were set. See the ipmon(8) manual page for a list of letters and their flags.

If the packet is an ICMP packet, there will be two fields at the end, the first always being ``ICMP'', and the next being the ICMP message and sub-message type, separated by a slash, e.g. ICMP 3/3 for a port unreachable message.

14.9.5.9 Building the Rule Script

Some experienced IPF users create a file containing the rules and code them in a manner compatible with running them as a script with symbolic substitution. The major benefit of doing this is you only have to change the value associated with the symbolic name and when the script is run all the rules containing the symbolic name will have the value substituted in the rules. Being a script, you can use symbolic substitution to code frequent used values and substitute them in multiple rules. You will see this in the following example.

The script syntax used here is compatible with the sh, csh, and tcsh shells.

Symbolic substitution fields are prefixed with a dollar sign $.

Symbolic fields do not have the $ prefix

The value to populate the Symbolic field must be enclosed with "double quotes".

Start your rule file with something like this:

```
############ Start of IPF rules script ######################

oif="dc0"            # name of the outbound interface
odns="192.0.2.11"    # ISP's dns server IP address Symbolic>
myip="192.0.2.7"     # My Static IP address from ISP
ks="keep state"
fks="flags S keep state"

# You can use this same to build the /etc/ipf.rules file
#cat >> /etc/ipf.rules << EOF

# exec ipf command and read inline data, stop reading
# when word EOF is found. There has to be one line
# after the EOF line to work correctly.
/sbin/ipf -Fa -f - << EOF

# Allow out access to my ISP's Domain name server.
pass out quick on $oif proto tcp from any to $odns port = 53 $fks
pass out quick on $oif proto udp from any to $odns port = 53 $ks

# Allow out non-secure standard www function
pass out quick on $oif proto tcp from $myip to any port = 80 $fks

# Allow out secure www function https over TLS SSL
pass out quick on $oif proto tcp from $myip to any port = 443 $fks
EOF
################## End of IPF rules script ######################
```

That is all there is to it. The rules are not important in this example, how the Symbolic substitution field are populated and used are. If the above example was in /etc/ipf.rules.script file, you could reload these rules by entering on the command line.

```
sh /etc/ipf.rules.script
```

There is one problem with using a rules file with embedded symbolics. IPF has no problem with it, but the rc startup scripts that read `rc.conf` will have problems.

To get around this limitation with a rc scripts, remove the following line:

```
ipfilter_rules=
```

Add a script like the following to your `/usr/local/etc/rc.d/` startup directory. The script should have a obvious name like `loadipfrules.sh` . The `.sh` extension is mandatory.

```
#!/bin/sh
sh /etc/ipf.rules.script
```

The permission on this script file must be read, write, exec for owner root.

```
chmod 700 /usr/local/etc/rc.d/ipf.loadrules.sh
```

Now when you system boots your IPF rules will be loaded using the script.

14.9.5.10 IPF Rule Sets

A rule set is a group of ipf rules coded to pass or block packets based on the values contained in the packet. The bi-directional exchange of packets between hosts comprises a session conversation. The firewall rule set processes the packet 2 times, once on its arrival from the public Internet host and again as it leaves for its return trip back to the public Internet host. Each tcp/ip service (i.e. telnet, www, mail, etc.) is predefined by its protocol, source and destination IP address, or the source and destination port number. This is the basic selection criteria used to create rules which will pass or block services.

IPF was originally written using a rules processing logic of 'the last matching rule wins' and used only stateless rules. Over time IPF has been enhanced to include a 'quick' option and a stateful 'keep state' option which drastically modernized the rules processing logic.

The instructions contained in this section is based on using rules that contain the 'quick' option. and the stateful 'keep state' option. This is the basic framework for coding an inclusive firewall rule set.

An inclusive firewall only allows services matching the rules through. This way you can control what services can originate behind the firewall destined for the public Internet and also control the services which can originate from the public Internet accessing your private network. Everything else is blocked and logged by default design. Inclusive firewalls are much, much securer than exclusive firewall rule sets and is the only rule set type covered herein.

> **Note:** Warning, when working with the firewall rules, always, always do it from the root console of the system running the firewall or you can end up locking your self out.

14.9.5.11 Rule Syntax

The rule syntax presented here has been simplified to only address the modern stateful rule context and 'first matching rule wins' logic. For the complete legacy rule syntax description see the online ipf man page at ipf(8)

is used to mark the start of a comment and may appear at the end of a rule line or on its own lines. Blank lines are ignored.

Rules contain keywords, These keywords have to be coded in a specific order from left to right on the line. Keywords are identified in bold type. Some keywords have sub-options which may be keywords them selves and also include more sub-options. Each of the headings in the below syntax has a bold section header which expands on the content.

ACTION IN-OUT OPTIONS SELECTION STATEFUL PROTO SRC_ADDR,DST_ADDR OBJECT PORT_NUM TCP_FLAG STATEFUL

ACTION = block | pass

IN-OUT = in | out

OPTIONS = log | quick | on interface-name

SELECTION = proto value | source/destination IP | port = number | flags flag-value

PROTO = tcp/udp | udp | tcp | icmp

SRC_ADD,DST_ADDR = all | from object to object

OBJECT = IP address | any

PORT_NUM = port number

TCP_FLAG = S

STATEFUL = keep state

14.9.5.11.1 ACTION

The action indicates what to do with the packet if it matches the rest of the filter rule. Each rule *must* have a action. The following actions are recognized:

block indicates that the packet should be dropped if the selection parameters match the packet.

pass indicates that the packet should exit the firewall if the selection parameters match the packet.

14.9.5.11.2 IN-OUT

This is a mandatory requirement that each filter rule explicitly state which side of the I/O it is to be used on. The next keyword must be either in or out and one or the other has to be coded or the rule will not pass syntax check.

in means this rule is being applied against an inbound packet which has just been received on the interface facing the public Internet.

out means this rule is being applied against an outbound packet destined for the interface facing the public Internet.

14.9.5.11.3 OPTIONS

> **Note:** These options must be used in the order shown here.

log indicates that the packet header will be written to the ipl log (as described in the LOGGING section below) if the selection parameters match the packet.

quick indicates that if the selection parameters match the packet, this rule will be the last rule checked, allowing a "short-circuit" path to avoid processing any following rules for this packet. This option is a mandatory requirement for the modernized rules processing logic.

on indicates the interface name to be incorporated into the selection parameters. Interface names are as displayed by ifconfig. Using this option, the rule will only match if the packet is going through that interface in the specified direction (in/out). This option is a mandatory requirement for the modernized rules processing logic.

When a packet is logged, the headers of the packet are written to the IPL packet logging pseudo-device. Immediately following the log keyword, the following qualifiers may be used (in this order):

body indicates that the first 128 bytes of the packet contents will be logged after the headers.

first If the 'log' keyword is being used in conjunction with a "keep state" option, it is recommended that this option is also applied so that only the triggering packet is logged and not every packet which there after matches the 'keep state' information.

14.9.5.11.4 SELECTION

The keywords described in this section are used to describe attributes of the packet to be interrogated when determining whether rules match or don't match. There is a keyword subject, and it has sub-option keywords, one of which has to be selected. The following general-purpose attributes are provided for matching, and must be used in this order:

14.9.5.11.5 PROTO

Proto is the subject keyword, it must be coded along with one of it.s corresponding keyword sub-option values. The value allows a specific protocol to be matched against. This option is a mandatory requirement for the modernized rules processing logic.

tcp/udp | udp | tcp | icmp or any protocol names found in /etc/protocols are recognized and may be used. The special protocol keyword tcp/udp may be used to match either a *TCP* or a UDP packet, and has been added as a convenience to save duplication of otherwise identical rules.

14.9.5.11.6 SRC_ADDR/DST_ADDR

The 'all' keyword is essentially a synonym for "from any to any" with no other match parameters.

from src to dst The from and to keywords are used to match against IP addresses. Rules must specify BOTH source and destination parameters. .any. is a special keyword that matches any IP address. As in 'from any to any' or 'from 0.0.0.0/0 to any' or 'from any to 0.0.0.0/0' or 'from 0.0.0.0 to any' or 'from any to 0.0.0.0'

IP addresses may be specified as a dotted IP address numeric form/mask-length, or as single dotted IP address numeric form.

