HP-UX IPFilter Version A.03.05.14 Administrator's Guide

HP-UX 11i v1 and HP-UX 11i v2

December 2006

HP Networking

Manufacturing Part Number : B9901-90031
E1206

United States

Legal Notices

The information in this document is subject to change without notice.

Hewlett-Packard makes no warranty of any kind with regard to this manual, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be held liable for errors contained herein or direct, indirect, special, incidental, or consequential damages in connection with the furnishing, performance, or use of this material.

Warranty

A copy of the specific warranty terms applicable to your Hewlett-Packard product and replacement parts can be obtained from your local Sales and Service Office.

U.S. Government License

Proprietary computer software. Valid license from HP required for possession, use, or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

Trademark Notices

UNIX® is a registered trademark of The Open Group.

Contents

Contents

4. Firewall Building Concepts

5. HP-UX IPFilter Utilities

Contents

Contents

Preface: About This Document

This document describes how to install, configure, and troubleshoot HP-UX IPFilter version A.03.05.14.

The document printing date and part number indicate the document's current edition. The printing date will change when a new edition is printed. Minor changes might be made at reprint without changing the printing date. The document part number will change when extensive changes are made.

Document updates might be issued between editions to correct errors or document product changes. To ensure that you receive the updated or new editions, you should subscribe to the appropriate product support service. See your HP sales representative for details.

The latest version of this document can be found online at http://docs.hp.com.

Intended Audience

This document is intended for network managers or network security administrators who install, configure, and troubleshoot HP-UX IPFilter on HP 9000 systems. Administrators are expected to have knowledge of HP-UX operating system concepts, commands, and configuration.

This document is not a tutorial.

New and Changed Documentation in This Edition

The documentation reflects the following changes to the HP-UX IPFilter product:

- HP-UX IPFilter support is added for IPv6.

This document has the following structural changes:

- New chapter added for IPv6 support, Chapter 6, "HP-UX and IPv6 Support," on page 103.

Publishing History

Table 1 Publishing History Details

Document Manufacturing Part Number	Operating Systems Supported	Supported Product Versions	Publication Date
B9901-90031	11i v1 11i v2	A.03.05.14	December 2006
B9901-90021	11.0 11i v1 11i v2	A.03.05.09	February 2004
B9901-90018	11.0 11i v1	A.03.05.08	October 2003
B9901-90016	11.0 11i v1	A.03.05.08	September 2003
B9901-90014	11.0 11i v1	A.03.05.07	June 2003
B9901-90009	11.0 11i v1 11i v1.6	A.03.05.05	September 2002

What is in This Document

HP-UX IPFilter Version A.03.05.14 Administrator's Guide is divided into several chapters, and each contains information about installing, configuring, or troubleshooting HP-UX IPFilter. The appendices contain supplemental information.

Chapter 1 **Installing and Configuring HP-UX IPFilter** Use this chapter to install and configure HP-UX IPFilter software.

Chapter 2	**Rules and Keywords** Use this chapter to utilize the HP-UX IPFilter configuration files and obtain in-depth information on IPFilter and NAT keywords.
Chapter 3	**Dynamic Connection Allocation** Use this chapter to learn about DCA features, DCA keywords, DCA variables, changing DCA rules dynamically, and setting the DCA mode.
Chapter 4	**Firewall Building Concepts** Use this chapter to learn about specific configuration procedures for HP-UX IPFilter, as well as basic and advanced firewall design using HP-UX IPFilter features.
Chapter 5	**HP-UX IPFilter Utilities** Use this chapter to learn to use IPFilter utilities. This chapter also contains a list of unsupported utilities and commands.
Chapter 6	**HP-UX and IPv6 Support** Use this chapter to learn about using IPv6 support in HP-UX IPFilter.
Chapter 7	**HP-UX IPFilter and FTP** Use this chapter to learn about running an FTP server and an FTP client with HP-UX IPFilter.
Chapter 8	**HP-UX IPFilter and RPC** Use this chapter to learn about running HP-UX IPFilter with RPC.
Chapter 9	**HP-UX IPFilter and IPSec** Use this chapter to learn how IPFilter and IPSec work together.
Chapter 10	**HP-UX IPFilter and Serviceguard** Use this chapter to learn about the configuration procedures for HP-UX IPFilter in a Serviceguard environment.
Appendix A	**HP-UX IPFilter Configuration Examples** Use this appendix to reference configuration examples.
Appendix B	**HP-UX IPFilter Static Linking** Use this appendix to learn about statically linking the HP-UX IPFilter modules to the kernel.
Appendix C	**Performance Guidelines** Use this appendix to learn about general performance guidelines for using HP-UX IPFilter on a system.

Typographical Conventions

This document uses the following conventions.

audit (5)	An HP-UX manpage. In this example, *audit* is the name and *5* is the section in the *HP-UX Reference*. On the Web and on the Instant Information CD, it might be a hot link to the manpage itself. From the HP-UX command line, enter `man audit` or `man 5 audit` to view the manpage. See *man* (1).
Book Title	The title of a book. On the Web and on the Instant Information CD, it might be a hot link to the book itself.
KeyCap	The name of a keyboard key. Note that **Return** and **Enter** both refer to the same key.
Emphasis	Text that is emphasized.
Bold	Text that is strongly emphasized.
Bold	The defined use of an important word or phrase.
`ComputerOut`	Text displayed by the computer.
`UserInput`	Commands and other text that you enter.
`Command`	A command name or qualified command phrase.
`variable`	The name of a variable that you might replace in a command or function, or information in a display that represents several possible values.
`[ ]`	The contents are optional in formats and command descriptions. If the contents are a list separated by `|`, you must select one of the items.
`{ }`	The contents are required in formats and command descriptions. If the contents are a list separated by `|`, you must select one of the items.
...	The preceding element might be repeated an arbitrary number of times.
|	Separates items in a list of choices.

HP-UX Release Name and Release Identifier

Each HP-UX 11i release has an associated release name and release identifier. The *uname* (1) command with the `-r` option returns the release identifier. This table shows the releases available for HP-UX 11i.

Table 2 HP-UX 11i Releases

Release Identifier	Release Name	Supported Processor Architecture
B.11.11	HP-UX 11i v1	PA-RISC
B.11.22	HP-UX 11i v1.6	Intel® Itanium®
B.11.23	HP-UX 11i v2	Intel® Itanium®

Related Documents

Additional information about HP-UX IPFilter can be found on http://docs.hp.com in the *Internet and Security Solutions* collection under *HP-UX IPFilter* at:

http://docs.hp.com/en/internet.html#IPFilter

Other documents in this collection include:

- *HP-UX IPFilter A.03.05.14 Release Notes*
- *IPFilter/9000 Sizing and Performance White Paper*

For information about HP-UX Bastille, see the “HP-UX Bastille” section of *Managing Systems and Workgroups: A Guide for HP-UX System Administrators*. This guide is available at:

http://docs.hp.com/en/oshpux11iv2.html

HP Encourages Your Comments

HP encourages your comments concerning this document. We are truly committed to providing documentation that meets your needs.

Please send comments to netinfo_feedback@cup.hp.com.

Please include document title, manufacturing part number, and any comment, error found, or suggestion for improvement you have concerning this document. Also, please include what we did right so we can incorporate it into other documents.

1 Installing and Configuring HP-UX IPFilter

This chapter describes the procedures to install and configure HP-UX IPFilter software on your system. It contains the following sections:

- Overview of HP-UX IPFilter Installation
- Step 1: Checking HP-UX IPFilter Installation Prerequisites
- Step 2: Loading HP-UX IPFilter Software
- Step 3: Determining the Rules for IPFilter
- Step 4: Adding Rules to the Rules Files
- Step 5: Loading IPFilter and NAT Rules
- Step 6: Verifying the Installation and Configuration
- Supported and Unsupported Interfaces
- Troubleshooting HP-UX IPFilter

Overview of HP-UX IPFilter Installation

The following section summarizes each step in the HP-UX IPFilter installation process.

Installation and Configuration Checklist

The following checklist provides the sequence of steps you need to complete installation and configuration of HP-UX IPFilter. References to more in-depth information in this manual are also included as part of each step.

Step 1. Check that your system meets the prerequisites. See “Step 1: Checking HP-UX IPFilter Installation Prerequisites” on page 4 for detailed information about this task.

Step 2. Install HP-UX IPFilter using `swinstall`. See “Step 2: Loading HP-UX IPFilter Software” on page 5 for detailed information about this task.

Step 3. Decide what rules you must configure to protect your system. Chapter 2 contains the rules for basic firewalls, Chapter 4 contains the rules for advanced firewalls and Chapter 3contains the rules for Dynamic Connection Allocation (DCA). Appendix A contains examples of rulesets for specific situations. You should base your rules on the services running on your system.

Step 4. Add the filtering rules for your system to the `/etc/opt/ipf/ipf.conf` file and add Network Address Translation (NAT) rules to the `/etc/opt/ipf/ipnat.conf` file. See “Step 4: Adding Rules to the Rules Files” on page 8 for details.

Step 5. Load the rules into the HP-UX IPFilter rules file. See “Step 5: Loading IPFilter and NAT Rules” on page 10 for details.

Step 6. Run the `ipf`, `ipfstat`, and `ipnat` commands to verify the installation as described in “Step 6: Verifying the Installation and Configuration” on page 13.

See the *ipf* (5) and *ipfstat* (8) manpages for more detailed information on these commands.

Step 1: Checking HP-UX IPFilter Installation Prerequisites

1. Be sure your system uses one of the following operating systems:
   - HP-UX 11i v1
   - HP-UX 11i v2

   To obtain information about the OS, execute the command:

   ```
   uname -a
   ```
2. Install all required patches.

IMPORTANT

Check the latest HP-UX IPFilter Release Notes for all other patch information.

To obtain information about a patch, execute the command:

```
swlist -l patch <patch_id>
```

3. Be sure you have super user access and are designated the network security administrator.

Step 2: Loading HP-UX IPFilter Software

Use the following steps to load HP-UX IPFilter software using the HP-UX `swinstall` program.

NOTE If the product is downloaded to the system using

`swinstall -s | <path to product depot>`

follow step 1, then steps 5 through 12.

1. Log in as **`root`**.
2. Insert the software media (disk) into the appropriate drive.
3. Run the `swinstall` program using the command:

   `swinstall`

   The `Software Selection` window and `Specify Source` window open.
4. Change the `Source Host Name`, if necessary, enter the mount point of the drive in the `Source Depot Path` field, and click **OK** to return to the `Software Selection` window. Click **Help** for more information.

   The `Software Selection` window now contains a list of available software bundles to install.
5. Highlight the HP-UX IPFilter software for your system type.
6. Select `Mark for Install` from the `Actions` menu to select the product to be installed. With an exception of the manpages and user manual, you must install the complete IPFilter product.
7. Select `Install` from the `Actions` menu to begin the product installation and open the `Install Analysis` window.
8. Click **OK** in the `Install Analysis` window when the `Status` field displays a Ready message.
9. Click **Yes** on the `Confirmation` window to confirm that you want to install the software. The `Install` window opens.

View the `Install` window to read processing data while the software is being installed. The `Status` field indicates `Ready` and the `Note` window opens. The fileset is loaded by `swinstall`.

The estimated time for processing is three to five minutes.

10. Click **OK** on the `Note` window to reboot the system.

    The user interface disappears and the system reboots.

11. After the system reboots, check the log files in `/var/adm/sw/swinstall.log` and `/var/adm/sw/swagent.log` to be sure the installation was successful.

NOTE Do not run the HP-UX IPFilter product when the system is booted in single-user mode.

NOTE The IPFilter modules are dynamically linked into the kernel by default. To statically link the modules, see Appendix B, “HP-UX IPFilter Static Linking,” on page 177 for instructions and information.

12. Go to Step 3: Determining the Rules for IPFilter.

Step 3: Determining the Rules for IPFilter

Review the IPFilter rule descriptions and examples in Chapter 2, Chapter 4, and Appendix A to determine the appropriate rules for your system.

Determine the rules you will configure based on the services running on your system. Determine DCA rules as well. For more information on DCA, see Chapter 3.

If you are using NAT, determine the NAT rules you will configure as well.

Go to Step 4: Adding Rules to the Rules Files.

Step 4: Adding Rules to the Rules Files

To add your rules to the `/etc/opt/ipf/ipf.conf` file (or your chosen rules file) and to the `/etc/opt/ipf/ipnat.conf` file, use a text editor such as `vi`.

NOTE DCA rules are added along with IPFilter rules in the `/etc/opt/ipf/ipf.conf` file or your selected rules file. DCA rules can be used with or without IPFilter rules. If using the DCA feature, DCA mode must be turned on. For more information, see “DCA Mode” on page 61.

Adding IPFilter Rules

When IPFilter is installed, the default rules file, `ipf.conf`, is empty. You must add rules to this file to create a firewall. Alternately, you can change the configuration to read different rules files you specify.

Filtering rules added to `/etc/opt/ipf/ipf.conf` are loaded when the system is booted. If you do not want the rules to load on bootup, place your rules in an alternate location, such as `/etc/ipf.conf`. You can then load these rules manually using the `ipf` command.

See the example rulesets in Appendix A, “HP-UX IPFilter Configuration Examples,” on page 147 for assistance in putting your ruleset together. You can find additional information on the `ipf` command in “The ipf Utility” on page 83.

Adding NAT Rules

When IPFilter is installed, the default NAT rules file `ipnat.conf` is empty. You must add rules to this file to enable NAT. Alternately, you can change the configuration to read different NAT rules files you specify by changing the default configuration file name or location in the `/etc/rc.config.d/ipfconf` file.

NOTE IPFilter NAT functionality and the associated commands and utilities are not suppored with IPv6.

Filtering rules added to `/etc/opt/ipf/ipnat.conf` are loaded when the system is booted. If you do not want the rules to load on bootup, place your rules in an alternate location, such as `/etc/ipnat.conf`. You can then load these rules manually using the `ipnat` command.

To enable the NAT functionality, you must configure at least one filter rule using the `ipf` command. If you do not want to use the filtering functionality, you should configure a redundant rule. For example:

```
pass in all
```

See the example NAT ruleset in Appendix A, “HP-UX IPFilter Configuration Examples,” on page 147 for assistance in putting your NAT rules together. You can find additional information on NAT functionality in “map and portmap: Basic NAT” on page 39 and “The ipnat Utility” on page 101.

Go to Step 5: Loading IPFilter and NAT Rules.

Step 5: Loading IPFilter and NAT Rules

This section describes how to install rules in the HP-UX IPFilter and NAT rules file to run on your system.

Loading IPFilter Rules

NOTE

The following is a list of commands and file names, some of which are very similar:

- `ipfboot`—The startup script for the `ipf` module.
- `/etc/rc.config.d/ipfconf`—The configuration file for the `ipfboot` startup script.
- `/etc/opt/ipf/ipf.conf`—The default IPFilter rules file. Any rules present in this file are automatically loaded at bootup by the `ipfboot` startup script.

The configuration file, `ipfconf`, contains information that determines how HP-UX IPFilter starts when the system is rebooted. When HP-UX IPFilter is installed, the `ipfconf` file is put in the `/etc/rc.config.d` directory. The HP-UX IPFilter `ipfboot` startup script reads `ipfconf`.

By default, HP-UX IPFilter starts on bootup and the rules from the `/etc/opt/ipf/ipf.conf` file are processed. If you do not want the rules to load on bootup, place your rules in an alternate location and then manually load the rules using the `ipf` command.

- Add new rules to your rules file using the `-f` option of the `ipf` command:

  `ipf -f <rules file>`

NOTE

When a rule has been loaded, it takes effect immediately. For example, if you add a rule to block all traffic and load it using `ipf -f <rules file>`, you will be blocked from X-Windows and networking processes.

- Flush rules from your ruleset using the `-Fa` option of the `ipf` command:

  ```
  ipf -Fa
  ```

  The `-Fa` option flushes previously configured rules. The `-A` option specifies the active rules list. For example:

  ```
  ipf -Fa -A -f /etc/opt/ipf/ipf.conf
  ```

  The previous command flushes the previously configured rules, specifies `/etc/opt/ipf/ipf.conf` as the active rules file, and loads the rules in `/etc/opt/ipf/ipf.conf` for immediate use.

  Optionally, use the `-I` option if you do not want to save previously configured rules. This command adds rules to the inactive rule list. For example:

  ```
  ipf -I -Fa -A -f /etc/opt/ipf/ipf.conf
  ```

  This command enables the new rules. The `-I` option swaps the active rules you just configured with the inactive rules. To make the old rules effective again, use `ipf -s` to swap the rulesets.

  The `-Fi` command flushes only the IN rules in the specified rules file. For example:

  ```
  ipf -Fi /etc/opt/ipf/ipf.conf
  ```

  The `-Fo` command flushes only the IN rules in the specified rules file. For example:

  ```
  ipf -Fo /etc/opt/ipf/ipf.conf
  ```

Removing IPFilter Rules

If necessary, the following command can be used to remove different rules files:

```
ipf -r -f <delete_rule_file>
```

This command can be executed while IPFilter is running.

Loading NAT Rules

To load IPFilter NAT rules:

1. Add NAT rules to the `ipnat.conf` file, or to another NAT rules file you select. See “map and portmap: Basic NAT” on page 39 and “The ipnat Utility” on page 101 for information and instructions.
2. Use the following command to load the NAT rules manually:

   ```
   ipnat -CF -f /etc/opt/ipf/ipnat.conf
   ```

   This command flushes any current mappings and NAT rules, and reads NAT rules from the specified rules file.