There isn't a way to match ranges of IP addresses which do not express themselves easily as mask-length. See this link for help on writing mask-length: http://jodies.de/ipcalc

14.9.5.11.7 PORT

If a port match is included, for either or both of source and destination, then it is only applied to *TCP* and UDP packets. When composing port comparisons, either the service name from /etc/services or an integer port number may be used. When the port appears as part of the from object, it matches the source port number, when it appears as part of the to object, it matches the destination port number. The use of the port option with the .to. object is a mandatory requirement for the modernized rules processing logic. As in 'from any to any port = 80'

Port comparisons may be done in a number of forms, with a number of comparison operators, or port ranges may be specified.

port "=" | "!=" | "<" | ">" | "<=" | ">=" | "eq" | "ne" | "lt" | "gt" | "le" | "ge".

To specify port ranges, port "<>" | "><"

> **Warning:** Following the source and destination matching parameters, the following two parameters are mandatory requirements for the modernized rules processing logic.

14.9.5.11.8 *TCP*_FLAG

Flags are only effective for *TCP* filtering. The letters represents one of the possible flags that can be interrogated in the *TCP* packet header.

The modernized rules processing logic uses the 'flags S' parameter to identify the tcp session start request.

14.9.5.11.9 STATEFUL

'keep state' indicates that on a pass rule, any packets that match the rules selection parameters is to activate the stateful filtering facility.

> **Note:** This option is a mandatory requirement for the modernized rules processing logic.

14.9.5.12 Stateful Filtering

Stateful filtering treats traffic as a bi-directional exchange of packets comprising a session conversation. When activated keep-state dynamically generates internal rules for each anticipated packet being exchanged during the bi-directional session conversation. It has the interrogation abilities to determine if the session conversation between the originating sender and the destination are following the valid procedure of bi-directional packet exchange. Any packets that do not properly fit the session conversation template are automatically rejected as impostors.

Keep state will also allow ICMP packets related to a *TCP* or UDP session through. So if you get ICMP type 3 code 4 in response to some web surfing allowed out by a keep state rule, they will be automatically allowed in. Any packet that IPF can be certain is part of a active session, even if it is a different protocol, will be let in.

What happens is:

Packets destined to go out the interface connected to the public Internet are first checked against the dynamic state table, if the packet matches the next expected packet comprising in a active session conversation, then it exits the firewall and the state of the session conversation flow is updated in the dynamic state table, the remaining packets get checked against the outbound rule set.

Packets coming in to the interface connected to the public Internet are first checked against the dynamic state table, if the packet matches the next expected packet comprising a active session conversation, then it exits the firewall and the state of the session conversation flow is updated in the dynamic state table, the remaining packets get checked against the inbound rule set.

When the conversation completes it is removed from the dynamic state table.

Stateful filtering allows you to focus on blocking/passing new sessions. If the new session is passed, all its subsequent packets will be allowed through automatically and any impostors automatically rejected. If a new session is blocked, none of its subsequent packets will be allowed through. Stateful filtering has technically advanced interrogation abilities capable of defending against the flood of different attack methods currently employed by attackers.

14.9.5.13 Inclusive Rule set Example

The following rule set is an example of how to code a very secure inclusive type of firewall. An inclusive firewall only allows services matching pass rules through and blocks all other by default. All firewalls have at the minimum two interfaces which have to have rules to allow the firewall to function.

All Unix flavored systems including FreeBSD are designed to use interface l0 and IP address 127.0.0.1 for internal communication with in the FreeBSD operating system. The firewall rules must contain rules to allow free unmolested movement of these special internally used packets.

The interface which faces the public Internet, is the one which you code your rules to authorize and control access out to the public Internet and access requests arriving from the public Internet. This can be your .user ppp. tun0 interface or your NIC card that is cabled to your DSL or cable modem.

In cases where one or more than one NICs are cabled to Private LANs (local area networks) behind the firewall, those interfaces must have a rule coded to allow free unmolested movement of packets originating from those LAN interfaces.

The rules should be first organized into three major sections, all the free unmolested interfaces, public interface outbound, and the public interface inbound.

The order of the rules in each of the public interface sections should be in order of the most used rules being placed before less often used rules with the last rule in the section being a block log all packets on that interface and direction.

The Outbound section in the following rule set only contains 'pass' rules which contain selection values that uniquely identify the service that is authorized for public Internet access. All the rules have the 'quick', 'on', 'proto', 'port', and 'keep state' option coded. The 'proto tcp' rules have the 'flag' option included to identify the session start request as the triggering packet to activate the stateful facility.

The Inbound section has all the blocking of undesirable packets first for two different reasons. First is these things being blocked may be part of an otherwise valid packet which may be allowed in by the later authorized service rules. Second reason is that by having a rule that explicitly blocks selected packets that I receive on an infrequent bases and don't want to see in the log, this keeps them from being caught by the last rule in the section which blocks and logs all packets which have fallen through the rules. The last rule in the section which blocks and logs all packets is how you create the legal evidence needed to prosecute the people who are attacking your system.

Another thing you should take note of, is there is no response returned for any of the undesirable stuff, their packets just get dropped and vanish. This way the attackers has no knowledge if his packets have reached your system. The less the attackers can learn about your system the more secure it is. The inbound 'nmap OS fingerprint' attempts rule I log the first occurrence because this is something a attacker would do.

Any time you see log messages on a rule with .log first. you should do an ipstat -h command to see the number of times the rule has been matched so you know if your are being flooded, i.e. under attack.

When you log packets with port numbers you do not recognize, go to http://www.securitystats.com/tools/portsearch.php and do a port number lookup to find what the purpose of that port number is.

Check out this link for port numbers used by Trojans http://www.simovits.com/trojans/trojans.html

The following rule set is a complete very secure 'inclusive' type of firewall rule set that I have used on my system. You can not go wrong using this rule set for your own. Just comment out any pass rules for services to don.t want to authorize.

If you see messages in your log that you want to stop seeing just add a block rule in the inbound section.

You have to change the `dc0` interface name in every rule to the interface name of the Nic card that connects your system to the public Internet. For user PPP it would be `tun0`.

Add the following statements to `/etc/ipf.rules`:

```
################################################################
# No restrictions on Inside Lan Interface for private network
# Not needed unless you have Lan
################################################################

#pass out quick on xl0 all
#pass in quick on xl0 all

################################################################
# No restrictions on Loopback Interface
################################################################
pass in quick on lo0 all
pass out quick on lo0 all

################################################################
# Interface facing Public Internet (Outbound Section)
# Interrogate session start requests originating from behind the
# firewall on the private network
# or from this gateway server destine for the public Internet.
################################################################

# Allow out access to my ISP's Domain name server.
# xxx must be the IP address of your ISP.s DNS.
# Dup these lines if your ISP has more than one DNS server
# Get the IP addresses from /etc/resolv.conf file
pass out quick on dc0 proto tcp from any to xxx port = 53 flags S keep state
pass out quick on dc0 proto udp from any to xxx port = 53 keep state

# Allow out access to my ISP's DHCP server for cable or DSL networks.
# This rule is not needed for .user ppp. type connection to the
# public Internet, so you can delete this whole group.
# Use the following rule and check log for IP address.
# Then put IP address in commented out rule & delete first rule
pass out log quick on dc0 proto udp from any to any port = 67 keep state
#pass out quick on dc0 proto udp from any to z.z.z.z port = 67 keep state


# Allow out non-secure standard www function
pass out quick on dc0 proto tcp from any to any port = 80 flags S keep state

# Allow out secure www function https over TLS SSL
pass out quick on dc0 proto tcp from any to any port = 443 flags S keep state

# Allow out send & get email function
pass out quick on dc0 proto tcp from any to any port = 110 flags S keep state
pass out quick on dc0 proto tcp from any to any port = 25 flags S keep state

# Allow out Time
pass out quick on dc0 proto tcp from any to any port = 37 flags S keep state

# Allow out nntp news
pass out quick on dc0 proto tcp from any to any port = 119 flags S keep state

# Allow out gateway & LAN users non-secure FTP ( both passive & active modes)
# This function uses the IPNAT built in FTP proxy function coded in
# the nat rules file to make this single rule function correctly.
# If you want to use the pkg_add command to install application packages
# on your gateway system you need this rule.
pass out quick on dc0 proto tcp from any to any port = 21 flags S keep state

# Allow out secure FTP, Telnet, and SCP
# This function is using SSH (secure shell)
pass out quick on dc0 proto tcp from any to any port = 22 flags S keep state

# Allow out non-secure Telnet
pass out quick on dc0 proto tcp from any to any port = 23 flags S keep state

# Allow out FBSD CVSUP function
pass out quick on dc0 proto tcp from any to any port = 5999 flags S keep state

# Allow out ping to public Internet
pass out quick on dc0 proto icmp from any to any icmp-type 8 keep state
```

```
pass out quick on dc0 proto tcp from any to any port = 43 flags S keep state

# Block and log only the first occurrence of everything
# else that.s trying to get out.
# This rule enforces the block all by default logic.
block out log first quick on dc0 all

################################################################
# Interface facing Public Internet (Inbound Section)
# Interrogate packets originating from the public Internet
# destine for this gateway server or the private network.
################################################################

# Block all inbound traffic from non-routable or reserved address spaces
block in quick on dc0 from 192.168.0.0/16 to any    #RFC 1918 private IP
block in quick on dc0 from 172.16.0.0/12 to any     #RFC 1918 private IP
block in quick on dc0 from 10.0.0.0/8 to any        #RFC 1918 private IP
block in quick on dc0 from 127.0.0.0/8 to any       #loopback
block in quick on dc0 from 0.0.0.0/8 to any         #loopback
block in quick on dc0 from 169.254.0.0/16 to any    #DHCP auto-config
block in quick on dc0 from 192.0.2.0/24 to any      #reserved for docs
block in quick on dc0 from 204.152.64.0/23 to any   #Sun cluster interconnect
block in quick on dc0 from 224.0.0.0/3 to any       #Class D & E multicast

##### Block a bunch of different nasty things. ############
# That I don't want to see in the log

# Block frags
block in quick on dc0 all with frags

# Block short tcp packets
block in quick on dc0 proto tcp all with short

# block source routed packets
block in quick on dc0 all with opt lsrr
block in quick on dc0 all with opt ssrr

# Block nmap OS fingerprint attempts
# Log first occurrence of these so I can get their IP address
block in log first quick on dc0 proto tcp from any to any flags FUP

# Block anything with special options
block in quick on dc0 all with ipopts

# Block public pings
block in quick on dc0 proto icmp all icmp-type 8

# Block ident
block in quick on dc0 proto tcp from any to any port = 113

# Block all Netbios service. 137=name, 138=datagram, 139=session
# Netbios is MS/Windows sharing services.
# Block MS/Windows hosts2 name server requests 81
block in log first quick on dc0 proto tcp/udp from any to any port = 137
block in log first quick on dc0 proto tcp/udp from any to any port = 138
block in log first quick on dc0 proto tcp/udp from any to any port = 139
block in log first quick on dc0 proto tcp/udp from any to any port = 81

# Allow traffic in from ISP's DHCP server. This rule must contain
# the IP address of your ISP.s DHCP server as it.s the only
# authorized source to send this packet type. Only necessary for
# cable or DSL configurations. This rule is not needed for
# .user ppp. type connection to the public Internet.
# This is the same IP address you captured and
# used in the outbound section.
pass in quick on dc0 proto udp from z.z.z.z to any port = 68 keep state

# Allow in standard www function because I have apache server
pass in quick on dc0 proto tcp from any to any port = 80 flags S keep state

# Allow in non-secure Telnet session from public Internet
# labeled non-secure because ID/PW passed over public Internet as clear text.
# Delete this sample group if you do not have telnet server enabled.
#pass in quick on dc0 proto tcp from any to any port = 23 flags S keep state

# Allow in secure FTP, Telnet, and SCP from public Internet
# This function is using SSH (secure shell)
pass in quick on dc0 proto tcp from any to any port = 22 flags S keep state
```

```
# coming into the firewall. The logging of only the first
# occurrence stops a .denial of service. attack targeted
# at filling up your log file space.
# This rule enforces the block all by default logic.
block in log first quick on dc0 all
################### End of rules file ###################################
```

14.9.5.14 *NAT*

NAT stands for Network Address Translation. To those familiar with Linux, this concept is called IP Masquerading, *NAT* and IP Masquerading are the same thing. One of the many things the IPF *NAT* function enables, is the ability to have a private Local Area Network (LAN) behind the firewall sharing a single ISP assigned IP address to the public Internet.

You ask why would someone want to do this. ISPs normally assign a dynamic IP address to their non-commercial users. Dynamic means the IP address can be different each time you dial in and logon to your ISP, or for cable and DSL modem users when you power off and then power on your modems you can get assigned a different IP address. This IP address is how you are known to the public Internet.

Now lets say you have 5 PCs at home and each one needs Internet access. You would have to pay your ISP for an individual Internet account for each PC and have 5 phone lines.

With *NAT* you only need a single account with your ISP, then cable your other 4 PC.s to a switch and the switch to the NIC in your FreeBSD system which is going to service your LAN as a gateway. *NAT* will automatically translate the private LAN IP address for each separate PC on the LAN to the single public IP address as it exits the firewall bound for the public Internet. It also does the reverse translation for returning packets.

NAT is most often accomplished without the approval, or knowledge, of your ISP and in most cases is grounds for your ISP terminating your account if found out. Commercial users pay a lot more for their Internet connection and usually get assigned a block of static IP address which never change. The ISP also expects and consents to their Commercial customers using *NAT* for their internal private LANs.

There is a special range of IP addresses reserved for *NAT*ed private LAN IP address. According to RFC 1918, you can use the following IP ranges for private nets which will never be routed directly to the public Internet.

Start IP	`10.0.0.0`	- Ending IP	`10.255.255.255`
Start IP	`172.16.0.0`	- Ending IP	`172.31.255.255`
Start IP	`192.168.0.0`	- Ending IP	`192.168.255.255`

14.9.5.15 IP*NAT*

NAT rules are loaded by using the ipnat command. Typically the *NAT* rules are stored in `/etc/ipnat.rules` . See ipnat(1) for details.

When changing the *NAT* rules after *NAT* has been started, Make your changes to the file containing the nat rules, then run ipnat command with the -CF flags to delete the internal in use *NAT* rules and flush the contents of the translation table of all active entries.

To reload the *NAT* rules issue a command like this:

```
ipnat -CF -f /etc/ipnat.rules
```

To display some statistics about your *NAT*, use this command:

```
ipnat -s
```

To list the *NAT* table's current mappings, use this command:

```
ipnat -l
```

To turn verbose mode on, and display information relating to rule processing and active rules/table entries:

```
ipnat -v
```

14.9.5.16 IP*NAT* Rules

NAT rules are very flexible and can accomplish many different things to fit the needs of commercial and home users.

The rule syntax presented here has been simplified to what is most commonly used in a non-commercial environment. For a complete rule syntax description see the man ipf page at ipnat(5).

The syntax for a *NAT* rule looks something like this:

```
map IF LAN_IP_RANGE -> PUBLIC_ADDRESS
```

The keyword `map' starts the rule.

Replace *IF* with the external interface.

The *LAN_IP_RANGE* is what your internal clients use for IP Addressing, usually this is something like `192.168.1.0/24`.

The *PUBLIC_ADDRESS* can either be the external IP address or the special keyword `0.32', which means to use the IP address assigned to *IF*.

14.9.5.17 How *NAT* works

A packet arrives at the firewall from the LAN with a public destination. It passes through the outbound filter rules, *NAT* gets his turn at the packet and applies its rules top down, first matching rule wins. *NAT* tests each of its rules against the packets interface name and source IP address. When a packets interface name matches a *NAT* rule then the [source IP address, i.e. private Lan IP address] of the packet is checked to see if it falls within the IP address range specified to the left of the arrow symbol on the *NAT* rule. On a match the packet has its source IP address rewritten with the public IP address obtained by the `0.32' keyword. *NAT* posts a entry in its internal *NAT* table so when the packet returns from the public Internet it can be mapped back to its original private IP address and then passed to the filter rules for processing.

14.9.5.18 Enabling IP*NAT*

To enable IP*NAT* add these statements to `/etc/rc.conf`

To enable your machine to route traffic between interfaces.

```
gateway_enable="YES"
```

To start IP*NAT* automatically each time:

```
ipnat_enable="YES"
```

To specify where to load the IP*NAT* rules from

```
ipnat_rules="/etc/ipnat.rules"
```

14.9.5.19 *NAT* for a very large LAN

For networks that have large numbers of PC's on the Lan or networks with more that a single LAN the process of funneling all those private IP address into a single public IP address becomes a resource problem that may cause problems with same port numbers being used many times across many *NAT*ed LAN PC's causing collisions. There are 2 ways to relieve this resource problem.