Go to Step 6: Verifying the Installation and Configuration.

Step 6: Verifying the Installation and Configuration

After HP-UX IPFilter is installed and you have configured and loaded the rules file, you must verify the installation and configuration.

- Verify that HP-UX IPFilter is running using the `-v` option of the `ipf` command:

  ```
  ipf -V
  ipf: HP IP Filter: v3.5alpha5 (A.03.05.07) (312)
  Kernel: HP IP Filter: v3.5alpha5 (A.03.05.07)
  Running: yes
  Log Flags: 0 = none set
  Default: pass all, Logging: available
  Active list: 1
  ```

- Verify that HP-UX IPFilter has been correctly loaded using the `kmadmin -s` command:

  ```
  kmadmin -s
  Name         ID         Status        Type
  ============================================
  pfil         1          LOADED        STREAMS
  ipf          2          LOADED        WSIO
  ```

- Execute the following commands to verify that your rules have been properly loaded. Run `ipfstat -i` to check for the inbound rules and run `ipfstat -o` to check for outbound rules.

  To view all rules at the same time, run:

  ```
  ipfstat -io
  ```

  By default, IPFilter processes the rules in the `ipf.conf` file.

Additional Configuration Information

IPFilter provides additional configuration options, such as the following `ndd` variables.

Name	Description	Default Value
`ipl_buffer_sz`	Size of the IPFilter logging buffer for `/dev/ipl`.	8K
`ipl_suppress`	If set, does not print identical log records separately, but counts them as *Nx*, where *N* is the number of times the log record occurs.	1
`ipl_logall`	If set, the entire packet is logged. Otherwise, only the first 128 bytes are logged. This should be used with the `log body` rules.	0
`cur_iplbuf_sz`	Tells the size of the log buffer and the amount of buffer space being used.	None

NOTE The previous `ndd` variables cannot be set through `/etc/rc.config.d/nddconf`. So, when IPFilter starts up, these `ndd` variables will have their default values.

Supported and Unsupported Interfaces

The following table lists the interfaces supported for each version of HP-UX IPFilter.

CAUTION For all versions of HP-UX IPFilter, the unsupported interfaces do not interact with IPFilter. IPFilter does not block or protect the system from traffic on unsupported interfaces.

HP-UX IPFilter is not tested with any third party products.

Table 1-1 HP-UX IPFilter Supported Interfaces

HP-UX IPFilter Version	Supported Interfaces
A.03.05.14 A.03.05.12 A.03.05.11.01 A.03.05.10 A.03.05.10.02 A.03.05.06.v2	• Ethernet (10Base-T) • Fast Ethernet (100Base-T) • Gigabit Ethernet (1000Base-T) • APA • VLAN • FDDI • Token Ring • InfiniBand (supported on HP-UX 11i v2 only)
A.03.05.10.04	• Ethernet (10Base-T) • Fast Ethernet (100Base-T) • Gigabit Ethernet (1000Base-T) • APA • VLAN • FDDI • Token Ring

Table 1-1 HP-UX IPFilter Supported Interfaces (Continued)

HP-UX IPFilter Version	Supported Interfaces
A.03.05.09 A.03.05.08 A.03.05.07 A.03.05.06	• Ethernet (10Base-T) • Fast Ethernet (100Base-T) • Gigabit Ethernet (1000Base-T) • APA • VLAN • FDDI • Token Ring

The following interfaces are unsupported (not protected by HP-UX IPFilter) on any HP-UX IPFilter releases:

- ATM
- Hyperfabric
- X.25
- Frame Relay
- PPP

Troubleshooting HP-UX IPFilter

This section describes how to troubleshoot an HP-UX IPFilter configuration. It provides information about possible problems that might occur along with the steps needed to resolve them.

- **HP-UX IPFilter is not filtering (it passes/allows all network traffic).**

  Verify whether HP-UX IPFilter is running using `ipf -V`. The running field should say `yes`. If it says `no`, then the HP-UX IPFilter module has not been loaded. It might have been explicitly unloaded.

  To load IPFilter again, use:

  ```
  /sbin/init.d/ipfboot start
  ```

  To determine if the HP-UX IPFilter DLKM modules are loaded, execute either the `kmadmin` (1M) command on HP-UX 11i v1 or the `kcmodule` (1M) command on HP-UX 11i v2. See the respective manpages for more information.

  Load the rules and check again that IPFilter works. If it still does not work, reboot the system and check `/etc/rc.log` and `/var/adm/syslog/syslog.log` for errors.

- **The host does not seem to be on the network and `pings` do not go through.**

  Check the rules you have configured using `ipfstat -io`. This command will show the in and the out rules.

NOTE

If you are using `/etc/opt/ipf/ipf.conf` as your rules file, then it will be loaded at boot time. The IPFilter startup script `/sbin/init.d/ipfboot` will:

— Load the IPFilter module.

— Start the logging daemon, `ipmon`.

— Load any uncommented rules present in `/etc/opt/ipf/ipf.conf`.

If the last effective rule amounts to "block in all," the boot sequence might not complete, for example, when sendmail, SNMP, and NIS are configured on the system.

- **Nothing is logged.**

  Verify the following:

  `ipf -V` should show the logging file as available.

  `ps -ef|grep ipmon` to verify if `ipmon` is running. During bootup, `ipmon` is started. If it is not running, start it by using:

  ```
  ipmon -sD
  ```

  The `-s` option specifies that the log records go to `/var/adm/syslog/syslog.log` and the `-D` option directs `ipmon` to run as a daemon in the background.

- **Errors occur when loading rules.**

  ```
  # ipf -f <rulefile>
  ioctl (add/insert rule); File Exists
  ```

  This occurs when you try to add a rule that is already loaded. Use the following command to load rules:

  `ipf -Fa -f <rulefile>`

  The `-Fa` option will flush any previous rules present and all rules will be reloaded.

  In addition, you can use `ipftest` to test a set of filter rules without having to put them in place. See the *ipftest* (1) manpage for more information on this tool.

- **IPFilter rules changed after using Bastille/ Install-Time-Security level.**

  If you configure an IPFilter ruleset-using Install-Time-Security level, or use HP-UX Bastille interactively to reconfigure IPFilter rules, existing rules will be overwritten. This will change IPFilter behavior.

  To reinsert your rules into the Bastille-setup firewall rules, edit `/etc/opt/sec_mgmt/bastille/ipf.customrules`, and run `bastille -b -f <config file>`. Alternatively, to remove all of the security hardening performed by Bastille, including the firewall configuration, run `bastille -r`. For more information, see the Bastille documentation.

2 Rules and Keywords

This chapter describes the basic procedures and building blocks used to create filtering rules for HP-UX IPFilter.

It contains the following sections:

- IPFilter Configuration Files
- Basic Rules Processing
- IPFilter Keywords
  - pass and block: Controlling IP Traffic
  - in and out: Bidirectional Filtering
  - quick: Optimizing IPFilter Rules Processing
  - on: Filtering by Network Interfaces
  - from and to: Filtering by IP Addresses and Subnets
  - log: Tracking Packets on a System
  - proto: Controlling Specific Protocols
  - opt and ipopts: Filtering on IP Options
  - icmp-type: Filtering ICMP Traffic by Type
  - port: Filtering on TCP and UDP Ports
  - keep state: Protecting TCP, UDP, and ICMP Sessions
  - flags: Tight Filtering Based on TCP Header Flags
  - keep frags: Letting Fragmented Packets Pass
  - with frags: Dropping Fragmented Packets
  - with short: Dropping Short Fragments
  - return-rst: Responding to Blocked TCP Packets
  - return-icmp: Responding to Blocked ICMP Packets
  - dup-to: Drop-Safe Logging
- NAT Keywords
  - map and portmap: Basic NAT
  - bimap: Bidirectional Mapping
  - rdr: Redirecting Packets
  - map-block: Mapping to a Block of Addresses

NOTE Most of the information in this chapter has been derived from the IPFilter-based Firewalls HOWTO document written by Brendan Conoby and Erik Fichtner. You can find this document at http://www.obfuscation.org/ipf/.

IPFilter Configuration Files

HP-UX IPFilter has two files it uses for configuration.

IPFilter Rules

The HP-UX IPFilter rules file is named `/etc/opt/ipf/ipf.conf`.

The UNIX configuration file conventions allow one rule per line. The number symbol (#) denotes a comment at the beginning of a line as well as a rule and a comment on the same line. Extra white space is allowed and encouraged to keep the rules readable.

By default, HP-UX IPFilter starts on bootup and the rules from the `/etc/opt/ipf/ipf.conf` file are processed.

When HP-UX IPFilter is first installed, the rules file is empty. You must put rules into this file or change the configuration to read another file that holds IPFilter rules. You can change the file information by editing the rules file using `vi` or another text editor.

IPFilter Configuration File

When HP-UX IPFilter is installed, the `ipfconf` file is put in the `/etc/rc.config.d` directory. The information in this file determines how HP-UX IPFilter starts when the system is booted and also gives the location of the rules file.

See Appendix A, “HP-UX IPFilter Configuration Examples,” on page 147 for example rules files to help you create your ruleset.

Basic Rules Processing

Rules are processed in order from top to bottom of the rules file. If the contents of your rules file are as follows, IPFilter processes the rules in the order they appear from top to bottom:

```
block in all
pass in all
```

IPFilter does not stop processing rules after a match is made. Instead, it acts on the **last** rule that matches a packet being checked. In the previous ruleset, all incoming packets match both rules, but all packets are passed according to the last rule matched, `pass in all`.

Unlike other packet filters, IPFilter keeps a flag on whether it passes a packet. Unless the flow is interrupted, IPFilter goes through the entire ruleset and passes or drops each packet based on the last matching rule.

Given the following ruleset:

```
block in all
block in all
block in all
block in all
pass in all
```

All packets pass through. There is no cumulative effect during processing. The last matching rule always takes precedence.

IPFilter Keywords

IPFilter rules are built using keywords and parameters that combine to filter packets coming in and out of a system. The following sections describe the keywords that form the basic building blocks of IPFilter rules. These sections include the purpose of the keywords and examples of how to use them in rules.

NOTE For more information about IPFilter rule syntax, see the *ipf* (5) manpage.

pass and block: Controlling IP Traffic

The first keyword in any IPFilter rule is usually either `pass` or `block`. To allow packets into the IPFilter system, use `pass`. For example, to allow all incoming packets, use:

```
pass in all
```

To deny all incoming packets, use:

```
block in all
```

in and out: Bidirectional Filtering

You can explicitly pass and block both inbound and outbound traffic. Inbound traffic is all traffic that enters the firewall on any interface. Outbound traffic is all traffic that leaves on any interface, whether locally generated or passing through. Packets coming in are not only filtered as they enter the firewall, they are also filtered as they exit.

To block all incoming packets, use the following rule:

```
block in all
```

To pass all outgoing packets, use the following rule:

```
pass out all
```

You can use in and out with all other keywords. IPFilter filtering techniques are applicable to both inbound and outbound traffic.

NOTE If you do not specify any out rules, the implied default is `pass out all`. If you do not specify any in rules, the implied default is `pass in all`.

quick: Optimizing IPFilter Rules Processing

HP-UX IPFilter behaves differently from other packet filters. Because it processes the whole ruleset for each packet, there might be a performance impact if your rules file is configured so that the most applicable rules are in the first 10 of 100 rules.

You can use the `quick` keyword to control rule processing and reduce performance impact on your IPFilter system. If IPFilter matches a packet to a rule that contains `quick`, IPFilter immediately acts on that rule without continuing to check the packet against the other rules in the ruleset. For example, if you configure the following ruleset:

```
block in quick all
pass in all
```

IPFilter matches all packets to the first rule, `block in quick all` and blocks the packet. Because `quick` is used, IPFilter does not consider the other rule in the ruleset.

on: Filtering by Network Interfaces

You can use the `on` keyword to control traffic to and from your system based on network interfaces.

Your system can have interfaces to more than one network. Every packet the system receives comes in on a network interface; every packet the system transmits goes out on a network interface.

For example, your machine has two interfaces, `lan0` and `lan1`, and you do not want packets coming in on the `lan0` interface. You add the following rules:

```
block in quick on lan0 all
pass in all
```

The `on` keyword means that network traffic is coming in on the named interface, `lan0`. If a packet comes in on `lan0`, the first rule blocks it. If a packet comes in on `lan1`, the first rule does not match. The second rule matches and the packet is passed.

from and to: Filtering by IP Addresses and Subnets

IPFilter can pass or block packets based on both source and destination IP addresses. It can also filter on subnets.

To configure IPFilter to pass or block packets based on their source IP address, use the `from` *`ip_address`* keyword. For example:

```
block in quick from 192.168.0.0 to any
```

For traffic coming from any address within a subnet, you can use `from` with the following subnet address syntax:

```
block in quick from 192.168.0.0/16 to any
```

For traffic coming from any address within a range of addresses, you can use `from` with the following address range syntax:

```
block in quick from 192.168.32.2-192.168.32.100 to any
```

To configure IPFilter to pass or block packets based on their destination IP address, use the `to` *`ip_address`* keyword. For example:

```
block in quick from any to 192.168.0.0
```

For packets originating in or destined for a subnet, you can use either `to` or `from` with any of the following subnet address syntaxes:

- A standard dot-notation address mask, for example:

  ```
  pass in from 192.168.1.1 to any
  ```

- A single hexadecimal number with a leading 0x, for example:

  ```
  pass in proto tcp from 0xc0a80101 to any
  ```

  or use the integer format, for example:

  ```
  pass in from 3232235777 to any
  ```

NOTE

`0xc0a80101` and `3233325777` are the hexadecimal and integer representations of `192.168.1.1`, respectively.

- A Classless Inter-Domain Routing (CIDR) notation, such as:

  ```
  pass in from 192.168.1.1/24 to any
  ```

You can combine specific `from` *`ip_address`* and `to` *`ip_address`* keywords to restrict traffic based on both source and destination IP addresses.

You can also filter traffic using both IP addresses and network interface names. For example, you want data from `lan0`, but not from `192.168.0.0/16`. Configure the following rules:

```
block in quick on lan0 from 192.168.0.0/16 to any
pass in all
```

With this ruleset, the on `lan0` keyword means that a packet is blocked only if it comes in on the `lan0` interface. If a packet comes in on the `lan0` interface from `192.168.0.0/16`, it passes.

log: Tracking Packets on a System

You can use the logging capability of IPFilter to track incoming and outgoing packets. Logging lets you determine if your IPFilter system is being attacked, and gives you some information about attacks.

While it is unnecessary to log every passed packet and, in some cases, every blocked packet, you can select to log specific blocked or passed packets. For example, if you want to log blocked packets from a specific address, such as `20.20.20.0/24`, use the following rule:

```
block in log quick on lan0 from 20.20.20.0/24 to any
```

You can use the log keyword with any IPFilter rule. HP recommends deciding which rules are the most important or the most likely to block attacks on your system and logging only those rules.

NOTE The log keyword can be used with several advanced options to control and enhance logging functionality and performance. See “Logging Techniques” on page 70 for more information.

proto: Controlling Specific Protocols

IPFilter can filter traffic based on protocol, such as TCP or ICMP, using the `proto` keyword.

For example, many Denial of Service (DoS) attacks rely on glitches in the TCP/IP stack of the OS, in the form of ICMP packets. To block ICMP packets, add the proto command to your ruleset as follows:

```
block in log quick on lan0 proto icmp from any to any
```

In this example, any ICMP traffic coming in from `lan0` will be logged and discarded.

IPFilter also has a shorthand for rules that apply to `proto tcp` and `proto udp` at the same time, such as portmap or NFS. The rule for portmap would be:

```
block in log quick on lan0 proto tcp/udp from any to
20.20.20.0/24 port = 111
```

opt and ipopts: Filtering on IP Options

IPFilter can filter packets based on IP options using the `opt` and `ipopts` keywords. You can configure IPFilter rules to pass or block packets that have a specific option set. For example:

```
block in quick all with opt lsrr, ssrr
```

NOTE If you configure IPFilter to filter on more than one option with the `opt` keyword, use a comma and a space to delimit each option. See the previous example for correct syntax.

You can also configure rules to pass or block packets that do not have a specific option set. For example:

```
pass in from any to any with opt ssrr not opt lsrr
```

If you want to block or pass a packet that has any IP option set or no IP options set, use the `ipopts` keyword. For example:

```
block in all with ipopts
```

For a complete list of IP options, see the IETF RFC at http://www.faqs.org/rfcs/std/std2.html.

icmp-type: Filtering ICMP Traffic by Type

You can filter specific types of ICMP traffic using the icmp-type keyword. This is a useful keyword if you want to block most ICMP traffic to prevent DoS attacks, but must allow certain types of ICMP messages to pass to your system.

For example if you want to specifically allow ping messages to pass on your system, configure the following rule:

```
pass in quick on lan0 proto icmp from any to 20.20.20.0/24
icmp-type 0
```

You must know the type number for any ICMP message you want to explicitly pass or block using the `icmp-type` keyword. The following is a list of ICMP message type numbers used by HP-UX IPFilter.