14.9.5.19.1 Assigning Ports to Use

BLAH

```
map dc0 192.168.1.0/24 -> 0.32
```

In the above rule the packet's source port is unchanged as the packet passes through IP*NAT*. By adding the portmap keyword you can tell IP*NAT* to only use source ports in a range. For example the following rule will tell IP*NAT* to modify the source port to be within that range.

```
map dc0 192.168.1.0/24 -> 0.32 portmap tcp/udp 20000:60000
```

Additionally we can make things even easier by using the `auto' keyword to tell IP*NAT* to determine by itself which ports are available to use:

```
map dc0 192.168.1.0/24 -> 0.32 portmap tcp/udp auto
```

14.9.5.19.2 Using a pool of public addresses

In very large LANs there comes a point where there are just too many LAN addresses to fit into a single public address. By changing the following rule:

```
map dc0 192.168.1.0/24 -> 204.134.75.1
```

Currently this rule maps all connections through `204.134.75.1`. This can be changed to specify a range:

```
map dc0 192.168.1.0/24 -> 204.134.75.1-10
```

Or a subnet using CIDR notation such as:

```
map dc0 192.168.1.0/24 -> 204.134.75.0/24
```

14.9.5.20 Port Redirection

An very common practice is to have a web server, email server, database server and DNS sever each segregated to a different PC on the LAN. In this case the traffic from these servers still have to be *NAT*ed, but there has to be some way to direct the inbound traffic to the correct LAN PC's. IP*NAT* has the redirection facilities of *NAT* to solve this problem. Lets say you have your web server on LAN address `10.0.10.25` and your single public IP address is `20.20.20.5` you would code the rule like this:

```
map dc0 20.20.20.5/32 port 80 -> 10.0.10.25 port 80
```

or

```
map dc0 0/32 port 80 -> 10.0.10.25 port 80
```

or for a LAN DNS Server on LAN address of `10.0.10.33` that needs to receive public DNS requests

```
map dc0 20.20.20.5/32 port 53 -> 10.0.10.33 port 53 udp
```

14.9.5.21 FTP and *NAT*

FTP is a dinosaur left over from the time before the Internet as it is know today, when research universities were leased lined together and FTP was used to share files among research Scientists. This was a time when data security was not even an idea yet. Over the years the FTP protocol became buried into the backbone of the emerging Internet and its username and password being sent in clear text was never changed to address new security concerns. FTP has two flavors, it can run in active mode or passive mode. The difference is in how the data channel is acquired. Passive mode is more secure as the data channel is acquired be the ordinal ftp session requester. For a real good explanation of FTP and the different modes see http://www.slacksite.com/other/ftp.html

14.9.5.21.1 IP*NAT* Rules

IP*NAT* has a special built in FTP proxy option which can be specified on the *NAT* map rule. It can monitor all outbound packet traffic for FTP active or passive start session requests and dynamically create temporary filter rules containing only the port number really in use for the data channel. This eliminates the security risk FTP normally exposes the firewall to from having large ranges of high order port numbers open.

This rule will handle all the traffic for the internal LAN:

```
map dc0 10.0.10.0/29 -> 0/32 proxy port 21 ftp/tcp
```

This rule handles the FTP traffic from the gateway.

```
map dc0 0.0.0.0/0 -> 0/32 proxy port 21 ftp/tcp
```

This rule handles all non-FTP traffic from the internal LAN.

```
map dc0 10.0.10.0/29 -> 0/32
```

The FTP map rule goes before our regular map rule. All packets are tested against the first rule from the top. Matches on interface name, then private LAN source IP address, and then is it a FTP packet. If all that matches then the special FTP proxy creates temp filter rules to let the FTP session packets pass in and out, in addition to also *NAT*ing the FTP packets. All LAN packets that

on interface and source IP, then are *NAT*ed.

14.9.5.21.2 IP*NAT* FTP Filter Rules

Only one filter rule is needed for FTP if the *NAT* FTP proxy is used.

Without the FTP Proxy you will need the following three rules

```
# Allow out LAN PC client FTP to public Internet
# Active and passive modes
pass out quick on rl0 proto tcp from any to any port = 21 flags S keep state

# Allow out passive mode data channel high order port numbers
pass out quick on rl0 proto tcp from any to any port > 1024 flags S keep state

# Active mode let data channel in from FTP server
pass in quick on rl0 proto tcp from any to any port = 20 flags S keep state
```

14.9.5.21.3 FTP *NAT* Proxy Bug

As of FreeBSD 4.9 which includes IPFILTER version 3.4.31 the FTP proxy works as documented during the FTP session until the session is told to close. When the close happens packets returning from the remote FTP server are blocked and logged coming in on port 21. The *NAT* FTP/proxy appears to remove its temp rules prematurely, before receiving the response from the remote FTP server acknowledging the close. Posted problem report to ipf mailing list.

Solution is to add filter rule like this one to get rid of these unwanted log messages or do nothing and ignore FTP inbound error messages in your log. Not like you do FTP session to the public Internet all the time, so this is not a big deal.

```
Block in quick on rl0 proto tcp from any to any port = 21
```

14.9.6 IPFW

The IPFIREWALL (IPFW) is a FreeBSD sponsored firewall software application authored and maintained by FreeBSD volunteer staff members. It uses the legacy Stateless rules and a legacy rule coding technique to achieve what is referred to as Simple Stateful logic.

The IPFW stateless rule syntax is empowered with technically sophisticated selection capabilities which far surpasses the knowledge level of the customary firewall installer. IPFW is targeted at the professional user or the advanced technical computer hobbyist who have advanced packet selection requirements. A high degree of detailed knowledge into how different protocols use and create their unique packet header information is necessary before the power of the IPFW rules can be unleashed. Providing that level of explanation is out of the scope of this section of the handbook.

IPFW is composed of 7 components, the primary component is the kernel firewall filter rule processor and its integrated packet accounting facility, the logging facility, the 'divert' rule which triggers the *NAT* facility, and the advanced special purpose facilities, the dummynet traffic shaper facilities, the 'fwd rule' forward facility, the bridge facility, and the ipstealth facility.

14.9.6.1 Enabling IPFW

IPFW is included in the basic FreeBSD install as a separate run time loadable module. IPFW will dynamically load the kernel module when the `rc.conf` statement *firewall_enable="YES"* is used. You do not need to compile IPFW into the FreeBSD kernel unless you want *NAT* function enabled.

After rebooting your system with *firewall_enable="YES"* in `rc.conf` the following white highlighted message is displayed on the screen as part of the boot process:

```
IP packet filtering initialized, divert disabled, rule-based forwarding
enabled, default to deny, logging disabled
```

You can disregard this message as it is out dated and no longer is the true status of the IPFW loadable module. The loadable module really does have logging ability compiled in.

To set the verbose logging limit, There is a knob you can set in `/etc/sysctl.conf` by adding this statement, logging will be enabled on future reboots.

```
net.inet.ip.fw.verbose_limit=5
```

14.9.6.2 Kernel Options

It is not a mandatory requirement that you enable IPFW by compiling the following options into the FreeBSD kernel unless you need *NAT* function. It is presented here as background information.

```
options     IPFIREWALL
```

This option enables IPFW as part of the kernel

```
options     IPFIREWALL_VERBOSE
```

Enables logging of packets that pass through IPFW and have the 'log' keyword specified in the rule set.

```
options     IPFIREWALL_VERBOSE_LIMIT=5
```

This specifies the default number of packets from a particular rule is to be logged. Without this option, each repeated occurrences of the same packet will be logged, and eventually consuming all the free disk space resulting in services being denied do to lack of resources. The 5 is the number of consecutive times to log evidence of this unique occurrence.

```
options     IPFIREWALL_DEFAULT_TO_ACCEPT
```

This option will allow everything to pass through the firewall by default. Which is a good idea when you are first setting up your firewall.

```
options     IPV6FIREWALL
options     IPV6FIREWALL_VERBOSE
options     IPV6FIREWALL_VERBOSE_LIMIT
options     IPV6FIREWALL_DEFAULT_TO_ACCEPT
```

These options are exactly the same as the IPv4 options but they are for IPv6. If you don't use IPv6 you might want to use IPV6FIREWALL without any rules to block all IPv6

```
options     IPDIVERT
```

This enables the use of *NAT* functionality.

> **Note:** If you don't include IPFIREWALL_DEFAULT_TO_ACCEPT or set your rules to allow incoming packets you will block all packets going to and from this machine.

14.9.6.3 `/etc/rc.conf` Options

If you don't have IPFW compliled into your kernel you will need to load it with the following statement in your `/etc/rc.conf`:

```
firewall_enable="YES"
```

Set the script to run to activate your rules:

```
firewall_script="/etc/ipfw.rules"
```

Enable logging:

```
firewall_logging="YES"
```

14.9.6.4 The IPFW Command

The ipfw command is the normal vehicle for making manual single rule additions or deletions to the firewall active internal rules while it is running. The problem with using this method is once your system is shutdown or halted all the rules you added or changed or deleted are lost. Writing all your rules in a file and using that file to load the rules at boot time, or to replace in mass the currently running firewall rules with changes you made to the files content is the recommended method used here.

The IPFW command is still a very useful to display the running firewall rules to the console screen. The IPFW accounting facility dynamically creates a counter for each rule that counts each packet that matches the rule. During the process of testing a rule, listing the rule with its counter is the only way of determining if the rule is functioning.

To list all the rules in sequence:

```
ipfw list
```

To list all the rules with a time stamp of when the last time the rule was matched:

```
ipfw -t list
```

To list the accounting information, packet count for matched rules along with the rules themselves. The first column is the rule number, followed by the number of outgoing matched packets, followed by the number of incoming matched packets, and then the rule itself.

```
ipfw -a list
```

List the dynamic rules in addition to the static rules:

```
ipfw -d list
```

Also show the expired dynamic rules:

```
ipfw -d -e list
```

Zero the counters:

```
ipfw zero
```

Zero the counters for just rule *NUM* :

```
ipfw zero NUM
```

14.9.6.5 IPFW Rule Sets

A rule set is a group of ipfw rules coded to allow or deny packets based on the values contained in the packet. The bi-directional exchange of packets between hosts comprises a session conversation. The firewall rule set processes the packet 2 times, once on its arrival from the public Internet host and again as it leaves for its return trip back to the public Internet host. Each tcp/ip service (i.e. telnet, www, mail, etc.) is predefined by its protocol, and port number. This is the basic selection criteria used to create rules which will allow or deny services.

When a packet enters the firewall it is compared against the first rule in the rule set and progress one rule at a time moving from top to bottom of the set in ascending rule number sequence order. When the packet matches a rule selection parameters, the rules action field value is executed and the search of the rule set terminates for that packet. This is referred to as the 'first match wins' search method. If the packet does not match any of the rules, it gets caught by the mandatory ipfw default rule, number 65535 which denies all packets and discards them without any reply back to the originating destination.

The instructions contained here are based on using rules that contain the stateful 'keep state', 'limit', 'in'/'out', and via options. This is the basic framework for coding an inclusive type firewall rule set.

An inclusive firewall only allows services matching the rules through. This way you can control what services can originate behind the firewall destine for the public Internet and also control the services which can originate from the public Internet accessing your private network. Everything else is denied by default design. Inclusive firewalls are much, much more secure than exclusive firewall rule sets and is the only rule set type covered here in.

> **Warning:** When working with the firewall rules be careful, you can end up locking your self out.

14.9.6.5.1 Rule Syntax

The rule syntax presented here has been simplified to what is necessary to create a standard inclusive type firewall rule set. For a complete rule syntax description see the online ipfw(8) man page.

Rules contain keywords, These keywords have to be coded in a specific order from left to right on the line. Keywords are identified in bold type. Some keywords have sub-options which may be keywords them selves and also include more sub-options.

is used to mark the start of a comment and may appear at the end of a rule line or on its own lines. Blank lines are ignored.

CMD RULE# ACTION LOGGING SELECTION STATEFUL

14.9.6.5.1.1 CMD

Each rule has to be prefixed with 'add' to add the rule to the internal table.

14.9.6.5.1.2 RULE#

Each rule has to have a rule number to go with it.

14.9.6.5.1.3 ACTION

A rule can be associated with one of the following actions, which will be executed when the packet matches the selection criterion of the rule.

allow | *accept* | *pass* | *permit*

These all mean the same thing which is to allow packets that match the rule to exit the firewall rule processing. The search terminates at this rule.

check-state

Checks the packet against the dynamic rules table. If a match is found, execute the action associated with the rule which generated this dynamic rule, otherwise move to the next rule. The Check-state rule does not have selection criterion. If no check-state rule is present in the rule set, the dynamic rules table is checked at the first keep-state or limit rule.

deny | *drop*

Both words mean the same thing which is to discard packets that match this rule. The search terminates.

14.9.6.5.1.4 Logging

log or *logamount*

When a packet matches a rule with the log keyword, a message will be logged to syslogd with a facility name of SECURITY. The logging only occurs if the number of packets logged so far for that particular rule does not exceed the logamount parameter. If no logamount is specified, the limit is taken from the sysctl variable net.inet.ip.fw.verbose_limit. In both cases, a value of zero removes the logging limit. Once the limit is reached, logging can be re-enabled by clearing the logging counter or the packet counter for that rule, see the ipfw reset log command. Note: logging is done after all other packet matching conditions have been successfully verified, and before performing the final action (accept, deny) on the packet. It is up to you to decide which rules you want to enable logging on.

14.9.6.5.1.5 Selection

The keywords described in this section are used to describe attributes of the packet to be interrogated when determining whether rules match or don't match the packet. The following general-purpose attributes are provided for matching, and must be used in this order:

udp | *tcp* | *icmp*

or any protocol names found in /etc/protocols are recognized and may be used. The value specified is protocol to be matched against. This is a mandatory requirement.

from src to dst

The from and to keywords are used to match against IP addresses. Rules must specify BOTH source and destination parameters. any is a special keyword that matches any IP address. me is a special keyword that matches any IP address configured on an interface in your FreeBSD system to represent the PC the firewall is running on. (i.e. this box) As in from me to any or from any to me or from 0.0.0.0/0 to any or from any to 0.0.0.0/0 or from 0.0.0.0 to any or from any to 0.0.0.0 or from me to 0.0.0.0. IP addresses are specified as a dotted IP address numeric form/mask-length, or as single dotted IP address numeric form. This is a mandatory requirement. See this link for help on writing mask-lengths. http://jodies.de/ipcalc

port number

For protocols which support port numbers (such as *TCP* and UDP). It is mandatory that you code the port number of the service you want to match on. Service names (from `/etc/services`) may be used instead of numeric port values.

in | *out*

Matches incoming or outgoing packets, respectively. The in and out are keywords and it is mandatory that you code one or the other as part of your rule matching criterion.

via IF

Matches packets going through the interface specified by exact name. The via keyword causes the interface to always be checked as part of the match process.

setup

This is a mandatory keyword that identifies the session start request for *TCP* packets.

keep-state

This is a mandatory> keyword. Upon a match, the firewall will create a dynamic rule, whose default behavior is to match bidirectional traffic between source and destination IP/port using the same protocol.

limit {src-addr | src-port | dst-addr | dst-port}

The firewall will only allow *N* connections with the same set of parameters as specified in the rule. One or more of source and destination addresses and ports can be specified. The 'limit' and 'keep-state' can not be used on same rule. Limit provides the same stateful function as 'keep-state' plus its own functions.

14.9.6.5.2 Stateful Rule Option

Stateful filtering treats traffic as a bi-directional exchange of packets comprising a session conversation. It has the interrogation abilities to determine if the session conversation between the originating sender and the destination are following the valid procedure of bi-directional packet exchange. Any packets that do not properly fit the session conversation template are automatically rejected as impostors.

'check-state' is used to identify where in the IPFW rules set the packet is to be tested against the dynamic rules facility. On a match the packet exits the firewall to continue on its way and a new rule is dynamic created for the next anticipated packet being exchanged during this bi-directional session conversation. On a no match the packet advances to the next rule in the rule set for testing.

The dynamic rules facility is vulnerable to resource depletion from a SYN-flood attack which would open a huge number of dynamic rules. To counter this attack, FreeBSD version 4.5 added another new option named limit. This option is used to limit the number of simultaneous session conversations by interrogating the rules source or destinations fields as directed by the limit option and using the packet's IP address found there, in a search of the open dynamic rules counting the number of times this rule and IP address combination occurred, if this count is greater that the value specified on the limit option, the packet is discarded.