TYPE	CODE	icmp-type icmp-code	MEANING
0	0	echorep	ECHO REPLY (ping reply) [RFC792]
3		unreach	DESTINATION UNREACHABLE
	0	net-unr	network unreachable
	1	host-unr	host unreachable
	2	proto-unr	protocol unreachable
	3	port-unr	port unreachable [RFC792]
	4	needfrag	need fragmentation [RFC792]
	5	srcfail	source route failed [RFC792]
	6	net-unk	destination network unknown
	7	host-unk	destination host unknown
	8	isolate	source host isolated [RFC792] (ping)
	9	net-prohib	destination network administratively prohibited [RFC1256]

TYPE	CODE	icmp-type icmp-code	MEANING
	10	host-prohib	destination host administratively prohibited [RFC1256]
	11	net-tos	network unreachable for TOS [RFC792]
	12	host-tos	host unreachable for TOS [RFC792]
	13	filter-prohib	prohibited by filtering [RFC1812]
	14	host-preced	host precedence violation [RFC1812]
	15	cutoff-preced	precendence cutoff in effect [RFC1812]
4	0	squench	SOURCE QUENCH
5		redir	REDIRECT
		network	
		host	
		network & TOS	
		host & TOS	
8	0	echo	ECHO REQUEST (ping request)
	0	routerad	ROUTER ADVERTISEMENT
	0	routesol	ROUTER SOLICITATION
11		timex	TIME EXCEEDED
			TTL=0 during transmit
			TTL=0 during reassembly
12		paramprob	PARAMETER PROBLEM
13	0	timest	TIMESTAMP REQUEST

TYPE	CODE	icmp-type icmp-code	MEANING
14	0	timestrep	TIMESTAMP REPLY
15	0	inforeq	INFO REQUEST (obsolete)
16	0	inforep	INFO REPLY (obsolete)
17	0	maskreq	ADDRESS MASK REQUEST
18	0	maskrep	ADDRESS MASK REPLY

Rule order is important if you are using the icmp-type keyword with the quick keyword. Place pass rules before block rules in the ruleset to be sure the correct packets are passed.

port: Filtering on TCP and UDP Ports

In addition to filtering network traffic by protocol, you can use IPFilter to block traffic on specific ports used by a protocol. You can pass or block traffic on a specific port, such as a well-known port used by a service like `telnet` or `rlogin`.

For example, you can block incoming `telnet` traffic with the following rule:

```
block in log quick on lan0 proto tcp from any to 20.20.20.0/24
 port = 23
```

You can also pass or block traffic on a range of ports, such as the high port numbers used for client telnet connections. The following is a list of operands you can use with port numbers:

Operand	Alias	Result
<	lt	true if port is less than configured value
>	gt	true if port is greater than configured value
=	eq	true if port is equal to configured value
!=	ne	true if port is not equal to configured value

Operand	Alias	Result
<=	le	true if port is less than or equal to configured value
>=	ge	true if port is greater than or equal to configured value

keep state: Protecting TCP, UDP, and ICMP Sessions

Use `keep state` to identify and authorize individual TCP, UDP, and ICMP sessions that pass multiple packets back and forth. `keep state` enables IPFilter to distinguish legitimate traffic from port scanners and DoS attacks.

IPFilter maintains a state table. The state table is a list of open TCP, UDP, and ICMP sessions. If a packet matches an entry in the state table, it passes through the firewall without being checked against the ruleset. This enhances the performance of the IPFilter system.

IPFilter checks both inbound and outbound packets against the state table. If either an inbound or an outbound packet matches a session in the state table, it is not checked against the ruleset.

You can use `keep state` to limit the number of rules you must configure. Use `keep state` to pass or block the first packet in a TCP, UDP, or ICMP session. When a packet matches a `keep state` rule, and entry is added to the state table. You do not need to configure rules for all the other types of traffic that might pass within a specific session.

For example, you can use the `keep state` keyword with IPFilter rules to protect an SSH server.

```
block out quick on lan0 all
pass in quick on lan0 proto tcp from any to 20.20.20.1/32 port
 = 22 keep state
```

Using the `keep state` keyword, after the first SYN packet is received by the SSH server, an entry is made in the IPFilter state table. The remainder of the SSH session continues without any further packets within the session being checked against the IPFilter ruleset. Therefore, the outbound traffic can flow freely within the session, despite the block-out rule specified.

The following rules also work for UDP and ICMP:

```
block in quick on lan0 all
pass out quick on lan0 proto tcp from 20.20.20.1/32 to any keep
 state
pass out quick on lan0 proto udp from 20.20.20.1/32 to any keep
 state
pass out quick on lan0 proto icmp from 20.20.20.1/32 to any
 keep state
```

flags: Tight Filtering Based on TCP Header Flags

You can use IPFilter to filter traffic by port number; you can additionally filter traffic to or from a specific port based on the flags set in the TCP header of the IP packet. Use the `flags <option>` keyword to filter traffic by flags.

For example, to allow only packets with the SYN flag set through on port 23, configure the following rules:

```
pass in quick on lan0 proto tcp from any to 20.20.20.1/32 port
 = 23 flags S keep state
pass out quick on lan0 proto tcp from any to any flags S keep
 state
block in quick all
block out quick all
```

Now only TCP packets destined for 20.20.20.1 at port 23 with a SYN flag pass in and are entered into the state table. A lone SYN flag is only present as the very first packet in a TCP session (called the TCP handshake). These rules have at least two advantages:

- No arbitrary packets can come in and negatively impact the state table.
- FIN and XMAS scans will fail; they set flags other than the SYN flag.

`Flags S` equates to flags S/AUPRFS and matches against only the SYN packet, out of all six possible flags.

`Flags S/SA` allows packets that might or might not have the URG, PSH, FIN, or RST flags set. Some protocols demand the URG or PSH flags. S/SAFR would be a better choice for these protocols.

It is more secure to use `flags S` when `flags S/SA` is not required.

NOTE To use the `flags <option>` keyword, you must know the correct designations for the flags you want to use in your rules. See RFC793, Transmission Control Protocol Specifications for a list of TCP flags.

keep frags: Letting Fragmented Packets Pass

You can configure IPFilter to track fragmented packets and to pass expected packet fragments. The `keep frags` keyword lets you configure IPFilter to pass fragmented packets while blocking packets that might be forgeries or port scans trying to attack the system.

In the following example, four rules are configured to log forgeries and allow fragments:

```
pass  in  quick on lan0 proto tcp from any to 20.20.20.1/32
 port = 23 flags S keep state keep frags
pass  out quick on lan0 proto tcp from any to any keep state
 flags S keep frags
block in log quick all
block out log quick all
```

In this example, every valid packet is entered into the state table before the blocking rules are processed. To further protect the system, log initial SYN packets to detect SYN scans.

with frags: Dropping Fragmented Packets

If you do not want packet fragments to pass through the firewall, use the `with frags` keyword. The `with frags` keyword drops all packet fragments. For example:

```
block in all with frags
```

with short: Dropping Short Fragments

You can configure IPFilter to drop packet fragments that are too short for comparison using the `with short` keyword. This is useful for security purposes, as an attacker can use fragments to attempt to access the system. For example:

```
block in proto tcp all with short
```

return-rst: Responding to Blocked TCP Packets

When you use the block keyword as described in "pass and block: Controlling IP Traffic" on page 26, the blocked packet is dropped and no response is sent to the remote system the packet. This can be a security risk, because it might alert an attacker that a packet filter is running on the system.

When a service is not running on a UNIX system, it normally notifies the remote host with a return packet. In TCP, this is done with a Reset (RST) packet. To configure IPFilter to return an RST packet to the origin, use the `return-rst` keyword. For example:

```
block return-rst in quick on lan0 proto tcp from HostA to any
port = 23
pass out quick on lan0 proto tcp from any port = 23 to any
flags R/RSFUP
```

The first rule blocks the telnet connection from HostA and generates a TCP RST packet. The second rule is necessary to let out the packet.

This example has two block statements since `return-rst` only works with TCP; it still blocks UDP and ICMP protocols. When this is done, the remote side receives a `Connection Refused` message instead of a `Connection Timed Out` message.

return-icmp: Responding to Blocked ICMP Packets

You can configure IPFilter to send an error message when a packet is sent to a UDP port on your system. For example:

```
block return-icmp(port-unr) in log quick on lan0 proto udp from
 any to 20.20.20.0/24 port
```

The port-unreachable (`port-unr`) message is the default for a `return-icmp` message. HP recommends that you use this message when configuring most `return-icmp` rules.

When rules with `return-icmp` are configured, IPFilter returns the ICMP packet with the IP address of the firewall, not the original destination of the packet. Use the `return-icmp-as-dest` keyword to return the original destination of the ICMP packet. The format is:

```
block return-icmp-as-dest(port-unr) in log on lan0 proto udp
 from any to 20.20.20.0/24 port = 111
```

dup-to: Drop-Safe Logging

IPFilter can pass packets on to another system for additional logging, examination, and processing.

Instead of configuring IPFilter rules to drop packets, you can configure rules to pass them to another system that can perform more extensive logging and analysis than `ipmon` does. A firewall system can have multiple interfaces. You can create a "drop-safe" for packets using the `dup-to` keyword.

For example, to configure IPFilter to send a copy of every packet going out the `lan0` interface to your drop-safe network on `ed0`, include this rule in your filter list:

```
pass out on lan0 dup-to ed0 from any to any
```

You can also send a packet directly to a specific IP address on your drop-safe network. For example:

```
pass out on lan0 dup-to ed0:192.168.254.2 from any to any
```

This method alters the destination address of the copied packet, which can negatively impact the usefulness of the IPFilter log. For this reason, HP recommends only using the known address method of logging to be certain that the logged address corresponds in some way to the system for which IPFilter is logging.

In general, `dup-to ed0` is all that is required to get a new copy of the packet over to the drop-safe network for logging and examination.

You could also use this feature to implement an intrusion detection network by hiding the presence of the intrusion detection system from the real network so that it cannot be detected from the outside.

In addition, there are some operational characteristics that should be noted. If you are only dealing with blocked packets, you can use the `to` keyword as described in "Using the to Keyword to Capture Blocked Packets" on page 75. If the system is configured to pass packets, you should configure rules to make a copy of the packet for the drop-safe log using the `dup-to` keyword.

NAT Keywords

The following section describes keywords specific to NAT functionality.

NOTE The maximum number of concurrent connections NAT can support is 16,383.

map and portmap: Basic NAT

Use the `map` keyword to create basic IPFilter NAT rules.

If you do not know the IP address of the target systems, configure the following rule:

```
map lan0 192.168.1.0/24 -> 0/32
```

IPFilter NAT automatically detects the IP address of the outgoing interface and translates 0/32 to the IP address of that interface. Outgoing traffic addresses are translated to the outgoing interface IP address.

If you do know the IP address of the outgoing interface, configure the following rule:

```
map lan0 192.168.1.0/24 -> 20.20.20.1/32
```

IPFilter NAT translates the source IP addresses of the outgoing packets to 20.20.20.1.

You can use the `portmap` keyword to force a translated IP packet onto a specific port on the target system. This is useful if there is another firewall the packet must pass through or if many systems are trying to use the same source port.

To force TCP and UDP packets onto a specific port range, configure the following rule:

```
map lan0 192.168.1.0/24 -> 0/32 portmap tcp/udp 20000:30000
```

All translated TCP and UDP packets are forced through ports 20000 through 30000.

bimap: Bidirectional Mapping

The `bimap` keyword allows IPFilter to map IP addresses bidirectionally. This can be used when you want the IP address of a particular device on the NAT-supported system to display as having a different IP address outside the system. The following rule demonstrates the `bimap` property:

```
bimap lan0 192.168.1.1/32 -> 20.20.20.1/32
```

In the previous example, devices with IP address 192.168.1.1 on the NAT-supported system display as having an IP address of 20.20.20.1 outside the system.

rdr: Redirecting Packets

The `rdr` keyword redirects packets coming into an IPFilter NAT system. The default protocol the `rdr` keyword uses is TCP.

You can use the `rdr` keyword to redirect packets from one port to another. For example, you can redirect traffic destined for the well-known port 80 to port 8000 to enhance security on your system. Configure the following rule:

```
rdr lan0 20.20.20.5/32 port 80 -> 192.168.0.5 port 8000
```

You can redirect UDP and ICMP packets as well as TCP packets. To redirect UDP packets, add `udp` to the rule you configure. For example:

```
rdr lan0 20.20.20.0/24 port 31337 -> 127.0.0.1 port 31337 udp
```

You can use NAT redirection and IPFilter filtering together to provide secure, redirected connections. For example, configure the following NAT rule:

```
rdr lan0 20.20.20.5/32 port 80 -> 192.168.0.5 port 8000
```

Then configure the following IPFilter rule:

```
pass in on lan0 proto tcp from 172.16.8.2 to 192.168.0.5/32
port = 8000 flags S keep state
```

When a packet comes in, the NAT rule is processed first. The destination address and port number are rewritten. Then the packet is passed to the IPFilter rules for processing and the packet is matched to the pass in rule.

You can use the `rdr` keyword to implement load-balancing systems and redirect traffic to multiple destination addresses. For example:

```
rdr lan0 20.20.20.5/32 port 80 -> 192.168.0.5,192.168.0.6 port
8000
```

map-block: Mapping to a Block of Addresses

IPFilter NAT can map an IP address to a specific block of IP addresses in two ways.

You can use the `map-block` keyword to statically map sessions from a host to a selected block of IP addresses. Configure the following rule:

```
map-block lan0 192.168.1.0/24 -> 20.20.20.0/24
```

Any outgoing packet with an IP address beginning with 192.168.1 is mapped to an IP address beginning with 20.20.20.

Alternately, you can configure IPFilter NAT to translate to a block of IP addresses using only the `map` and `portmap` keywords. Configure the following rule:

```
map lan0 192.168.0.0/16 -> 20.20.20.0/24 portmap tcp/udp
20000:60000
```

3 Dynamic Connection Allocation

This chapter describes Dynamic Connection Allocation (DCA). It includes DCA keywords, rule syntax and conditions, and variables. It also contains procedures for changing DCA rules dynamically and setting DCA mode at startup.

This chapter contains the following sections:

- DCA with HP-UX IPFilter
  - Overview: DCA Functionality
  - Using DCA
- DCA Keywords
  - keep limit: Limiting Connections
  - log limit: Logging Exceeded Connections
  - log limit freq: Log Frequency
- DCA Rule Syntax
- DCA Rule Conditions
- keep limit Rules and Rule Hits
- DCA Rule Modifications
  - Updating keep limit Rules
  - Adding New keep limit Rules
  - Integrating keep limit Rules
  - Extracting an Individual Rule from a Subnet Rule
- DCA Variables
  - fr_statemax
  - fr_tcpidletimeout
  - Configuring Variables
- DCA Mode

DCA with HP-UX IPFilter

An HP-UX IPFilter system can act as a secure intermediary, tracking all incoming TCP connections to a system or network. DCA lets you limit incoming TCP connections passing through an IPFilter system. DCA uses stateful packet inspection to limit the number of incoming TCP connections to a system.

To use DCA functionality, be sure DCA mode is enabled. For more information, see “DCA Mode” on page 61. DCA functionality does not work if DCA mode is not enabled.

Overview: DCA Functionality

DCA provides a set of flexible rules for controlling incoming TCP connections. You allocate a number of TCP connections to a system using a **limit value**. The limit value is the number of concurrent TCP connections that can be established by any given source.

You can configure DCA rules to limit the number of connections from:

- A specific IP address.
- Each IP address in an IP subnet or IP address range.
- An IP subnet or IP address range where all the IP addresses in the subnet share the cumulative limit.
- Unknown IP addresses, where each unknown IP address has a connection limit.

When the configured limit is reached, any additional connections to the HP-UX IPFilter system are dropped. You can configure HP-UX IPFilter to send a TCP reset when it drops these connection. See “return-rst: Responding to Blocked TCP Packets” on page 37 for more information.

A set of commands helps collect data about the connections that are being controlled. This data includes the source and destination IP address, allocated number of connections, number of active connections, and number of times the allocated quota of connections was exceeded. These new commands can be found in:

- “The ipf Utility” on page 83.

  — `ipf -Q <interface name>`

  - ipf -E <*interface name*>
  - ipf -D <*interface name*>
  - ipf -m <*option*>

- "The ipfstat Utility" on page 86.
  - ipfstat -L
  - ipfstat -vL
  - ipfstat -r <*group:rule*>
- "The ipmon Utility" on page 93.
  - ipmon -r

DCA also provides logging records that can serve as alert messages or as a summary of the connections made from a specific IP address. You can fine-tune the rules configured by identifying IP addresses or subnets that could be subjected to more conservative connection allocation or blocked altogether.

Using DCA

DCA helps protect systems from floods of TCP connections created by DoS attacks. You can use DCA to:

- Protect a mail server from a flood of SMTP connections. IP addresses or subnets that are trying to flood the SMTP server can be slowed down. At the same time, known users can be given unlimited connection limits. This ensures that customers and partners can still access the mail server while attackers are prevented from tying up resources.
- Protect an LDAP server from a flood of bogus SSL connections or any other types of connections trying to tie up the LDAP server.

DCA Keywords

The following section describes keywords specific to DCA. For additional information about DCA rule syntax and rule conditions, see “DCA Rule Syntax” on page 52 and “DCA Rule Conditions” on page 53.

keep limit: Limiting Connections

Use the `keep limit` keyword to limit the number of connections made to an IPFilter system at a given time. Connections can be limited by IP address, subnet, cumulative limit of connections, and a default individual limit.

When setting the limit of connections, be aware that the number of connections stated is for each service on the destination IP address. For example, if the `keep limit` is set to 5, then five connections are allowed for telnet, five for http, and so on.