14.9.6.5.3 Logging Firewall Messages

The benefits of logging are obvious, provides the ability to review after the fact the rules you activated logging on which provides information like, what packets had been dropped, what addresses they came from, where they were going, giving you a significant edge in tracking down attackers.

Even with the logging facility enabled, IPFW will not generate any rule logging on it's own. The firewall administrator decides what rules in the rule set he wants to log and adds the log verb to those rules. Normally only deny rules are logged. Like the deny rule for incoming *ICMP* pings. It's very customary to duplicate the ipfw default deny everything rule with the log verb included as your last rule in the rule set. This way you get to see all the packets that did not match any of the rules in the rule set.

Logging is a two edged sword, if you're not careful, you can lose yourself in the over abundance of log data and fill your disk up with growing log files. DoS attacks that fill up disk drives is one of the oldest attacks around. These log message are not only written to syslogd, but also are displayed on the root console screen and soon become very annoying.

The *IPFIREWALL_VERBOSE_LIMIT=5* kernel option limits the number of consecutive messages sent to the system logger syslogd, concerning the packet matching of a given rule. When this option is enabled in the kernel, the number of consecutive messages concerning a particular rule is capped at the number specified. There is nothing to be gained from 200 log messages saying the same identical thing. For instance, 5 consecutive messages concerning a particular rule would be logged to syslogd, the remainder identical consecutive messages would be counted and posted to the syslogd with a phrase like this:

```
last message repeated 45 times
```

All logged packets messages are written by default to `/var/log/security` file, which is defined in the `/etc/syslog.conf` file.

14.9.6.5.4 Building Rule Script

Most experienced IPFW users create a file containing the rules and code them in a manner compatible with running them as a script. The major benefit of doing this is the firewall rules can be refreshed in mass without the need of rebooting the system to activate the new rules. This method is very convenient in testing new rules as the procedure can be executed as many times as needed. Being a script, you can use symbolic substitution to code frequent used values and substitution them in multiple rules. You will see this in the following example.

The script syntax used here is compatible with the 'sh', 'csh', 'tcsh' shells. Symbolic substitution fields are prefixed with a dollar sign $. Symbolic fields do not have the $ prefix. The value to populate the Symbolic field must be enclosed to "double quotes".

Start your rules file like this:

```
############### start of example ipfw rules script #############
#
ipfw -q -f flush       # Delete all rules
# Set defaults
oif="tun0"             # out interface
odns="192.0.2.11"      # ISP's dns server IP address
cmd="ipfw -q add "     # build rule prefix
ks="keep-state"        # just too lazy to key this each time
$cmd 00500 check-state
$cmd 00502 deny all from any to any frag
$cmd 00501 deny tcp from any to any established
$cmd 00600 allow tcp from any to any 80 out via $oif setup $ks
 $cmd 00610 allow tcp from any to $odns 53 out via $oif setup $ks
 $cmd 00611 allow udp from any to $odns 53 out via $oif $ks
################### End of example ipfw rules script #############
```

That is all there is to it. The rules are not important in this example, how the Symbolic substitution field are populated and used are.

If the above example was in `/etc/ipfw.rules` file, you could reload these rules by entering on the command line.

```
sh /etc/ipfw.rules
```

The `/etc/ipfw.rules` file could be located any where you want and the file could be named any thing you would like.

The same thing could also be accomplished by running these commands by hand:

```
ipfw -q -f flush
ipfw -q add check-state
ipfw -q add deny all from any to any frag
ipfw -q add deny tcp from any to any established
ipfw -q add allow tcp from any to any 80 out via tun0 setup keep-state
ipfw -q add allow tcp from any to 192.0.2.11 53 out via tun0 setup keep-state
ipfw -q add 00611 allow udp from any to 192.0.2.11 53 out via tun0 keep-state
```

14.9.6.5.5 Stateful Ruleset

The following non-*NAT*ed rule set is a example of how to code a very secure 'inclusive' type of firewall. An inclusive firewall only allows services matching pass rules through and blocks all other by default. All firewalls have at the minimum two interfaces which have to have rules to allow the firewall to function.

All UNIX® flavored operating systems, FreeBSD included, are designed to use interface lo and IP address `127.0.0.1` for internal communication with in FreeBSD. The firewall rules must contain rules to allow free unmolested movement of these special internally used packets.

The interface which faces the public Internet, is the one which you code your rules to authorize and control access out to the public Internet and access requests arriving from the public Internet. This can be your ppp tun0 interface or your NIC that is connected to your DSL or cable modem.

In cases where one or more than one NIC are connected to a private LANs behind the firewall, those interfaces must have rules coded to allow free unmolested movement of packets originating from those LAN interfaces.

The rules should be first organized into three major sections, all the free unmolested interfaces, public interface outbound, and the public interface inbound.

The order of the rules in each of the public interface sections should be in order of the most used rules being placed before less often used rules with the last rule in the section being a block log all packets on that interface and direction.

The Outbound section in the following rule set only contains 'allow' rules which contain selection values that uniquely identify the service that is authorized for public Internet access. All the rules have the, proto, port, in/out, via and keep state option coded. The 'proto tcp' rules have the 'setup' option included to identify the start session request as the trigger packet to be posted to the keep state stateful table.

The Inbound section has all the blocking of undesirable packets first for 2 different reasons. First is these things being blocked may be part of an otherwise valid packet which may be allowed in by the later authorized service rules. Second reason is that by having a rule that explicitly blocks selected packets that I receive on an infrequent bases and don't want to see in the log, this keeps them from being caught by the last rule in the section which blocks and logs all packets which have fallen through the rules. The last rule in the section which blocks and logs all packets is how you create the legal evidence needed to prosecute the people who are attacking your system.

Another thing you should take note of, is there is no response returned for any of the undesirable stuff, their packets just get dropped and vanish. This way the attackers has no knowledge if his packets have reached your system. The less the attackers can learn about your system the more secure it is. When you log packets with port numbers you do not recognize, go to http://www.securitystats.com/tools/portsearch.php and do a port number lookup to find what the purpose of that port number is. Check out this link for port numbers used by Trojans: http://www.simovits.com/trojans/trojans.html .

14.9.6.5.6 An Example Inclusive Ruleset

The following non-*NAT*ed rule set is a complete inclusive type ruleset. You can not go wrong using this rule set for you own. Just comment out any pass rules for services to don't want. If you see messages in your log that you want to stop seeing just add a deny rule in the inbound section. You have to change the 'dc0' interface name in every rule to the interface name of the NIC that connects your system to the public Internet. For user ppp it would be 'tun0'.

You will see a pattern in the usage of these rules.

- All statements that are a request to start a session to the public Internet use keep-state.
- All the authorized services that originate from the public Internet have the limit option to stop flooding.
- All rules use in or out to clarify direction.
- All rules use via interface name to specify the interface the packet is traveling over.