Limiting Connections by IP Address

Use the following rule to limit connections by IP address:

```
pass [return-rst] in quick proto tcp from <ip addr> to any port
= <port_num> keep limit <limit_num>
```

For example:

```
pass return-rst in quick proto tcp from 192.34.23.1 to any port
= 25 keep limit 5
```

The example rule limits the maximum concurrent connections to 5 from host 192.34.23.1 to SMTP port 25 of any host. Because the `[return-rst]` option is specified, a TCP reset will be sent to the initiating TCP connection at IP address 192.34.23.1 when the connection request is blocked.

Limiting Connections by Subnet

Use the following rule to limit connections by subnet:

```
pass [return_rst] in quick proto tcp from <ip_subnet> to any
port = <port_num> keep limit <limit_num>
```

For example:

```
pass in quick proto tcp from 192.168.5.0/24 to any port = 25
keep limit 4
```

The example rule limits the maximum concurrent connections to 4 from any individual host in subnet 192.168.5.0/24 to port 25 of any host.

Limiting Connections by IP Address Range

Use the following rule to limit connections for each IP address within an IP address range:

```
pass [return_rst] in quick proto tcp from <ip_address_range> to
<ip | ip_address_range | ip_subnet | any> [port = <port_num>]
keep limit <limit_num>
```

For example:

```
pass in quick proto tcp from 10.10.10.1-10.10.20.1 to any port
= 25 keep limit 15
```

The example rule allows 15 connections from each IP address within the IP address range of 10.10.10.1-10.10.20.1.

Use the following rule to limit connections for all IP addresses within an IP address range:

```
pass [return_rst] in quick proto tcp from <ip_address_range> to
<ip | ip_address_range | ip_subnet | any> [port = <port_num>]
keep limit <limit_num> cumulative
```

For example:

```
pass in quick proto tcp from 10.10.10.1-10.10.20.1 to any port
= 25 keep limit 15 cumulative
```

The example rule allows 15 connections to all IP addresses within the IP address range of 10.10.10.1-10.10.20.1. The IP addresses in the stated range share the connection limit of 15. So, if there are five connections from 10.10.10.1 and ten from 10.10.20.1, then no more connections are allowed from any IP address within the stated range.

Limiting Cumulative Connections

Use the following rule to limit connections to a subnet range using a shared cumulative limit for all addresses:

```
pass [return-rst] in quick proto tcp from <ip_subnet> to any
port = <port_num> keep limit <limit_num> cumulative
```

For example:

```
pass in quick proto tcp from 192.168.7.0/24 to any port = 25
keep limit 15 cumulative
```

The example rule limits the cumulative concurrent connections to 15 from all hosts in subnet 192.168.7.0/24 to port 25 of any host.

Default Individual Connection Limits

Use the following rule to create default individual connection limits:

```
pass [return-rst] in proto tcp from any to any port =
<port_num> keep limit <limit_num>
```

For example:

```
pass in proto tcp from any to any port = 25 keep limit 5
```

This rule specifies a connection limit of 5 for all hosts when trying to connect to port 25.

IMPORTANT The default individual connection limit must be the last rule in the configuration file.

log limit: Logging Exceeded Connections

Use the `log limit` rule to log each connection that exceeds a configured limit in a `keep limit` rule. For example:

```
pass in log limit quick proto tcp from IP1 to Server keep limit
10
```

IP1 is allowed to open only 10 connections at a time. Any subsequent connection will be blocked. Since `log limit` is set, each additional connection attempt is logged.

`log limit` generates two types of log records:

- Alert Log records—created when a source IP address is trying to exceed its configured connection limit. Every time the connection limit is exceeded, an alert log record is created.

- Summary Log records—created when a limit entry ceases to exist after all the connections for that limit entry have been closed. This log record summarizes the connection activity of a particular IP address.

The format of an alert log record is:

```
Date and time stamp, Interface packet is on, Source IP, Source
port, Destination IP, Destination Port, protocol, TCP flags
keep limit, Limit type, Configured Limit, Current # of
connections, # times limit exceeded, Log freq, Packet Direction
```

The format of a summary log record is:

```
Date and time stamp, Source IP, Source port, Destination IP,
Destination Port, protocol, TCP flags keep limit, Limit type,
Configured Limit, Current # of connections, # times limit
exceeded, Rule #, Time limit the entry was created
```

Summary Logs and Cumulative Limits

The summary logs for cumulative limits can be printed using the `ipmon -r` option. When `ipmon -r` is invoked, the summary log record is written and the connection exceeded counter for each cumulative limit is set to zero.

NOTE Unlike non-cumulative limits, cumulative summary logs are not printed when all the connections under a cumulative limit are closed.

The following is an example cumulative summary log:

```
06/02/2004 19:32:39.370000 LIMIT LOG 19.13.15.65-19.13.15.85,*
-> 0.0.0.0,23 PR ip Type 4 Cur Lim 1 Exceeded 1 @0:1 First Time
19:32:35.800000
```

The example log record was written for the following IP address range cumulative rule:

```
pass in log limit freq 1 quick proto tcp from
19.13.15.65-19.13.15.85 to any port = 23 keep limit 1
cumulative
```

In the example summary log, the source IP address displayed is actually the IP address range specified in the rule. Wildcard IP addresses are shown as `0.0.0.0`. The destination port information is also printed from the rule. The other fields are similar to a non-cumulative summary record.

For further information, see “ipmon and DCA Logging” on page 95.

log limit freq: Log Frequency

Use the `log limit freq <num>` keyword to control the frequency at which alert log records are logged.

For example, `log limit` is set to 10 and `log limit freq` is set to 3. The system begins tracking exceeded connections at the eleventh connection. It logs every third exceeded connection, that is the fourteenth, seventeenth, twentieth, and so on.

The `log limit freq` keyword can also be used with `keep limit` cumulative rules. For example:

```
pass in log limit freq 5 quick proto tcp from 18.9.90.0/24 to
any keep limit 10 cumulative
```

In the previous rule, `log limit freq 5` specifies that the log records should be printed for every five connections that exceeds the connection limit of 10. If 100 connections came in, it logs the eleventh, sixteenth, twenty-first, and so on.

Cumulative limits are shared by different IP addresses and it is possible that connections from some source IPs will not display. For example, the initial connections might come from IP1 and the next 10 from IP2. IP1 will not be logged, but IP2 will be logged, as one of its connections will be the eleventh connection.

DCA Rule Syntax

The following is the complete syntax for creating a DCA rule:

```
pass [return-rst] in [log limit [freq <num>]] quick proto tcp
from <ip | ip_subnet | ip_address_range | any > to
<ip | ip_subnet | any> [port = port_num] keep limit <num>
[cumulative]
```

NOTE Be sure to use the `quick` keyword in all DCA rules.

DCA Rule Conditions

DCA rules must conform to the following conditions:

- The rule must be a `quick` rule.
- The rule must be an `in` rule.
- The rule can be used only with `proto tcp`.
- The `log limit` and `log limit freq #` rules can only be used with the `keep limit` rule.
- The source port must be a wildcard (*).
- Port ranges are not allowed for source ports.
- The connection limit specified in a `keep limit` rule must be a non-zero, positive number. `keep limit 0` rules are not allowed.
- You cannot use the `keep state` keyword with the `keep limit` keyword in the same rule.
- If `keep limit` is used, TCP state is kept on all connections that are within the limit and are allowed through.

keep limit Rules and Rule Hits

For each new packet, every time there is a rule match, the hit count for that rule is incremented. The rule does not have to be the final matching rule. Some examples are:

- A rule is a matching, non-quick rule. If another rule match is later found on the list, both hit counts are incremented.
- A rule is a matching group head. If a matching rule is found within the group, both hit counts are incremented.

Rule hit count can be displayed using `ipfstat -ioh`. This command is useful as a troubleshooting mechanism, along with `ipfstat -sl` and `ipfstat-vL`, which allow connections to be examined in realtime. And lastly, logging can be used to analyze history for past connections.

The rule hits are registered differently for cumulative and non-cumulative limits. A rule hit is usually registered only once for non-cumulative limits because, when the connection matches a non-cumulative keep limit rule, a limit entry is created and subsequent connections are controlled by that limit entry.

For cumulative limits, each new connection registers a rule hit and displays in the rule hit count because cumulative limit connections require a rule walk for each new connection.

DCA Rule Modifications

The following sections describe how to modify DCA rules when HP-UX IPFilter is running.

NOTE HP recommends configuring a redundant rule, such as `pass in all`, in all DCA rules files. IPFilter does not process packets without a rule.

To modify an active rules file:

1. Run the following command:

   `ipf -f <rules file>`

2. Add new rules to the rules file.

   DCA begins processing incoming packets with the new rules as you add them.

CAUTION If a non-cumulative rule already has a connection limit entry in the limit table, DCA matches incoming packets with the same source IP address, destination IP address, and destination port to the old rule. This occurs even if you enter a new rule higher up the list in the active rules file. The new rule does not take effect until the current connection limit entry expires.

To force a new rule to take effect immediately, follow the procedures described in “Updating keep limit Rules” on page 56. Alternately, use the following procedure to modify an inactive rules file and switch it with the active rules file.

To modify an inactive rules file, then switch it with the active rules file:

1. Run the following command to add or modify rules in an inactive rules file:

   `ipf -If <rules file>`

2. Run the following command to switch the active rules file with the inactive rules file you modified:

   ```
   ipf -s
   ```

When you modify an inactive rules file, then switch it with an active rules file, DCA processes new connections according to the new rules file whether or not there are existing connection limit entries in the limit table.

TIP For performance-critical applications, HP recommends that you load rules into the inactive list, then switch the inactive rules file with the active rules file.

Updating keep limit Rules

The following sections describe procedures for updating `keep limit` rules.

Changing the Current Individual, Subnet, or IP Address Range Rule

You can dynamically lower the number of connections a `keep limit` rule allows without letting DCA pass unwanted packets while it activates the updated rules. You can also increase the connection limit for an IP address, subnet, or IP address range.

For example, your IPFilter system has many connections coming from a specific IP address range. You have a `keep limit` rule configured for that IP address range. You want to lower the connection limit in the rule so that DCA starts using the new limit immediately, before more packets from the suspect IP address range can pass through.

To change the number of connections allowed by a `keep limit` rule:

1. Create a new rule identical to the current rule except for a different `keep limit` count.

   When adding a new rule, IPFilter recognizes it as the update of an existing rule. Current limit entries made by the old rule are updated with the new connection limit when a new connection is processed. New connections are processed with the new rule.

For example, the original rule is:

```
pass in quick proto tcp from 14.13.45.0-14.13.45.255 to any
keep limit 10 cumulative
```

To decrease the limit to 5, add the following new rule:

```
pass in quick proto tcp from 14.13.45.0-14.13.45.255 to any
keep limit 5 cumulative
```

DCA detects a similar rule in the ruleset, but the limit count has changed. DCA updates the limit count in the original rule and waits until the current number of connections drops to 5. During this period, DCA does not allow any new connections, but it does not terminate any existing connections. When the number of active connections drops to 5, DCA allows 5 or fewer connections from the specified IP address range. If you increase a connection limit from a specified IP address from 15 to 20, DCA detects the change and allows up to 20 connections from the specified IP address.

If you increase the connection limit in a `keep limit` rule, DCA immediately updates the limit count and controls connections based on the new higher connection limit.

Updating a Subnet or IP Address Range Rule

To update a subnet or IP address range `keep limit` rule:

1. Add the same rule, changing only the `keep limit` value. Be sure the subnet or IP address range is identical to the old rule.

   IPFilter recognizes the new rule as an update to an existing rule. IPFilter uses the new connection limit instead of the old connection limit. Limit entries made by the old rule are updated when a new connection is processed. New connections are processed with the new rule.

Adding New keep limit Rules

The following procedures describe how to dynamically add new rules to active rules files.

To Add a New Individual keep limit Rule:

1. Add the new rule on the line before the old rule which the new rule is to replace.

2. Delete the old rule.

To Add a New Subnet or IP Address Range Rule:

1. Add the new rule on the line before the old rule which the new rule is to replace.
2. Delete the old rule.

   Limit entries made by the old rule are updated when a new connection is processed. New connections are processed with the new rule.

   To add a more specific subnet or IP address range rule, see the following section, Integrating keep limit Rules.

Integrating keep limit Rules

The following procedure describes how to add a specific subnet or IP address range rule before an existing general subnet or IP address range rule.

1. Add the new subnet or IP address range rule. Be sure to re-enter the old subnet or IP address range rule exactly as it was entered before.

   When a new connection matches an existing limit entry, the new connection will be processed by the new subnet or IP address range rule. The subnet or IP address range can be cumulative or non-cumulative.

Extracting an Individual Rule from a Subnet Rule

To extract an individual rule from a subnet rule:

1. Add the new rule on the line before the subnet rule. Be sure the subnet or IP address range rule is identical to the old rule.

   When a new connection matches an existing limit entry, the new connection will be processed by the new individual rule. The subnet or IP address range can be cumulative or non-cumulative.

DCA Variables

The following sections provide information on the fr_statemax, fr_limitmax, and fr_tcpidletimeout variables, and how to use the kmtune command to configure each of these variables.

fr_statemax

The purpose of the fr_statemax variable is to restrict how many state entries can be created. Configure the values of this variable appropriately for your environment.

The following table displays the default and minimum values for fr_statemax. HP recommends not setting the value below the stated minimum value. For information on changing fr_statemax and other variables using kmtune, see “Configuring Variables” on page 60.

Variable Name	Default Value	Minimum
fr_statemax	200,000 entries	4,000 entries

Memory is allocated for state and limit entries in chunks. For state entries, memory is allocated by increments of 1,300 entries. For limit entries, memory is allocated by increments of 500 entries. The approximate size of the state and limit entry is 384 and 96 bytes respectively. HP-UX IPFilter keeps the allocated memory for state and limit entries in its private free pool.

IMPORTANT The state and limit values should not be set too high because the memory allocations are not released back to the kernel memory pool for general use.

Limits of fr_statemax

The fr_statemax variable indicates the number of state entries that can be created and exist at the same time. The number of state entries, as well as other statistics, can be viewed in the general state table. The ipfstat -s command gives general state table statistics.

When the number of states created reaches the fr_statemax limit, HP-UX IPFilter will try to free up state entries and increments the maximum counter. If HP-UX IPFilter fails to free up state entries, then no more state entries are created. The maximum counter is incremented each time a state entry is to be created but the state table is full. If the state table is full, the connection is let through but no state entry is created. This is true even if DCA mode is enabled.

The counter No Memory indicates that the system is out of memory and no state entry can be created.

Limits of fr_limitmax

The fr_limitmax tunable has been deprecated and no longer used to control the number of limit entries that can be created on the system.

fr_tcpidletimeout

The purpose of fr_tcpidletimeout is to determine the timeout period of states kept on TCP connections that are idle.

The default timeout value is 86,400 seconds. The minimum value that can be set for fr_tcpidletimeout is 300 seconds. For information on changing the fr_tcpidletimeout variable, see the following section, “Configuring Variables”.

Configuring Variables

Use the kmtune command to query and configure DCA variables. For new values to take effect, you must unload, reconfigure, and reload the ipf module. For example, to set fr_statemax to 6,000:

1. Unload the ipf module.
   ```
   /sbin/init.d/ipfboot stop
   ```
2. Set the new value for fr_statemax.
   ```
   kmtune -s fr_statemax=6000
   ```
3. Configure the module for the new value using the following commands:
   ```
   cd /stand/ipf
   config -M ipf -u
   ```
4. Reload the ipf module.
   ```
   /sbin/init.d/ipfboot start
   ```

DCA Mode

The DCA mode can be disabled, enabled, queried, or toggled between disabled and enabled by using the `ipf -m <option>`.

DCA mode is disabled by default. To enable DCA, run the following command:

```
ipf -m e
```

To disable DCA, run the following command:

```
ipf -m d
```

To query the current DCA setting, use the following command:

```
ipf -m q
```

You can toggle between being enabled or disabled by using the following command:

```
ipf -m t
```

To automatically enable DCA:

1. Open `/etc/rc.config.d/ipfconf`, the IPFilter startup configuration file.
2. Set the `DCA_START` flag to 1 to enable DCA.

   or

   Set the `DCA_START` flag to 0 to disable DCA.

NOTE When there are no `keep limit` rules and no connection allocation configured, HP recommends that you disable DCA.

4 Firewall Building Concepts

This chapter describes specific configuration procedures for HP-UX IPFilter. It contains concepts for basic and advanced firewall design using HP-UX IPFilter features.

It contains the following sections:

- Blocking Services by Port Number
- Using Keep State
- Using Keep State with UDP
- Using Keep State with ICMP
- Logging Techniques
- Improving Performance with Rule Groups
- Localhost Filtering
- Using the to Keyword to Capture Blocked Packets
- Creating a Complete Filter by Interface
- Combining IP Address and Network Interface Filtering
- Using Bidirectional Filtering Capabilities
- Using port and proto to Create a Secure Filter

NOTE Most of the information in this chapter has been derived from the IP Filter-based Firewalls HOWTO document written by Brendan Conoby and Erik Fichtner. You can find this document at http://www.obfuscation.org/ipf/.

Blocking Services by Port Number

To create a ruleset that explicitly passes packets for a specific service or services, but blocks all other traffic:

1. Configure the first rule to block all traffic.
2. Configure subsequent rules pass packets to specific services by port number.