The following rules go into `/etc/ipfw.rules`.

```
################ Start of IPFW rules file ###############################
# Flush out the list before we begin.
ipfw -q -f flush

# Set rules command prefix
cmd="ipfw -q add"
pif="dc0"     # public interface name of Nic card
                        # facing the public Internet

#############################################################
# No restrictions on Inside Lan Interface for private network
# Not needed unless you have Lan.
# Change xl0 to your Lan Nic card interface name
#############################################################
#$cmd 00005 allow all from any to any via xl0

#############################################################
```

```
#####################################################################
$cmd 00010 allow all from any to any via lo0

#####################################################################
# Allow the packet through if it has previous been added to the
# the "dynamic" rules table by a allow keep-state statement.
#####################################################################
$cmd 00015 check-state

#####################################################################
# Interface facing Public Internet (Outbound Section)
# Interrogate session start requests originating from behind the
# firewall on the private network or from this gateway server
# destine for the public Internet.
#####################################################################

# Allow out access to my ISP's Domain name server.
# x.x.x.x must be the IP address of your ISP.s DNS
# Dup these lines if your ISP has more than one DNS server
# Get the IP addresses from /etc/resolv.conf file
$cmd 00110 allow tcp from any to x.x.x.x 53 out via $pif setup keep-state
$cmd 00111 allow udp from any to x.x.x.x 53 out via $pif keep-state

# Allow out access to my ISP's DHCP server for cable/DSL configurations.
# This rule is not needed for .user ppp. connection to the public Internet.
# so you can delete this whole group.
# Use the following rule and check log for IP address.
# Then put IP address in commented out rule & delete first rule
$cmd 00120 allow log udp from any to any 67 out via $pif keep-state
#$cmd 00120 allow udp from any to x.x.x.x 67 out via $pif keep-state

# Allow out non-secure standard www function
$cmd 00200 allow tcp from any to any 80 out via $pif setup keep-state

# Allow out secure www function https over TLS SSL
$cmd 00220 allow tcp from any to any 443 out via $pif setup keep-state

# Allow out send & get email function
$cmd 00230 allow tcp from any to any 25 out via $pif setup keep-state
$cmd 00231 allow tcp from any to any 110 out via $pif setup keep-state

# Allow out FBSD (make install & CVSUP) functions
# Basically give user root "GOD" privileges.
$cmd 00240 allow tcp from me to any out via $pif setup keep-state uid root

# Allow out ping
$cmd 00250 allow icmp from any to any out via $pif keep-state

# Allow out Time
$cmd 00260 allow tcp from any to any 37 out via $pif setup keep-state

# Allow out nntp news (i.e. news groups)
$cmd 00270 allow tcp from any to any 119 out via $pif setup keep-state

# Allow out secure FTP, Telnet, and SCP
# This function is using SSH (secure shell)
$cmd 00280 allow tcp from any to any 22 out via $pif setup keep-state

# Allow out whois
$cmd 00290 allow tcp from any to any 43 out via $pif setup keep-state

# deny and log everything else that.s trying to get out.
# This rule enforces the block all by default logic.
$cmd 00299 deny log all from any to any out via $pif

#####################################################################
# Interface facing Public Internet (Inbound Section)
# Interrogate packets originating from the public Internet
# destine for this gateway server or the private network.
#####################################################################

# Deny all inbound traffic from non-routable reserved address spaces
$cmd 00300 deny all from 192.168.0.0/16 to any in via $pif  #RFC 1918 private IP
$cmd 00301 deny all from 172.16.0.0/12 to anyin via $pif     #RFC 1918 private IP
$cmd 00302 deny all from 10.0.0.0/8 to anyin via $pif          #RFC 1918 private IP
$cmd 00303 deny all from 127.0.0.0/8 to anyin via $pif        #loopback
$cmd 00304 deny all from 0.0.0.0/8 to anyin via $pif            #loopback
$cmd 00305 deny all from 169.254.0.0/16 to anyin via $pif   #DHCP auto-config
$cmd 00306 deny all from 192.0.2.0/24 to anyin via $pif       #reserved for docs
```

```
$cmd 00308 deny all from 224.0.0.0/3 to anyin via $pif         #Class D & E multicast

# Deny public pings
$cmd 00310 deny icmp from any to anyin via $pif

# Deny ident
$cmd 00315 deny tcp from any to any 113in via $pif

# Deny all Netbios service. 137=name, 138=datagram, 139=session
# Netbios is MS/Windows sharing services.
# Block MS/Windows hosts2 name server requests 81
$cmd 00320 deny tcp from any to any 137in via $pif
$cmd 00321 deny tcp from any to any 138in via $pif
$cmd 00322 deny tcp from any to any 139in via $pif
$cmd 00323 deny tcp from any to any 81 in via $pif

# Deny any late arriving packets
$cmd 00330 deny all from any to any frag in via $pif

# Deny ACK packets that did not match the dynamic rule table
$cmd 00332 deny tcp from any to any established in via $pif

# Allow traffic in from ISP's DHCP server. This rule must contain
# the IP address of your ISP.s DHCP server as it.s the only
# authorized source to send this packet type.
# Only necessary for cable or DSL configurations.
# This rule is not needed for .user ppp. type connection to
# the public Internet. This is the same IP address you captured
# and used in the outbound section.
#$cmd 00360 allow udp from any to x.x.x.x 67 in via $pif keep-state

# Allow in standard www function because I have apache server
$cmd 00400 allow tcp from any to me 80 in via $pif setup limit src-addr 2

# Allow in secure FTP, Telnet, and SCP from public Internet
$cmd 00410 allow tcp from any to me 22 in via $pif setup limit src-addr 2

# Allow in non-secure Telnet session from public Internet
# labeled non-secure because ID & PW are passed over public
# Internet as clear text.
# Delete this sample group if you do not have telnet server enabled.
$cmd 00420 allow tcp from any to me 23 in via $pif setup limit src-addr 2

# Reject & Log all incoming connections from the outside
$cmd 00499 deny log all from any to any in via $pif

# Everything else is denied by default
# deny and log all packets that fell through to see what they are
$cmd 00999 deny log all from any to any
################ End of IPFW rules file ###############################
```

14.9.6.5.7 An Example *NAT* and Stateful Ruleset

There are some additional configuration statements that need to be enabled to activate the *NAT* function of IPFW. The kernel source needs 'option divert' statement added to the other IPFIREWALL statements compiled into a custom kernel.

In addition to the normal IPFW options in `/etc/rc.conf`, the following are needed.

```
natd_enable="YES"                   # Enable NATD function
natd_interface="rl0"                # interface name of public Internet NIC
natd_flags="-dynamic -m"            # -m = preserve port numbers if possible
```

Utilizing stateful rules with divert natd rule (Network Address Translation) greatly complicates the rule set coding logic. The positioning of the check-state, and 'divert natd' rules in the rule set becomes very critical. This is no longer a simple fall-through logic flow. A new action type is used, called 'skipto'. To use the skipto command it is mandatory that you number each rule so you know exactly where the skipto rule number is you are really jumping to.

The following is an uncommented example of one coding method, selected here to explain the sequence of the packet flow through the rule sets.

The processing flow starts with the first rule from the top of the rule file and progress one rule at a time deeper into the file until the end is reach or the packet being tested to the selection criteria matches and the packet is released out of the firewall. It's important to take notice of the location

and inbound packets so their entries in the keep-state dynamic table always register the private Lan IP address. Next notice that all the allow and deny rules specified the direction the packet is going (IE outbound or inbound) and the interface. Also notice that all the start outbound session requests all skipto rule 500 for the network address translation.

Lets say a LAN user uses their web browser to get a web page. Web pages use port 80 to communicate over. So the packet enters the firewall, It does not match 100 because it is headed out not in. It passes rule 101 because this is the first packet so it has not been posted to the keep-state dynamic table yet. The packet finally comes to rule 125 a matches. It's outbound through the NIC facing the public Internet. The packet still has it's source IP address as a private Lan IP address. On the match to this rule, two action take place. The keep-state option will post this rule into the keep-state dynamic rules table and the specified action is executed. The action is part of the info posted to the dynamic table. In this case it's "skipto rule 500". Rule 500 *NAT*s the packet IP address and out it goes. Remember this, this is very important. This packet makes it's way to the destination and returns and enters the top of the rule set. This time it does match rule 100 and has it destination IP address mapped back to it's corresponding Lan IP address. It then is processed by the check-state rule, it's found in the table as an existing session conversation and released to the LAN. It goes to the LAN PC that sent it and a new packet is sent requesting another segment of the data from the remote server. This time it gets checked by the check-state rule and it's outbound entry is found, the associated action, 'skipto 500', is executed. the packet jumps to rule 500 gets *NAT*ed and released on it's way out.

On the inbound side, everything coming in that is part of an existing session conversation is being automatically handled by the check-state rule and the properly placed divert natd rules. All we have to address is denying all the bad packets and only allowing in the authorized services. Lets say there is a apache server running on the firewall box and we want people on the public Internet to be able to access the local web site. The new inbound start request packet matches rule 100 and its IP address is mapped to LAN IP for the firewall box. The packet is them matched against all the nasty things we want to check for and finally matches against rule 425. On a match two things occur, the limit option is an extension to keep-state. The packet rule is posted to the keep-state dynamic table but this time any new session requests originating from that source IP address is limited to 2. This defends against DoS attacks of service running on the specified port number. The action is allow so the packet is released to the LAN. On return the check-state rule recognizes the packet as belonging to an existing session conversation sends it to rule 500 for *NAT*ing and released to outbound interface.