For example, to create a firewall on a Web server that will accept connections on TCP port 80 only, configure the following ruleset:

```
block in on lan0 all
pass  in quick on lan0 proto tcp from any to 20.20.20.1/32 port
 = 80
```

This machine will pass in port 80 traffic for 20.20.20.1 and deny all other traffic. This ruleset provides a basic firewall.

Using Keep State

The `keep state` keyword must be used with other IPFilter keywords and filtering techniques so that IPFilter completely and correctly makes an entry in the state table.

If you configure rules to both filter on TCP flags and keep state, you must be sure you configure the rules correctly. In most cases, you should use the `keep state` keyword on the first rule that interacts with a packet for a connection. You might also need to add the `keep state` keyword to subsequent rules in the ruleset.

The following rules do not filter on TCP flags, but use the `keep state` keyword correctly:

```
block in all
pass in quick proto tcp from any to 20.20.20.20/32 port = 23
 keep state
block out all
```

The following rules both filter on TCP flags and use the `keep state` keyword:

```
block in all
pass in quick proto tcp from any to 20.20.20.20/32 port = 23
flags S keep state
pass out all keep state
```

Either of these sets of rules will result in a fully established state entry for a connection to your server.

For more examples of correct uses of the `keep state` keyword, see Appendix A, “HP-UX IPFilter Configuration Examples,” on page 147.

Protecting SSH Server Connections Using Keep State

The previous examples demonstrate keeping state on TCP, UDP, and ICMP. The IPFilter system can make outgoing connections seamlessly, and attackers cannot get back into the system. The ruleset specifies the ports systems can access, and adds entries to the state table to monitor each connection.

To protect an SSH server using the `keep state` keyword, use the following ruleset:

```
pass in quick on lan0 proto tcp from any to 20.20.20.1/32 port
= 22 keep state
pass out quick on lan0 proto tcp from any to any keep state
block in quick all
block out quick all
```

With this ruleset, IPFilter enters the first packet of a connection in the state table. Other processing works as expected. When the three-way handshake has been witnessed by the state engine, it is marked in 4/4 mode (the connection is marked as fully established). It is set up for long-term data exchange until the connection is torn down; at that time the mode will change again. You can see the current modes of your state table using `ipfstat`. See “The ipfstat Utility” on page 86 for more information.

NOTE The `keep state` keyword can create states even if it detects packets for a connection that are part of the middle of a connection. The only exception to this is when the `flags S` rule is also specified. In such a case, a state would only be created when the SYN packet is detected.

Using Keep State with UDP

You can configure IPFilter rules for UDP connections using the `keep state` keyword. An entry is added to the state table for UDP connections, the same as with a TCP connection acted on by a rule with the `keep state` keyword. For example:

```
pass out on lan0 proto udp from any to any port 33434><33690
 keep state
```

For more information on using the `keep state` keyword, see “keep state: Protecting TCP, UDP, and ICMP Sessions” on page 34 and “Using Keep State” on page 66.

Using Keep State with ICMP

The majority of ICMP messages are status messages generated by a failure in UDP or TCP. For any ICMP error status message that matches an active state table entry that might have generated that message, IPFilter passes the ICMP packet. For example:

```
pass out on lan0 proto udp from any to any port 33434><33690
 keep state
```

Even though an error status message (such as `icmp-type 3 code 3 port unreachable` or `icmp- type 11 time exceeded`) for the UDP session is an ICMP packet, the `keep state` rule passes the error message.

The two types of ICMP messages are requests and replies. You can configure a rule to pass outbound echo requests such as ping. IPFilter passes in the subsequent icmp-type 0 packet that returns. For example:

```
pass out on lan0 proto icmp from any to any icmp-type 8 keep
 state
```

This state entry has a default timeout of an incomplete 0/0 state of 60 seconds.

NOTE If you configure rules to keep state on any outbound ICMP messages that might receive a reply ICMP message, you must use both the `proto icmp` and the `keep state` keywords.

To provide protection against a third party sneaking ICMP messages through your firewall when an active connection is known to be in your state table, check the incoming ICMP packet not only for matching source and destination addresses (and ports, when applicable), but a tiny part of the payload of the packet that the ICMP message is claiming it was generated by.

Logging Techniques

The log keyword tells IPFilter to log packets matching the rule to the IPFilter logging device, `/dev/ipl`. To read the log, run the `ipmon` utility. See "The ipmon Utility" on page 93 for more information. You can use the `ipmon -s` command to log the information in `/dev/ipl` to `syslog`.

You can use the following advanced options with the log keyword to refine the log IPFilter creates.

level *log-level*

You can control the level of logging IPFilter does by using the `level` *`log-level`* option with the `log` keyword.

The syntax for level is:

`log level` *`facility.priority|priority`*

The options available for *`facility`* are:

kern	user	mail
daemon	auth	syslog
lpr	news	uucp
cron	ftp	authpriv
audit	logalert	local0
local1	local2	local3
local4	local5	local6
local7		

The options available for *`priority`* are:

emerg	alert	crit
err	warn	notice
info	debug	

Example:

```
block in log level auth.info quick on lan0 from 20.20.20.0/24
 to any
block in log level auth.alert quick on lan0 proto tcp from any
 to 20.20.20.0/24 port = 21
```

first

You can use the `first` option with the `log` keyword to log only the first instance of a certain type of packet. For example, it might not be important to log 500 attempts to probe your telnet port from one source. It is a good idea to log the first attempt, however.

The `first` option only applies to packets in a specific session. You can use the first option to monitor traffic on your system. For best results, use the first option in conjunction with rules that use `pass` and `keep state`.

Example:

```
pass in log first proto tcp from amy to any flags S keep state
```

body

You can use the `body` option with the `log` keyword to track parts of an IP packet in addition to the packet header information. IPFilter logs the first 128 bytes of a packet if the `body` option is specified. For example:

```
block in log body proto tcp from 192.168.1.1 to any flags S
keep state
```

NOTE Using the `body` option with the `log` keyword can make your log files very long. Limit the use of the `body` option to necessary instances.

Improving Performance with Rule Groups

Rule groups allow you to write your ruleset in a tree structure, instead of as a linear list, so that if an incoming packet is unrelated to a set of rules, those rules will never be processed. This reduces IPFilter processing time on each packet and improves IPFilter system performance.

The following is a simple rule group example:

```
block out quick on lan1 all head 10
pass out quick proto tcp from any to 20.20.20.64/26 port = 80
 flags S keep state group 10
block out on lan2 all
```

In this example, if the packet is not destined for `lan1`, the head of rule `group 10` does not match; IPFilter does not process any of the rules in `group 10`. Rules processing continues at the root level (group 0). If the packet does match `lan1`, the `quick` keyword stops further processing at the `group 0` level. IPFilter then processes all rules in `group 10` against the packet.

Rule groups can be used to break up a complex firewall ruleset. For example, there are three interfaces in the firewall with interfaces `lan0`, `lan1`, and `lan2`.

- `lan0` is connected to external network 20.20.20.0/26.
- `lan1` is connected to DMZ network 20.20.20.64/26.
- `lan2` is connected to protected network 20.20.20.128/25.

A complete ruleset for this situation would be complex and significantly slow user connections to the network. To prevent this, a ruleset is created with rule groups:

```
block in quick  on lan0 all head 1
block in quick on lan0 from 192.168.0.0/16 to any group 1
block in quick on lan0 from 172.16.0.0/12 to any group 1
block in quick on lan0 from 10.0.0.0/8 to any group 1
block in quick on lan0 from 127.0.0.0/8 to any group 1
block in log quick on lan0 from 20.20.20.0/24 to any group 1
block in log quick on lan0 from any to 20.20.20.0/32 group 1
block in log quick on lan0 from any to 20.20.20.63/32 group 1
block in log quick on lan0 from any to 20.20.20.64/32 group 1
block in log quick on lan0 from any to 20.20.20.127/32 group 1
block in log quick on lan0 from any to 20.20.20.128/32 group 1
```

```
block in log quick on lan0 from any to 20.20.20.255/32 group 1
pass  in on lan0 all group 1
pass out on lan0 all
block out quick on lan1 all head 10
pass out quick on lan1 proto tcp from any to 20.20.20.64/26
 port = 80 flags S keep state group 10
pass out quick on lan1 proto tcp from any to 20.20.20.64/26
 port = 21 flags S keep state group 10
pass out quick on lan1 proto tcp from any to 20.20.20.64/26
 port = 20 flags S keep state group 10
pass out quick on lan1 proto tcp from any to 20.20.20.65/32
 port = 53 flags S keep state group 10
pass out quick on lan1 proto udp from any to 20.20.20.65/32
 port = 53 keep state group 10
pass out quick on lan1 proto tcp from any to 20.20.20.66/32
 port = 53 flags S keep state group 10
pass out quick on lan1 proto udp from any to 20.20.20.66/32
 port = 53 keep state group 10
```

For a host on the `lan2` network, IPFilter bypasses all the rules in `group 10` when a packet is not destined for hosts on that network.

Multi-level grouping is also supported, allowing IPFilter rules to be arranged in hierarchical, nested groups. By using the `head` and `group` keywords in a rule, multi-level grouping allows the user to fine tune a range to improve performance. The following is an example of a multi-level rule grouping:

```
pass in proto tcp from 1.0.0.0-9.0.0.0 to any port = 23 keep
state head 1
pass in proto tcp from 2.0.0.0-8.0.0.0 to any port = 23 keep
state head 2 group 1
pass in proto tcp from 3.0.0.0-7.0.0.0 to any port = 23 keep
state head 3 group 2
pass in proto tcp from 4.0.0.0-6.0.0.0 to any port = 23 keep
state head 4 group 3
pass in proto tcp from 5.0.0.0-5.5.0.0 to any port = 23 keep
state group 4
```

You can group your rules by protocol, machine, netblock, or other logical criteria that help system performance. There is not a hard limit to the number of group levels you can maintain. For more information, see Appendix C, “Performance Guidelines,” on page 181.

Localhost Filtering

Use localhost filtering with IPFilter to provide both security and convenience for your users.

Localhost filtering with IPFilter can be used effectively in conjunction with other security products, such as external firewalls and internal software products.

The following example is a ruleset configured to run on a machine that also uses TCP Wrapper to protect its network services.

```
pass in quick on lan0 all
pass out quick on lan0 all
block in log all
block out all
pass in quick proto tcp from any to any port = 113 flags S keep
 state
pass in quick proto tcp from any to any port = 22 flags S keep
 state
pass in quick proto tcp from any port = 20 to any port 39999 >
 < 45000 flags S keep state
pass out quick proto icmp from any to any keep state
pass out quick proto tcp/udp from any to any keep state keep
 frags
```

This IPFilter ruleset provides enhanced protection for the system and services using TCP Wrapper. Any security holes left by TCP Wrapper are plugged.

No negative impact results from running IPFilter all the time.

Using the to Keyword to Capture Blocked Packets

You can use the `to` keyword apart from the `from` keyword. If you want to block a packet, you can use the `to` keyword to push the packet past the normal routing table and force it to go out on a different interface. For example:

```
block in quick on lan0 to lan1 proto tcp from any to any port <
 1024
```

This rule blocks incoming packets, but also forces them over to the `lan1` interface, where they can be logged. If you log blocked packets this way, you can then analyze blocked traffic for possible attacks on the system.

Use `block quick` for `to` interface routing because the `to` interface code will generate two packet paths through IPFilter when used with `pass`.

NOTE If you are configuring rules to pass packets, but also want the packets to go to another interface, use the `dup-to` keyword. See “dup-to: Drop-Safe Logging” on page 38.

Creating a Complete Filter by Interface

When you create a ruleset, you should set up rules for all directions and all interfaces. The default state of IPFilter is to pass packets both in and out. Instead of relying on the IPFilter default behavior, make every ruleset as specific as possible, interface by interface, until all possibilities are explicitly covered.

For example, if you have an IPFilter system with a `lan1` interface, and a `lan0` interface, configure the following rules:

```
pass out quick on lan1
pass in quick on lan1
```

```
block out quick on lan0 from any to 192.168.0.0/16
block out quick on lan0 from any to 172.16.0.0/12
block out quick on lan0 from any to 10.0.0.0/8
pass  out quick on lan0 from 20.20.20.0/24 to any
block out quick on lan0 from any to any
block in quick on lan0 from 192.168.0.0/16 to any
block in quick on lan0 from 172.16.0.0/12 to any
block in quick on lan0 from 10.0.0.0/8 to any
block in quick on lan0 from 127.0.0.0/8 to any
block in log quick on lan0 from 20.20.20.0/24 to any
pass in all
```

In this example, no restrictions are on traffic in and out on `lan1`. Traffic has significant restrictions both in and out of `lan0`.

NOTE When setting up your ruleset, be sure that you add rules for all appropriate directions and interfaces.

Combining IP Address and Network Interface Filtering

If you know that your system will send and receive packets only from specific IP addresses and interfaces, configure your IPFilter rules to only allow traffic from those addresses and interfaces.

Also, there are addresses and subnets used for specific purposes on specific interfaces. The following examples show rulesets that block packets coming to or from places that should not have traffic.

For example, to block private address space to keep it from entering `lan0`:

```
block in quick on lan0 from 192.168.0.0/16 to any
block in quick on lan0 from 172.16.0.0/12 to any
block in quick on lan0 from 10.0.0.0/8 to any
block in quick on lan0 from 127.0.0.0/8 to any
pass in all
```

It is common for software to communicate with itself on 127.0.0.1. Therefore, it is good practice to block any packets coming from this address from outside. Also, no packets from 10.0.0.0/8 should come in on `lan0` because such packets cannot have a reply.

If you have an internal network, you can be sure that traffic destined for the network should only be coming from addresses within that network. If a packet that comes from an address on the internal network arrives on a dialup interface, it should be blocked by IPFilter.

For example, if your internal network subnet is 20.20.20.0/24, use the following rules to keep traffic from this subnet from passing through on the external `lan0` interface:

```
block in quick on lan0 from 192.168.0.0/16 to any
block in quick on lan0 from 172.16.0.0/12 to any
block in quick on lan0 from 10.0.0.0/8 to any
block in quick on lan0 from 127.0.0.0/8 to an
block in quick on lan0 from 20.20.20.0/24 to any
pass in all
```

Using Bidirectional Filtering Capabilities

You can use bidirectional filtering to limit packets leaving a system to those that come from a specific subnet. For example, to limit traffic passing out of the IPFilter system to packets coming from the 20.20.20.0/24 subnet, configure the following rules:

```
pass out quick on lan0 from 20.20.20.0/24 to any
block out quick on lan0 from any to any
```

If a packet originates from IP address 20.20.20.1/32, it is sent out by the first rule. If a packet originates from IP address 1.2.3.4/32, it is blocked by the second rule.

You can also configure similar rules for unroutable addresses. If a machine routes a packet through IPFilter with a destination of 192.168.0.0/16, you can drop it to save bandwidth. Use the following ruleset:

```
block out quick on lan0 from any to 192.168.0.0/16
block out quick on lan0 from any to 172.16.0.0/12
block out quick on lan0 from any to 10.0.0.0/8
```

This enhances the security of other systems. Spoofed packets cannot be sent from your site.

NOTE The in and out directions refer to the IPFilter system *only*.

Using port and proto to Create a Secure Filter

To configure IPFilter for effective security, use several techniques and building blocks together.

For example, you can configure rules to allow rsh, rlogin, and telnet to run only on your internal network. Your internal network subnet is 20.20.20.0/24. All three services use specific TCP ports (513, 514, and 23). Configure the following rules in the following order:

```
pass in quick on lan0 proto icmp from any to 20.20.20.0/24
icmp-type 0
pass in quick on lan0 proto icmp from any to 20.20.20.0/24
icmp-type 11
block in log quick on lan0 proto icmp from any to any
block in log quick on lan0 proto tcp from any to 20.20.20.0/24
port = 513
block in log quick on lan0 proto tcp from any to 20.20.20.0/24
port = 514
block in log quick on lan0 proto tcp from any to 20.20.20.0/24
port = 23
pass in all
```

Be sure the rules for the services are placed before the `pass in all` rule to close them off to systems outside your network.

To block UDP instead of TCP, replace `proto tcp` with `proto udp`. The rule for syslog would then be:

```
block in log quick on lan0 proto udp from any to 20.20.20.0/24
 port = 514
```

Several services allow you to block by port number for security:

- syslog on UDP port 514
- portmap on TCP port 111 and UDP port 111
- lpd on TCP port 515
- NFS on TCP port 2049 and UDP port 2049
- X11 on TCP port 6000

To get a complete listing of ports being listed on, use `netstat -a`, or check `/etc/services`.

5 HP-UX IPFilter Utilities

This chapter describes IPFilter utilities. It contains the following sections:

- The ipf Utility

- The ipfstat Utility
- The ipmon Utility
- The ipftest Utility
- The ipnat Utility
- Unsupported Utilities and Commands

NOTE Most of the information in this chapter has been derived from the IP Filter-based Firewalls HOWTO document written by Brendan Conoby and Erik Fichtner. You can find this document at http://www.obfuscation.org/ipf/.

The ipf Utility

The `ipf` utility performs a broad range of actions on the active and inactive IPFilter rulesets. You can use `ipf` to add rules, delete rules, switch active and inactive rulesets, and flush the existing ruleset from the system. You can perform other actions with `ipf`. See the *ipf* manpages for more information.

Syntax

`ipf <-options> <rules file name>`

Options

The following are a few of the common options used with the `ipf` utility:

`-s`

Switches the active rules file with the inactive rules file.

`-Fa`

Flushes all rules in the specified rules file.

`-Fi`

Flushes only the IN rules in the specified rules file.

`-Fo`

Flushes only the OUT rules in the specified rules file.

`-I`

Specifies that the inactive rules file is to be manipulated.

`-Z`

Zeroes out the `TCP Connections` counters displayed in the `ipfstat` output.

`-m <d|e|q|t>`

Disables or enables DCA mode, queries the DCA mode, or toggles DCA between being enabled or disabled by using the following options:

- `d`

  Disables DCA.

- `e`

  Enables DCA.

- `q`

  Queries whether DCA is disabled or enabled.

- `t`

  Toggles DCA between disabled or enabled.

When there are no `keep limit` rules and there is no connection allocation, disable DCA. See “DCA Mode” on page 61 for more information about how to disable, enable, query, or toggle DCA.

`-E <interface name>`

Enables IPFilter processing for traffic on a given interface.

`-D <interface name>`

Disables IPFilter processing for traffic on a given interface.

`-Q <interface name>`

Verifies that IPFilter processing is enabled or disabled for a given interface.

The `-E`, `-D`, and `-Q` commands let you control IPFilter processing on a given interface. For example, `ipf -D lan0` disables IPFilter processing for traffic on lan0 and `ipf -E lan0` enables IPFilter processing on lan0. `ipf -Q lan0` is used to verify if IPFilter processing is enabled or disabled for lan0.

NOTE

All `ipf` actions are performed on the active rules file by default. To perform actions on the inactive rules file, you must specify the `-I` option.

For a complete list of `ipf` options and their uses, see the *ipf*(5) and *ipf*(8) manpages.

Example

Enter the following command to load a ruleset:

```
ipf -Fa -f <rules file>
```

The ipfstat Utility

The `ipfstat` utility displays a table of data detailing firewall performance, including how many packets have been passed or blocked, whether the packets were logged or not, how many state entries have been made, and DCA statistics. You can also use options with `ipfstat` to display active rules.

Syntax

`ipfstat <-options>`

Options

`-i`

Displays currently loaded rules for inbound packets.

`-o`

Displays currently loaded rules for outbound packets.

`-h`

Displays the hit count for each rule as well as the rules themselves. Use with `-i` or `-o` options.

`-s`

Displays state table statistics.

`-sl`

Displays detailed state table statistics.

`-n`

Displays the number of each rule next to the rule itself.

`-L`

Displays global limit statistics.

`-v-L`

Displays detailed global limit statistics.

`-r <group:rule>`

Displays the limit statistic by rule number.

`-v`

Sets verbose mode. Use for debugging.

NOTE Statistics counters cannot increment when both active in and out rule sets are empty. This is due to a performance optimization that bypasses IPFilter when there are no active rule sets present.

For a complete list of options used with `ipfstat`, see the *ipfstat* manpage.

Examples

```
# ipfstat
dropped packets:        in 0    out 0
non-data packets:       in 0    out 0
no-data packets:        in 0    out 0
non-ip packets:         in 0    out 0
   bad packets:         in 0    out 0
copied messages:        in 0    out 0
   input packets: blocked 15 passed 2647 nomatch 2537 counted 0
short 0
   output packets: blocked 0 passed 245 nomatch 141 counted 0
short 0
input packets logged:   blocked 0 passed 0
output packets logged:  blocked 0 passed 0
packets logged:         input 0  output 0
TCP connections:        in 5     out 50
log failures:           input 0  output 0
fragment state(in):     kept 0  lost 0
fragment state(out):    kept 0  lost 0
packet state(in):       kept 5  lost 0
packet state(out):      kept 0  lost 0
ICMP replies:   0       TCP RSTs sent:  0
Invalid source(in):     0
Result cache hits(in):  14      (out):  0
IN Pullups succeeded:   0       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)
   none
```

The `TCP Connections` statistics are derived from the number of states added and is valid only in the context of stateful filtering. These statistics will be accurate only when `keep limit` or `keep state` rules are used for all TCP connections.

For example, you have the following ruleset:

```
pass in log limit freq 500 quick proto tcp from any to any port
= 80 keep limit 100

pass in log quick proto tcp from any to any port = 25 flags S
keep state

pass in log quick proto tcp from any to any port = 23

pass out log quick proto tcp from any port = 23 to any
```

These rules only count connections that match the first two rules. Both the third and fourth rule allow telnet connections but telnet connections are not counted, since the system is not keeping state on these connections.

Example:

```
# ipfstat -ho

2451423  pass out on lan0 from any to any
354727 block out on ppp0 from any to any
430918  pass out quick on ppp0 proto tcp/udp from
20.20.20.0/24 From to any keep state keep frags
```

This status report shows that the ruleset may not be working as intended. Many outbound packets are being blocked despite a `pass out` rule configured to pass most outbound packets.

`ipfstat` cannot indicate whether a ruleset is configured correctly. It can only display what is happening at the present time with a given ruleset.

Set the `-n` option to display the rule number next to each rule. The rule number is displayed as `@`*`group:rule`*. This can help you determine which rules are incorrectly configured. For example:

```
# ipfstat -on

@0:1 pass out on lan0 from any to any
@0:2 block out on ppp0 from any to any
@0:3 pass out quick on ppp0 proto tcp/udp from 20.20.20.0/24 to
any keep state keep frags
```

The following example uses the `-s` option to display the state table.

```
# ipfstat -s

281458 TCP
319349 UDP
0 ICMP
19780145 hits
5723648 misses
0 maximum
0 no memory
0 bkts in use
1 active
319349 expired
281419 closed
```

A TCP connection has one state entry. One fully established connection is represented by the 4/4 state. Other states are incomplete and will be documented later. The state entry has a time life of 24 hours, which is the default for an established TCP connection. The TTL counter is decremented every second that the state entry is not used and will result in the connection being purged if it is left idle.

The TTL counter is reset to 86400 whenever the state is used, ensuring the entry will not time out while it is being actively used. 196 packets consisting of about 17KB worth of data have been passed over this connection. The ports for the endpoints are 987 and 22; this state entry represents a connection from 100.100.100.1 port 987 to 20.20.20.1 port 22. The numbers in the second line are the TCP sequence numbers for this connection. These numbers help ensure that an attacker cannot insert a forged packet into your session. The TCP window is also shown. The third line is a synopsis of the implicit rule generated by the `keep state` code showing that this is an inbound connection.

The `ipfstat -sl` option is often used in place of `ipfstat -s` to show held state information in the kernel, if present. The `ipfstat -sl` gives detailed information for each state entry that is active.

The following is an example of the output information of the `ipfstat -sl` option:

```
#ipfstat -sl
  empty list for ipfilter(out)
1 pass in quick proto tcp from 15.13.106.175/32 to any keep
state
# ipfstat -sl
15.13.106.175 -> 15.13.137.135 ttl 872678 pass 0x500a pr 6
state 4/4
  pkts 31 bytes 1564      57906 -> 23 22c0861c:712c2bd9
32768:32768
  cmsk 0000 smsk 0000 isc 0000000000000000 s0 22c085e0/712c2b7f
  sbuf[0] [\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0] sbuf[1]
[\0\0\0\0\0\0\0\0\0\0
\0\0\0\0\0\0]
  pass in quick keep state        IPv4
  pkt_flags & 2(b2) = b,          pkt_options & ffffffff = 0
  pkt_security & ffff = 0, pkt_auth & ffff = 0
interfaces: in lan0[00000000480baf00] out -[0000000000000000]
```

The following is an example of the output information of the `ipfstat -L` option.

```
Current connections to limited IP addresses
Connection Type     Active Limits
Individual               2
Subnet                   3
Cumulative               5
Unknown IP               9
Total                    19

No Memory                0
Logged Records           13
Log Failures             0
Limits Added             13
Add Failures             0
```

- The first six lines display the number of current active connections of each described type.
- `No Memory` is the number of times a limit entry could not be created because no memory was available. If this is a non-zero, positive value, then the system memory should be checked and, if necessary, increased.
- `Logged Records` is the number of limit entries logged, both summary and alert log records.

- `Log Failures` is the number of times log entries have not been logged. A non-zero, positive value for `Log Failures` indicates that the size of the kernel log buffer is small. The kernel log buffer `ipl_buff_sz` should be set to an appropriate value.
- `Limits Added` is the number of limit entries that have been added.
- `Add Failures` is the number of times a limit entry could not be created. This happens when a state entry is not added. The output of `ipfstat -s` should be used to further diagnose the problem.

These statistics are cumulative. They are automatically reset to zero when the `ipf` module is unloaded and loaded again.

See "Additional Configuration Information" on page 14 for more information on setting the size of the state table, limit table, and log buffer.

The following is an example of the output information of the `ipfstat -v-L` option:

```
Type Rule   Src IP  Src Port  Dest IP Dest Port Limit Current
S  @0:3    10.39.1.2    *    10.133.1.5   80     50000 951 (0)
S  @0:1    10.2.1.2     *    10.129.1.5   80     50000 942 (0)
U  @0:1000 10.30.1.2    *    10.130.1.5   80     10    10(102)
U  @0:1000 10.30.1.3    *    10.130.1.5   80     10    9 (501)
U  @0:1000 10.30.1.4    *    10.130.1.5   80     10    10(100)
U  @0:1000 10.30.1.5    *    10.130.1.5   80     10    10(118)
U  @0:1000 10.30.1.6    *    10.130.1.5   80     10    10(196)
U  @0:1000 10.30.1.7    *    10.130.1.5   80     10    10(198)
U  @0:1000 10.30.1.8    *    10.130.1.5   80     10    10(104)
U  @0:1000 10.30.1.0    *    10.130.1.5   80     10    10(111)
U  @0:1000 10.49.1.2    *    10.131.1.5   80     10    10 (55)
U  @0:1000 10.49.1.3    *    10.131.1.5   80     10    10 (53)
U  @0:1000 10.49.1.4    *    10.131.1.5   80     10    10(102)
U  @0:1000 10.49.1.5    *    10.131.1.5   80     10    9  (52)
U  @0:1000 10.49.1.6    *    10.131.1.5   80     10    9  (52)
U  @0:1000 10.49.1.7    *    10.131.1.5   80     10    10(103)
U  @0:1000 10.49.1.8    *    10.131.1.5   80     10    10(120)
U  @0:1000 10.49.1.9    *    10.131.1.5   80     10    10(50)
S  @0:1000 10.40.1.2    *    10.134.1.5   80     50000 943(0)
U  @0:1000 10.46.1.2    *    10.128.1.5   80     10    10 (49)
U  @0:1000 10.46.1.3    *    10.128.1.5   80     10    10 (41)
```

- The Type column displays the type of limit being kept:

  I—Fully resolved individual IP

S—IP subnet

C—Cumulative

U—Unknown IP

These limit entries are created through the default rule. See “DCA Keywords” on page 47 for detailed information on the different types of limit entries.

- The Rule column displays the rule number that caused the creation of this limit entry. This information can in turn be used to get per-rule statistics using the `ipfstat -r` command.
- The third through sixth columns display IP-port pairs of the TCP connection.
- The Limit column displays the configured limit specified in the `keep limit` rule.
- The Current column displays the number of fully established connections under that limit entry. The figure in the parenthesis indicates the number of times the configured limit was exceeded. For example, the first entry shows that, even though the IP address 15.10.40.10 currently has two active connections, it had exceeded the configured limit of 10 connections twice. These numbers can serve as guide for adjusting and tuning the limit value for an IP address or IP subnet.

The following is an example of the output information of the `ipfstat -r` *<group:rule>* option.

```
Limit Type                  Individual
Group:Rule Number           @0:6
Configured Limit            7
Current connections         3
Limit Exceeded (#times)     33
TCP RSTs sent (#times)      33
```

In this example, rule number 6 created a limit entry of type `Individual`. The rule specifies a connection limit of 7. There are three current connections using this rule. The limit has been exceeded 33 times. `return-rst` was set, so a TCP reset was sent each time an attempt was made to exceed the configured limit.

If the rule is deleted or switched to the inactive set, `@(del)` is displayed in the Group:Rule Number field.

The ipmon Utility

Use the `ipmon` utility to monitor IPFilter while it is in use.

You can use `ipmon` to watch the packet log, as created with the log keyword in the IPFilter rules. `ipmon` can also monitor the state log, the NAT log, or any combination of these three. You can run `ipmon` in the foreground or as a daemon that logs to `syslog` or a file.

Syntax

`ipmon <-options>`

Options

`-a`

Opens and reads data from all available log files. Equivalent to `-o NSI`.

`-o [NSI]`

Specifies which log file to read data from.

- N—NAT log file
- S—State log file
- I—IPFilter log file

`-A`

Logs the summary records created for DCA logging.

`-r`

Prints the summary records to the summary log file and clears the block count for each limit entry.

`-F`

Flushes the packet log buffer. Output displays the number of bytes flushed.

`-n`

Maps IP addresses and port numbers to host names and services wherever possible.

For a complete list of ipmon options and their uses, see the *ipmon* manpage.

Examples

To view the state table as it updates, use the ipmon -o S command.

Example:

```
# ipmon -o S

01/08/1999 15:58:57.836053 STATE:NEW 100.100.100.1,53
->20.20.20.15,53 PR udp

01/08/1999 15:58:58.030815 STATE:NEW 20.20.20.15,123
->128.167.1.69,123 PR udp

01/08/1999 15:59:18.032174 STATE:NEW 20.20.20.15,123
->128.173.14.71,123 PR udp

01/08/1999 15:59:24.570107 STATE:EXPIRE 100.100.100.1,53
->20.20.20.15,53 PR udp Pkts 4 Bytes 356

01/08/1999 16:03:51.754867 STATE:NEW 20.20.20.13,1019
->100.100.100.10,22 PR tcp

01/08/1999 16:04:03.070127 STATE:EXPIRE 20.20.20.13,1019
->100.100.100.10,22 PR tcp Pkts 63 Bytes 4604
```

A state entry for an external DNS request to the nameserver is displayed by ipmon. Two xntp pings to well-known time servers and a short outbound SSH connection are also displayed.

You can also use ipmon to display packets that have been logged.

To view the IPFilter packet log, use the ipmon -o I command.

Example:

```
# ipmon -o I

15:57:33.803147 ppp0 @0:2 b 100.100.100.103,443 ->
20.20.20.10,4923 PR tcp len 20 1488 -A:
```

The fields in this output are as follows:

- Field 1—Time stamp
- Field 2—The interface on which the event occurred

- Field 3—Rule group number: rule number of the rule that acted on the packet
- Field 4—Blocked (b) or Passed (p) packet
- Field 5—Packet origin
- Field 6—Packet destination
- Field 7 and 8—Protocol used
- Field 9—Packet size
- Field 10—Flags set on packet

Run the `ipfstat -in` command to determine which rule caused the problem. In this example, you would use this command to look at rule 2 in rule group 0.

Occasionally, a packet that was part of a state connection might appear in the `ipmon -o I` log. This can happen if a packet with the same sequence number as another packet is processed by IPFilter. A state packet might also be logged by the regular IPFilter log if it is the last packet in a stateful connection, and arrives after the state has been torn down by IPFilter.

Example:

```
#ipfstat -n

12:46:12.470951 lan0 @0:1 S 20.20.20.254 -> 255.255.255.255 PR
 icmp len 20 9216 icmp 9/0
```

This is a ICMP router discovery broadcast. It is indicated by the ICMP type 9/0.

ipmon and DCA Logging

DCA logging creates a new device file. The log alerts records go to `/dev/ipl` and the summary records are logged to `/dev/iplimit`. To log the summary records, use `ipmon` with the `-A` option. Using `ipmon -A` prints a summary log for a limit entry before the entry being removed from the limit table.

Example:

```
ipmon -A /dev/iplimit > $LOGDIR/limit_summary.log &
```

You can use `ipmon -r` to print the summary records to the log file for all existing limit entries that are active. For example, you have the following rule configured:

```
pass in log limit quick proto tcp from IP1 to Server keep
limit 10
```

If IP1 creates 70 connections, then 10 connections are let through and remaining 60 are blocked, which is the block count. When `ipmon -r` is called, a summary record is logged to the summary log records and the block count is set to 0. This is useful in a case where IP1 created many connections and has a large block count, but subsequently has connections that are within the connection limit.

`ipmon -r` works only on active limit entries. If there are no limit entries, `ipmon -r` does not log any Summary Log records. Summary logs are printed only for those limit entries which have a non-zero connection exceeded counter. For cumulative limits, this option is the only way to obtain summary logs.

The ipftest Utility

Use the `ipftest` utility to test your ruleset in user space without compromising the security of your IPFilter system. The `ipftest` utility can be run by a non-root user.

The `ipftest` utility tests a ruleset using a set of packet descriptions that simulate real network traffic. Actions taken by IPFilter on each simulated packet are written to `stdout`.

When you generate simulated traffic, you can use example data obtained from a packet probe or similar monitor. These packets can show the specifics of the traffic the subject machine will encounter in a production environment. Be sure to include the various flags in TCP packets, as they are used in the various `keep state` rules.

Syntax

`ipftest` *<options><filename>*

Options

`-i` *<filename>*

Specifies the file from which to take input. The default is `stdin`.

`-r` *<filename>*

Specifies the rules file from which to read rules.

Many other options are available to refine testing with `ipftest`. For a complete list of options and their functions, see the *ipftest* manpage.