Example Ruleset #1:

```
#!/bin/sh
cmd="ipfw -q add"
skip="skipto 500"
pif=rl0
ks="keep-state"
good_tcpo="22,25,37,43,53,80,443,110,119"

ipfw -q -f flush

$cmd 002 allow all from any to any via xl0  # exclude Lan traffic
$cmd 003 allow all from any to any via lo0  # exclude loopback traffic

$cmd 100 divert natd ip from any to any in via $pif
$cmd 101 check-state

# Authorized outbound packets
$cmd 120 $skip udp from any to xx.168.240.2 53 out via $pif $ks
$cmd 121 $skip udp from any to xx.168.240.5 53 out via $pif $ks
$cmd 125 $skip tcp from any to any $good_tcpo out via $pif setup $ks
$cmd 130 $skip icmp from any to any  out via $pif $ks
$cmd 135 $skip udp from any to any 123 out via $pif $ks


# Deny all inbound traffic from non-routable reserved address spaces
$cmd 300 deny all from 192.168.0.0/16  to any in via $pif  #RFC 1918 private IP
$cmd 301 deny all from 172.16.0.0/12   to any in via $pif  #RFC 1918 private IP
$cmd 302 deny all from 10.0.0.0/8      to any in via $pif  #RFC 1918 private IP
$cmd 303 deny all from 127.0.0.0/8     to any in via $pif  #loopback
$cmd 304 deny all from 0.0.0.0/8       to any in via $pif  #loopback
$cmd 305 deny all from 169.254.0.0/16  to any in via $pif  #DHCP auto-config
$cmd 306 deny all from 192.0.2.0/24    to any in via $pif  #reserved for docs
$cmd 307 deny all from 204.152.64.0/23 to any in via $pif  #Sun cluster
$cmd 308 deny all from 224.0.0.0/3     to any in via $pif  #Class D & E multicast

# Authorized inbound packets
$cmd 400 allow udp from xx.70.207.54 to any 68 in $ks
$cmd 420 allow tcp from any to me 80 in via $pif setup limit src-addr 1
```

```
$cmd 450 deny log ip from any to any

# This is skipto location for outbound stateful rules
$cmd 500 divert natd ip from any to any out via $pif
$cmd 510 allow ip from any to any

######################## end of rules  #################
```

The following is pretty much the same as above but, uses a self documenting coding style full of description comments to help the inexperienced IPFW rule writer to better understand what the rules are doing.

Example Ruleset #2:

```
#!/bin/sh
################ Start of IPFW rules file ###############################
# Flush out the list before we begin.
ipfw -q -f flush

# Set rules command prefix
cmd="ipfw -q add"
skip="skipto 800"
pif="rl0"     # public interface name of Nic card
              # facing the public Internet

#################################################################
# No restrictions on Inside Lan Interface for private network
# Change xl0 to your Lan Nic card interface name
#################################################################
$cmd 005 allow all from any to any via xl0

#################################################################
# No restrictions on Loopback Interface
#################################################################
$cmd 010 allow all from any to any via lo0

#################################################################
# check if packet is inbound and nat address if it is
#################################################################
$cmd 014 divert natd ip from any to any in via $pif

#################################################################
# Allow the packet through if it has previous been added to the
# the "dynamic" rules table by a allow keep-state statement.
#################################################################
$cmd 015 check-state

#################################################################
# Interface facing Public Internet (Outbound Section)
# Interrogate session start requests originating from behind the
# firewall on the private network or from this gateway server
# destine for the public Internet.
#################################################################

# Allow out access to my ISP's Domain name server.
# x.x.x.x must be the IP address of your ISP's DNS
# Dup these lines if your ISP has more than one DNS server
# Get the IP addresses from /etc/resolv.conf file
$cmd 020 $skip tcp from any to x.x.x.x 53 out via $pif setup keep-state


# Allow out access to my ISP's DHCP server for cable/DSL configurations.
$cmd 030 $skip udp from any to x.x.x.x 67 out via $pif keep-state

# Allow out non-secure standard www function
$cmd 040 $skip tcp from any to any 80 out via $pif setup keep-state

# Allow out secure www function https over TLS SSL
$cmd 050 $skip tcp from any to any 443 out via $pif setup keep-state

# Allow out send & get email function
$cmd 060 $skip tcp from any to any 25 out via $pif setup keep-state
$cmd 061 $skip tcp from any to any 110 out via $pif setup keep-state

# Allow out FreeBSD (make install & CVSUP) functions
```

```
$cmd 070 $skip tcp from me to any out via $pif setup keep-state uid root

# Allow out ping
$cmd 080 $skip icmp from any to any out via $pif keep-state

# Allow out Time
$cmd 090 $skip tcp from any to any 37 out via $pif setup keep-state

# Allow out nntp news (i.e. news groups)
$cmd 100 $skip tcp from any to any 119 out via $pif setup keep-state

# Allow out secure FTP, Telnet, and SCP
# This function is using SSH (secure shell)
$cmd 110 $skip tcp from any to any 22 out via $pif setup keep-state

# Allow out whois
$cmd 120 $skip tcp from any to any 43 out via $pif setup keep-state

# Allow ntp time server
$cmd 130 $skip udp from any to any 123 out via $pif keep-state

################################################################
# Interface facing Public Internet (Inbound Section)
# Interrogate packets originating from the public Internet
# destine for this gateway server or the private network.
################################################################

# Deny all inbound traffic from non-routable reserved address spaces
$cmd 300 deny all from 192.168.0.0/16  to any in via $pif  #RFC 1918 private IP
$cmd 301 deny all from 172.16.0.0/12   to any in via $pif  #RFC 1918 private IP
$cmd 302 deny all from 10.0.0.0/8      to any in via $pif  #RFC 1918 private IP
$cmd 303 deny all from 127.0.0.0/8     to any in via $pif  #loopback
$cmd 304 deny all from 0.0.0.0/8       to any in via $pif  #loopback
$cmd 305 deny all from 169.254.0.0/16  to any in via $pif  #DHCP auto-config
$cmd 306 deny all from 192.0.2.0/24    to any in via $pif  #reserved for docs
$cmd 307 deny all from 204.152.64.0/23 to any in via $pif  #Sun cluster
$cmd 308 deny all from 224.0.0.0/3     to any in via $pif  #Class D & E multicast

# Deny ident
$cmd 315 deny tcp from any to any 113 in via $pif

# Deny all Netbios service. 137=name, 138=datagram, 139=session
# Netbios is MS/Windows sharing services.
# Block MS/Windows hosts2 name server requests 81
$cmd 320 deny tcp from any to any 137 in via $pif
$cmd 321 deny tcp from any to any 138 in via $pif
$cmd 322 deny tcp from any to any 139 in via $pif
$cmd 323 deny tcp from any to any 81  in via $pif

# Deny any late arriving packets
$cmd 330 deny all from any to any frag in via $pif

# Deny ACK packets that did not match the dynamic rule table
$cmd 332 deny tcp from any to any established in via $pif

# Allow traffic in from ISP's DHCP server. This rule must contain
# the IP address of your ISP's DHCP server as it's the only
# authorized source to send this packet type.
# Only necessary for cable or DSL configurations.
# This rule is not needed for 'user ppp' type connection to
# the public Internet. This is the same IP address you captured
# and used in the outbound section.
$cmd 360 allow udp from x.x.x.x to any 68 in via $pif keep-state

# Allow in standard www function because I have apache server
$cmd 370 allow tcp from any to me 80 in via $pif setup limit src-addr 2

# Allow in secure FTP, Telnet, and SCP from public Internet
$cmd 380 allow tcp from any to me 22 in via $pif setup limit src-addr 2

# Allow in non-secure Telnet session from public Internet
# labeled non-secure because ID & PW are passed over public
# Internet as clear text.
# Delete this sample group if you do not have telnet server enabled.
$cmd 390 allow tcp from any to me 23 in via $pif setup limit src-addr 2

# Reject & Log all unauthorized incoming connections from the public Internet
$cmd 400 deny log all from any to any in via $pif
```

```
$cmd 450 deny log all from any to any out via $pif

# This is skipto location for outbound stateful rules
$cmd 800 divert natd ip from any to any out via $pif
$cmd 801 allow ip from any to any

# Everything else is denied by default
# deny and log all packets that fell through to see what they are
$cmd 999 deny log all from any to any
################ End of IPFW rules file ################################
```

Prev **Kerberos5** | Home Up | Next OpenSSL