Example

The following ruleset is used for this example:

```
block in all
pass in from 10.1.84.195 to any
```

The following packets will be used to test this rule set:

```
in on lan0 udp 10.1.84.195,16000 10.1.84.196,16000
in on lan1 udp 10.1.84.195,16000 10.1.85.196,16000
in on lan0 udp 10.1.84.195,16000 10.1.80.196,16000

in on lan0 udp 10.1.85.195,16000 10.1.84.196,16000
in on lan1 udp 10.1.85.195,16000 10.1.85.196,16000
in on lan0 udp 10.1.85.195,16000 10.1.80.196,16000

out on lan0 udp 10.1.84.196,16000 10.1.84.195,16000
out on lan1 udp 10.1.85.196,16000 10.1.84.195,16000
out on lan0 udp 10.1.80.196,16000 10.1.84.195,16000

out on lan0 udp 10.1.84.196,16000 10.1.85.195,16000
out on lan1 udp 10.1.85.196,16000 10.1.85.195,16000
out on lan0 udp 10.1.80.196,16000 10.1.85.195,16000

in on lan0 udp 10.1.81.195,16000 10.1.84.196,16000
in on lan1 udp 10.1.81.195,16000 10.1.85.196,16000

out on lan0 udp 10.1.84.196,16000 10.1.81.195,16000
out on lan1 udp 10.1.85.196,16000 10.1.81.195,16000

out on lan0 icmp 10.1.84.196 10.1.84.195
in on lan0 icmp 10.1.84.195 10.1.84.196

out on lan0 udp 10.1.80.196,16001 10.1.84.195,16000
out on lan0 udp 10.1.80.196,16001 10.1.85.195,16000

in on lan0 udp 10.1.84.195,16000 10.1.80.196,16001
in on lan0 udp 10.1.85.195,16000 10.1.80.196,16001
```

These packets are similar to a test machine setup that is used in the actual testing of IPFilter. The name of the rules file is `test01` and the name of the packet file is `packets01`. The packets are processed with `ipftest` using the following command:

```
ipftest -r test01 -i packets01
```

The following is the output of `ipftest`:

```
opening rule file "test01"
input: in on lan0 udp 10.1.84.195,16000 10.1.84.196,16000
pass ip 28(20) 17 10.1.84.195,16000 > 10.1.84.196,16000
--------------
input: in on lan1 udp 10.1.84.195,16000 10.1.85.196,16000
pass ip 28(20) 17 10.1.84.195,16000 > 10.1.85.196,16000
```

```
--------------
input: in on lan0 udp 10.1.84.195,16000 10.1.80.196,16000
pass ip 28(20) 17 10.1.84.195,16000 > 10.1.80.196,16000
--------------
input: in on lan0 udp 10.1.85.195,16000 10.1.84.196,16000
block ip 28(20) 17 10.1.85.195,16000 > 10.1.84.196,16000
--------------
input: in on lan1 udp 10.1.85.195,16000 10.1.85.196,16000
block ip 28(20) 17 10.1.85.195,16000 > 10.1.85.196,16000
--------------
input: in on lan0 udp 10.1.85.195,16000 10.1.80.196,16000
block ip 28(20) 17 10.1.85.195,16000 > 10.1.80.196,16000
--------------
input: out on lan0 udp 10.1.84.196,16000 10.1.84.195,16000
nomatch ip 28(20) 17 10.1.84.196,16000 > 10.1.84.195,16000
--------------
input: out on lan1 udp 10.1.85.196,16000 10.1.84.195,16000
nomatch ip 28(20) 17 10.1.85.196,16000 > 10.1.84.195,16000
--------------
input: out on lan0 udp 10.1.80.196,16000 10.1.84.195,16000
nomatch ip 28(20) 17 10.1.80.196,16000 > 10.1.84.195,16000
--------------
input: out on lan0 udp 10.1.84.196,16000 10.1.85.195,16000
nomatch ip 28(20) 17 10.1.84.196,16000 > 10.1.85.195,16000
--------------
input: out on lan1 udp 10.1.85.196,16000 10.1.85.195,16000
nomatch ip 28(20) 17 10.1.85.196,16000 > 10.1.85.195,16000
--------------
input: out on lan0 udp 10.1.80.196,16000 10.1.85.195,16000
nomatch ip 28(20) 17 10.1.80.196,16000 > 10.1.85.195,16000
--------------
input: in on lan0 udp 10.1.81.195,16000 10.1.84.196,16000
block ip 28(20) 17 10.1.81.195,16000 > 10.1.84.196,16000
--------------
input: in on lan1 udp 10.1.81.195,16000 10.1.85.196,16000
block ip 28(20) 17 10.1.81.195,16000 > 10.1.85.196,16000
--------------
input: out on lan0 udp 10.1.84.196,16000 10.1.81.195,16000
nomatch ip 28(20) 17 10.1.84.196,16000 > 10.1.81.195,16000
--------------
input: out on lan1 udp 10.1.85.196,16000 10.1.81.195,16000
nomatch ip 28(20) 17 10.1.85.196,16000 > 10.1.81.195,16000
--------------
input: out on lan0 icmp 10.1.84.196 10.1.84.195
nomatch ip 48(20) 1 10.1.84.196 > 10.1.84.195
--------------
```

```
input: in on lan0 icmp 10.1.84.195 10.1.84.196
pass ip 48(20) 1 10.1.84.195 > 10.1.84.196
--------------
input: out on lan0 udp 10.1.80.196,16001 10.1.84.195,16000
nomatch ip 28(20) 17 10.1.80.196,16001 > 10.1.84.195,16000
--------------
input: out on lan0 udp 10.1.80.196,16001 10.1.85.195,16000
nomatch ip 28(20) 17 10.1.80.196,16001 > 10.1.85.195,16000
--------------
input: in on lan0 udp 10.1.84.195,16000 10.1.80.196,16001
pass ip 28(20) 17 10.1.84.195,16000 > 10.1.80.196,16001
--------------
input: in on lan0 udp 10.1.85.195,16000 10.1.80.196,16001
block ip 28(20) 17 10.1.85.195,16000 > 10.1.80.196,16001
--------------
```

Each result is one of the following: `pass`, `block`, or `nomatch`. For HP-UX IPFilter, the default is `pass`. From the results you can verify that the filter should operate as expected.

More complex rulesets and sample traffic can be tested to reflect a production environment.

The ipnat Utility

Use the `ipnat` utility to view and load NAT rules. The default NAT rules file is `/etc/opt/ipf/ipnat.conf`.

Syntax

`ipnat <options> <full path name>`

Options

`-f`

Reads rules from a specified rules file.

`-l`

Views NAT rules and active mappings.

`-C`

Flushes the current ruleset.

`-F`

Removes active mappings.

`-r`

Removes rules from the NAT rules file.

Example

Enter the following command:

```
ipnat -CF -f /etc/opt/ipf/ipnat.conf
```

This command flushes any existing NAT rules and removes any active mappings, then loads the NAT rules in the `ipnat.conf` file.

Unsupported Utilities and Commands

HP does not support the following public domain IPFilter utilities and commands:

- Rule keywords
  - `fastroute`
- Commands
  - `ipscan`
  - `ipsyncs`
  - `ipsyncm`
  - `ipfs`
  - `ipsend`
  - `ipresend`
- Application proxy

6 HP-UX and IPv6 Support

This chapter describes IPv6 support in HP-UX IPFilter.

It contains the following sections:

- Product Configuration

- Product Installation and Dependencies
- Rules Configuration
- Commands
- New Features for IPv6
- Command and Configuration Examples
- Installation Details and Dependencies
- Features Not Supported with IPv6
- Key Points to Note

Using IPv6 Support in HP-UX IPFilter

IPv6 support has been added to HP-UX IPFilter. The functionality is mostly equivalent to IPv4 functionality in HP-UX IPFilter. There are some differences, which are described in this chapter.

Product Configuration

No new software modules or filesets have been introduced in the IPv6 version. The current version of HP-UX IPFilter has been enhanced to include IPv6 functionality. Filter rules configuration in IPv6 support is identical to IPv4 support rules configuration. A new file (`/etc/opt/ipf/ipf6.conf`) is provided which is read during IPFilter startup. This file is just like the one provided for IPv4 (`etc/opt/ipf/ipf.conf`).

Both of these files can be changed, if necessary, by modifying the IPF_CONF and IPF6_CONF variables in `/etc/rc.confif.d/ipfconf`.

Product Installation and Dependencies

HP-UX IPFilter IPv6 filtering functionality is dependent on the Transport patch TOUR 3.1 or later. If this dependency is not met, IPFilter will be capable of filtering only IPv4 traffic. Therefore, IPv6 traffic will not be secured.

Rules Configuration

Internally, HP-UX IPFilter maintains IPv4 and IPv6 filter rules as separate rule sets. Each requires separate configuration and administration. Any given rule will apply to either IPv4 or IPv6, but not both. These include rules which have addresses and ports specified as wildcards.

The rule `block in from any to any` will match to IPv4 traffic or IPv6 traffic. A traffic match will depend upon the command options used while configuring the rules. New command line options have been introduced which can be used to apply and operate the IPv6 rules. These options are further explained in “Commands” on page 106.

Similarly, rules cannot mix IPv4 and IPv6 addresses. For example, the following rule is not valid:

```
pass in proto tcp from 101.11.23.1 to 3ffe::2
```

Filter Rules

The syntax of basic filter rules is not changed for IPv6. The same set of keywords applies and has the same effect. For example, use the following rule to block an inbound telnet connection:

```
block in proto tcp from 3ffe::2 to 3ffe::9 port = 23
```

Protocol-Based Filtering

There are no major changes for IPv6 protocol-based filtering. Upper layer protocols can be used in the same method and TCP or UDP can be specified as for IPv4 rules. The only exception is ICMPv6, which is new protocol with IPv6.

The keyword to filter ICMPv6 is “icmpv6” or “ipv6-icmp,” which is the standard keyword specified in `/etc/protocols`. Also HP-UX IPFilter can filter any ICMPv6 message type-code pair.

IPv6 extension headers can also be filtered. This is described in “IPv6 Extension Headers” on page 109.

Stateful Filtering

TCP and UDP stateful filtering has not changed for IPv6, because these protocols have not changed. However, ICMPv6 changes are described in “Stateful ICMPv6” on page 108.

Commands

The commands used for IPv6 are similar to those used for IPv4, but new command line options have been introduced.

ipf

The `ipf` command is used to manipulate IPFilter rules. The `ipf` command with options has the capability of reading, deleting, or swapping rules.

The following command reads the rules in the *<rulefile>*, where *<rulefile>* is a file containing a list of rules, and adds them to the IPv4 ruleset:

```
ipf -f <rulefile>
```

The new `-6` option must be added if the *<rulefile>* contains IPv6 rules that must be configured:

```
ipf -6 -f <rulefile>
```

To delete all active IPv6 rules, use the following command:

```
ipf -6 -Fa
```

To selectively remove IPv6 rules, use the following command:

```
ipf -6 -r -f <rules to be deleted>
```

Most options to the `ipf` command, when prepended with the `-6` option, will affect the IPv6 rule set. The one exception is the `-s` option. The `-s` option is used to swap active and inactive rules, but does not require a `-6` option. The `ipf -s` command swaps an active ruleset with an inactive ruleset for both IPv4 and IPv6.

The following options enable you to control IPFilter processing on a given IPv6 interface.

```
-E -6 <interface name>
```

Enables IPFilter processing for traffic on a given interface.

```
-D -6 <interface name>
```

Disables IPFilter processing for traffic on a given interface.

```
-Q -6 <interface name>
```

Verifies that IPFilter processing is enabled or disabled for a given interface.

For example, `ipf -D -6 lan0` disables IPFilter processing for traffic on lan0 and `ipf -E -6 lan0` enables IPFilter processing on lan0. `ipf -Q -6 lan0` is used to verify if IPFilter processing is enabled or disabled for lan0.

ipfstat

The `ipfstat` command generates packet filter statistics and filter lists. It also uses the `-6` option for IPv6, but only for the following operations:

- `-i`—Lists IN rules
- `-o`—Lists OUT rules
- `-h`—Lists rule hits

For example, to list the IN rule hits for IPv6, use the following command:

```
ipfstat -6 -ih
```

ipmon

There are no major changes to logging for IPv6. The `ipmon` command takes the same options as before and no new options are needed to log IPv6 traffic. Log files include both IPv4 and IPv6 log records, ordered according to how IPFilter receives the traffic. The log records indicate IPv6 equivalents such as IPv6 addresses and protocol ICMPv6.

NOTE IPv4 and IPv6 log records cannot be sent to different log files.

New Features for IPv6

The following new features are supported by HP-UX IPFilter for IPv6:

- Stateful ICMP
- IPv6 extension headers
- Tunneled packets
- Fragmentation

Stateful ICMPv6

The following feature is not new, but has been enabled on IPv6. State can be retained on ICMPv6 using Request-Response ICMPv6 messages. The only message types are the echo request and echo reply.

ICMPv6 filtering

ICMPv6 filtering must be carefully configured to ensure that an IPv6 network functions properly. For example, do not block Neighbor Discovery messages (type 135 and 136). Other examples of critical ICMPv6 messages are Destination Unreachable (type 1) and Packet Too Big (type 2).

HP-UX IPFilter enables you to uniquely identify an ICMPv6 message using its type and code. A new keyword, icmpv6-type, is introduced. Use the following rule to pass ICMPv6 type 135 code 0 packets:

```
pass in quick proto icmpv6 from any to any icmpv6-type 135 code
0
```

NOTE The type and code can only be specified as a decimal number.

At minimum, the following rules must be configured:

```
pass in quick proto icmpv6 from any to any icmpv6-type 133
pass in quick proto icmpv6 from any to any icmpv6-type 134
pass in quick proto icmpv6 from any to any icmpv6-type 135
pass in quick proto icmpv6 from any to any icmpv6-type 136
pass out quick proto icmpv6 from any to any icmpv6-type 133
pass out quick proto icmpv6 from any to any icmpv6-type 134
pass out quick proto icmpv6 from any to any icmpv6-type 135
pass out quick proto icmpv6 from any to any icmpv6-type 136
```

The following is additional information about message types 133-136:

- 133—Router solicitation
- 134—Router advertisement
- 135—Neighbor solicitation
- 136—Neighbor advertisement

IPv6 Extension Headers

The following extension headers are supported with IPv6:

- Destination options header (dstopts)
- Hop-by-hop options header(hopopts)
- Mobility header (mobility)

- Routing options header (routing)
- Authentication header (ah)
- IPSec header (esp)
- IPv6 header for tunneled packets(IPv6) (ipv6)
- IPv6 fragment (frags)

Currently, filtering is available to either block or pass packets with designated extension headers. For example, to block all TCP packets with a Routing options header, use the following rule:

```
block in proto tcp from any to any with v6hdrs routing
```

To block all UDP packets with destination option and mobility headers, use the following rule:

```
block in proto udp from any to any with v6hdrs
dstopts,mobility
```

NOTE Extension headers are matched explicitly. A packet with only a routing header will not match the previous rule. Only packets with both mobility and destination option headers will match the rule.

Tunneled Packets

HP-UX IPFilter can filter the following types of tunnel packets:

- 6-in-4—Use the following rule to filter this type of tunnel packet:

  ```
  ipf -f
  block in proto 41 from any to any
  ```

- 6-in-6—Use the following rule to filter this type of tunnel packet:

  ```
  ipf -6 -f
  block in proto 41 from any to any
  ```

Fragmentation

Unlike IPv4, a fragment cache is not maintained for IPv6 fragments. It is possible to filter IPv6 fragments using the “with v6hdrs frags” keywords. Use the following rule to filter IPv6 fragmented traffic:

```
block in proto udp from any to any with v6hdrs frags
```

Command and Configuration Examples

- To configure IPv6 rules from files containing IPv6 rules:

  ```
  ipf -6 -f <ipv6 rule file>
  ```

- To flush IPv6 IN rules:

  ```
  ipf -6 -Fi
  ```

- To see rule hits on OUT rules:

  ```
  ipfstat -6 -oh
  ```

- To disable IPv6 filtering on LAN0 inet6:

  ```
  ipf -6 -D lan0
  ```

Installation Details and Dependencies

HP-UX IPFilter depends on TOUR 3.1 to provide IPv6 functionality. IPFilter installs successfully without TOUR 3.1, but IPv6 network interfaces are not detected.

Features Not Supported with IPv6

The following features are not supported with IPv6:

- Dynamic Connection Allocation (DCA) (the configuration of the IPv6 keep limit rules is not allowed.)
- IPFilter NAT functionality and the associated commands and utilities
- The `ipftest` utility
- RPC scripts
- IPFilter group rules

Key Points to Note

- If the `-6` option is not specified in the `ipf` and `ipfstat` commands, the operation is applied to IPv4 rules.
- IPv6 filtering is enabled only if IPv6 interfaces are configured.
- The following separate files for IPv4 and IPv6 are loaded during boot:
  - IPv4: `opt/ipf/ipf.conf`
  - IPv6: `opt/ipf/ipf6.conf`

    Rules can be loaded from any other file, but IPv4 and IPv6 rules should not be loaded from the same file.

7 HP-UX IPFilter and FTP

This chapter describes how to filter FTP services. It contains the following sections:

- FTP Basics
- WU-FTPD on HP-UX
- Running an FTP Server
- Running an FTP Client

CAUTION **NAT and FTP are incompatible. If you are using FTP on your IPFilter system, do not use NAT rules.**

FTP Basics

The File Transfer Protocol (FTP) is a user-level protocol for transferring files between host computers.

An FTP session involves two separate connections:

- Control connection

  1. The server listens for client connections on port 21.
  2. The client opens a connection to the server port 21 on a client port above 1023.
  3. The client uses this connection to send commands to, and receive replies from, the server.

     This connection lasts through the FTP session.

- Data connection

  The data connection is used for transferring data between the client and server. A new data connection is opened for each FTP command. The way the data connection is created depends on the type of FTP session—active or passive.

In active FTP, the client actively opens a connection to the FTP server at port 21. It uses a port number > 1023 as its port for the control connection. The client then opens a new port (passive open) as its data port and sends this port number across to the server using the PORT command. The server then opens a data connection (active open) to the data port specified in the PORT command of the client. The server uses port 20 as its data connection port.

In passive FTP, the control connection is established the same as it is in active FTP. In passive FTP, to establish a data connection the server opens an arbitrary data port >1023. It uses the PASV command to send the data port number to the client. The client connects to the port specified by the PASV command and uses a different port >1023 as its data port.

WU-FTPD on HP-UX

The HP implementation of the FTP daemon for HP-UX 11i core networking is based on the WU-FTPD daemon, version 2.4. Additional security correction has been added to WU-FTPD 2.6.1. HP recommends upgrading to WU-FTPD 2.6.1 for enhanced security.

For systems on HP-UX 11.0, you can upgrade to WU-FTPD 2.6.1 from either the legacy FTP version that is delivered with the core networking products on 11.0, or from WU-FTPD 2.4, which has been made available as the patch PHNE_21936.

WU-FTPD 2.6.1 is downloadable from the HP Software Depot for systems running HP-UX 11.0 or HP-UX 11i v1. The URL is http://www.software.hp.com/cgi-bin/swdepot_parser.cgi/cgi/displayProductInfo.pl?productNumber=3DWUFTPD26.

WU-FTPD 2.6.1 is a core product on HP-UX 11i v2.

Running an FTP Server

This section describes active FTP and passive FTP server setup.

Active FTP

FTP Server	Direction of Connection Initiated	FTP Client
port 21 control port	<----------	any port 1024 or higher
port 20 data port	---------->	any port 1024 or higher

On an FTP server using active FTP, configure IPFilter rules to allow control connections in and data connections out.

For example:

```
pass in quick proto tcp from any port > 1023 to <server-ip>
port = 21 flags S  keep state
pass out quick proto tcp from any port = 20 to any  port > 1023
flags S keep state
block in  from any to any
block out from any to any
```

Passive FTP

FTP Server	Direction of Connection Initiated	FTP Client
port 21 control port	<----------	any port 1024 or higher
any port 1024 or higher data port	<----------	any port 1024 or higher

To use IPFilter to protect passive FTP sessions, you must limit the port range your system can use for FTP access. For example, you can allocate ports 15001-15500 as FTP ports and only open up that range of your firewall. In WU-FTPD, you use the `passive ports` directive in the `/etc/ftpaccess` configuration file to designate the ports, as follows:

```
passive ports <server-ip> 15001 15500
```

See the *ftpaccess* (4) manpage for details on WU-FTPD configuration.

Configure the following IPFilter rules to let the passive FTP traffic pass:

```
pass in quick proto tcp from any port > 1023 to <server-ip>
port = 21 flags S keep state
pass in quick proto tcp from any port > 1023 to <server-ip>
port 15000 ><15501 flags S keep state
block in  from any to any
block out from any to any
```

Running an FTP Client

As with FTP servers, there are two types of FTP client transfers, active and passive.

Active FTP

FTP Server	Direction of Connection Initiated	FTP Client
port 21 control port	<----------	any port 1024 or higher
port 20 data port	---------->	any port 1024 or higher

To let an FTP client open an active FTP session, configure IPFilter rules to allow control connections out and data connections in.

```
pass out quick proto tcp from <client-ip> port > 1023 to any
port = 21 flags S  keep state
pass in quick proto tcp from any port 20 to <client-ip> port >
1023 flags S  keep state
block in  from any to any
block out from any to any
```

NOTE FTP Proxy is not supported by HP. For a complete list of unsupported utilities and commands, see “Unsupported Utilities and Commands” on page 102.

Passive FTP

FTP Server	Direction of Connection Initiated	FTP Client
port 21 control port	<----------	any port 1024 or higher
any port 1024 or higher data port	<----------	any port 1024 or higher

To let an FTP client open a passive FTP session, configure IPFilter to allow both the control and data connections out.

Use the following ruleset for client-side, passive FTP:

```
pass out quick proto tcp from <client-ip> port > 1023 to any
port = 21 flags S  keep state
pass out quick proto tcp from <client-ip> port> 1023 to any
port > 1023 flags S keep state
block in  from any to any
block out from any to any
```

TIP For stronger security, configure IPFilter to allow only active FTP connections from FTP servers.

8 HP-UX IPFilter and RPC

This chapter describes the use of RPC with IPFilter. It contains the following sections:

- Introduction
- Quick Start Information
- Configuration Files

Introduction

The script information and configuration files in this chapter are designed to allow a system running IPFilter/9000 to run server processes that use the Remote Procedure Call (RPC) mechanism.

The purpose is to automate the construction of appropriate IPFilter rules for RPC server processes that do not use a fixed port number, but register their port numbers with `rpcbind` instead. The services in this example are for an NFS server, but the concept should work for any RPC server application.

NOTE

These set of files and scripts serve as basic building blocks for use at startup time. All files are installed in `/etc/opt/ipf/rpc.ipf`. The configuration files must be present in the appropriate directories for the scripts to work correctly.

Quick Start Information

To use RPC with IPFilter:

1. Copy the sample file to `/etc/rc.config.d/rpc_ipfconf`

   ```
   cp rpc_ipfconf.sample /etc/rc.config.d/rpc_ipfconf
   ```

   Edit the file as needed.

2. Create the `rpc.ipf` directory and change to that directory.

   ```
   mkdir /etc/opt/ipf/rpc.ipf
   cd /etc/opt/ipf/rpc.ipf
   ```

3. Create an empty RPC rules file.

   ```
   touch /etc/opt/ipf/rpc.ipf/rpc.rules
   ```

4. Start the script configuration.

   ```
   ./rpc.ipfboot start
   ```

Configuration Files

Rules Files

This section gives details on the two rules files that contain the IPFilter rules. The two rules files are:

- The IPFilter rules file specified in `$IPF_CONF` in `/etc/rc.config.d/ipfconf`
- The IPFilter RPC rules file specified in `$RPC_RULES_FILE` specified in `/etc/rc.config.d/rpc_ipfconf`

NOTE See the following section for a description of `/etc/rc.config.d/rpc_ipfconf`. A sample file is also provided.

To incorporate the dynamic ports used by the NFS processes, the administrator should decide the position from which RPC rule should be configured by setting `RPC_RULE_POSITION` to the desired value. For example:

```
RPC_RULE_POSITION=5
```

The RPC rules will then be added from the 5th position onwards. If there are 10 RPC rules, they will be inserted at positions 5 to 14. The position must be chosen carefully. If there are only two rules present, then `RPC_RULE_POSITION` must be 1,2 or 3 [`RPC_RULE_POSITION = <current # of rules>`]. The Original rules file specified in `/etc/rc.config.d/ipfconf` containing other rules is not modified.

By default, all RPC rules are configured as the first rules, for example, `RPC_RULE_POSITION=1`. The RPC rules are well defined in terms of IP addresses and ports and will have unique matches and, since they are `quick` rules, they should be at top.

RPC Rules Configuration File

This file specifies details based on which IPFilter RPC rules will be generated. `/etc/opt/ipf/rpc.ipf/rpc_ipfconf.sample` is provided as an example.

The /etc/opt/ipf/rpc.ipf/rpc_ipfconf file contains the client list and program list. The sample file grants access to the program numbers listed from the IP addresses and IP subnets listed in the client list.

The example shown in the sample file lists the program numbers used by an NFS server, rpc.mountd, rpc.statd, rpc.lockd, and nfsd. This file also has the following declared:

- ADD_RPC_IPFILTER_RULES=1

  Set this to 1 to configure RPC IPFilter rules.

- RPC_RULE_POSITION=1

  Must be 1 or greater, as noted in the previous section.

- RPC_RULES_FILE=./rpc.rules

  This is the path to the RPC rule file. Rules to be added or deleted.

9 HP-UX IPFilter and IPSec

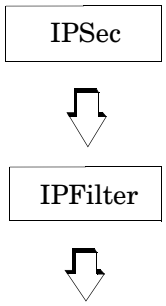
This chapter describes how HP-UX IPFilter and HP-UX IPSec work together. It contains the following sections:

- IPFilter and IPSec Basics
- IPSec UDP Negotiation
- When Traffic Appears to Be Blocked
- Allowing Protocol 50 and Protocol 51 Traffic
- IPSec Gateways

IPFilter and IPSec Basics

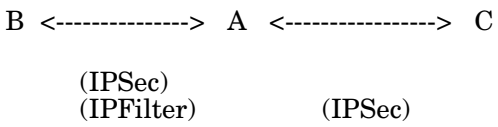
IPSec and IPFilter will not panic or corrupt each other. However, there are situations in which one product might block traffic for the other. The following figure shows the positions of IPFilter and IPSec in the network stack:

Figure 9-1 IPFilter and IPSec



IPFilter, which is below IPSec in the networking stack, filters network packets before they reach IPSec. You can have both IPFilter and IPSec configured and running on a machine without them negatively affecting each other.

Figure 9-2 Scenario One



In Scenario One, you have IPFilter and IPSec on machine A with IPFilter blocking packets from machine B and IPSec encrypting packets from machine C. When a packet arrives at machine A, IPFilter checks to see if it is from machine B, and, if so, blocks the packet. If not, the packet continues up the stack to IPSec. IPSec checks to see if it is from machine C. If so, the packet arrives encrypted.

No overlap is in the configurations of IPFilter and IPSec in this network topology, so there are no conflicts in Scenario One.

CAUTION HP-UX IPSec does not support NAT traversal. If you are using HP-UX IPFilter with HP-UX IPSec, do not use NAT functionality. However, it is possible that IPFilter and NAT can be used in network configurations containing other vendors' IPSec products that do support NAT traversal.

IPSec UDP Negotiation

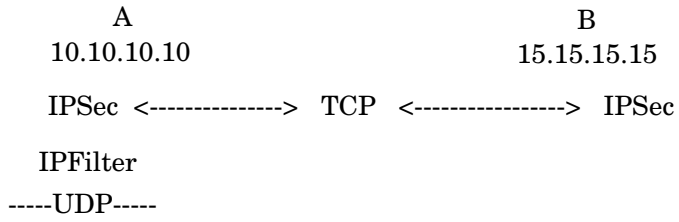
You can configure IPSec and IPFilter so that there is some overlap in the configurations. However, you must be sure the overlapping configurations do not block each other.

IPSec negotiates between two machines on a connection using the UDP protocol from port 500 and port 4500 if IPSec NAT traversal is used.

If the IPFilter configuration is so broad that it is blocking all UDP traffic, then IPSec cannot complete negotiations. When an IPSec negotiation is not completed, the encrypted packets are not received. If this happens, you will see an IPSec error on the initiating side of “MM negotiation timeout.”

To let IPSec complete negotiations, configure IPFilter to let the IPSec negotiation packets through.

Figure 9-3 Scenario Two



In Scenario Two, IPFilter is configured to block UDP traffic on machine A, you want all TCP traffic to pass through, and, from machine B on the network, you want all TCP traffic encrypted. Machine A has IP address 10.10.10.10 and machine B has IP address 15.15.15.15.

As the TCP traffic with machine B must be encrypted, you configure IPSec on both machines using IPSec Manager. To do so, use the IP addresses to specify that the TCP traffic is to be encrypted.

When TCP traffic is initiated from A to B or from B to A, IPSec on both machines communicates through a UDP/500 connection. You must configure IPFilter on machine A to let this traffic through. To do so, add the following rules to your configuration:

```
pass in quick proto UDP from 15.15.15.15 port = 500 to
10.10.10.10 port = 500
pass out quick proto UDP from 10.10.10.10 port = 500 to
15.15.15.15 port = 500
block in proto UDP
block out proto UDP
```

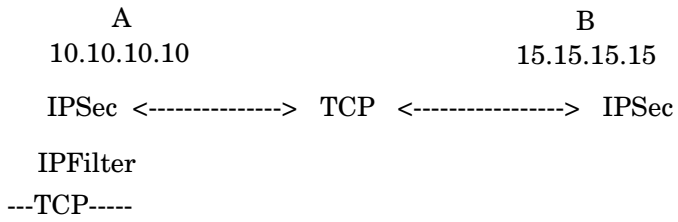
These rules let IPSec traffic pass correctly.

NOTE You must configure IPFilter to pass traffic both in and out on UDP port 500 for IPSec to work properly. If IPFilter is used with IPSec requiring the NAT traversal function, UDP port 4500 must be set to `pass` for in and out traffic.

When Traffic Appears to Be Blocked

In the following scenario there is overlap in the configurations of IPFilter and IPSec. To get this negotiation through, you must configure IPFilter rules to let TCP traffic through.

Figure 9-4 Scenario Three



In Scenario Three, IPSec is configured to encrypt TCP traffic between machine A and machine B and IPFilter is configured to block all TCP traffic with the following rules:

```
block in proto TCP
block out proto TCP
```

Allowing Protocol 50 and Protocol 51 Traffic

IPSec uses Encapsulating Security Payload (ESP) to provide data confidentiality and Authentication Header (AH) to provide data integrity at the IP layer. Depending on a user's IPSec traffic policy configuration, IPSec inserts ESP, AH, or both as protocol headers into an IP datagram that immediately follows an IP header. The protocol field of that IP header will be 50 (esp) or 51 (ah) to indicate the next protocol.

Figure 9-5 Packet with Unencrypted TCP Data

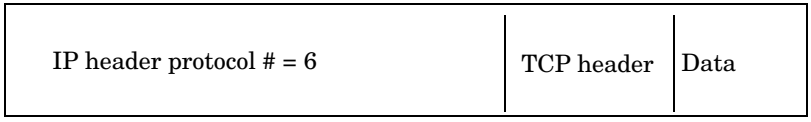
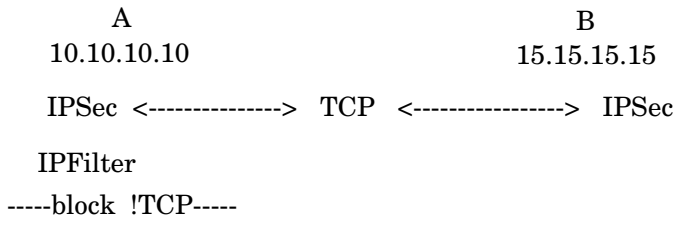


Figure 9-6 Packet with IPSec-Encrypted TCP Data

IPFilter never sees the TCP packets between machine A and machine B with a protocol number of 6. These packets are encrypted (or wrapped) in a packet that has a protocol number of 50. If you configure IPFilter to block packets with protocol number 6, it lets protocol number 50 pass through. IPSec takes apart the packet and unencrypt the TCP data.

If the IPFilter configuration is so broad that it blocks protocol 50 or protocol 51 traffic, then IPSec traffic will not get through.

Figure 9-7 Scenario Four



In Scenario Four, IPSec is configured to encrypt TCP traffic between the two machines and IPFilter is configured to block non-TCP traffic. IPFilter rules are also configured to let UDP/500 traffic pass on machine B.

```
# IPSec hole with machine B
pass in quick proto UDP from 15.15.15.15 port 500 to
10.10.10.10 port = 500
pass out quick proto UDP from 10.10.10.10 port 500 to
15.15.15.15 port = 500
# Let in encrypted IPSec traffic
pass in quick proto 50 from 15.15.15.15 to 10.10.10.10
pass out quick proto 50 from 10.10.10.10 to 15.15.15.15
# Allow TCP traffic to/from anywhere
pass in quick proto TCP
pass out quick proto TCP
# Block all other traffic to/from anywhere
block in from any to any
block out from any to any
```

NOTE If IPSec is configured to use AH rather than ESP, you must configure IPFilter to let protocol 51 traffic pass. If IPSec uses nested AH and ESP, IPFilter can be configured to let only protocol 51 (ah) traffic pass.

IPSec Gateways

You can configure IPSec to encrypt and authenticate traffic to a gateway between two end hosts. A configuration that encrypts IPSec packets to a gateway is called an IPSec tunnel.

IPFilter can coexist with IPSec tunnels without conflict. However, you must configure IPFilter to allow IPSec traffic with the gateway instead of the end node. The IPFilter rules for the UDP/500 and protocol 50/51 traffic must be passed to and from the gateway IP address rather than the end node IP address.

10 HP-UX IPFilter and Serviceguard

This chapter describes configuration procedures for HP-UX IPFilter used in a Serviceguard environment.

It contains the following sections for using HP-UX IPFilter with Serviceguard:

- Local Failover
- Remote Failover
  - — Filtering on a Package IP Address
  - — Mandatory Rules
- DCA Remote Failover

Using HP-UX IPFilter with Serviceguard

HP-UX IPFilter supports local failover in a Serviceguard environment.

CAUTION NAT functionality is not supported with Serviceguard.

Local Failover

NOTE See the Serviceguard documentation for information on configuring a local failover system in Serviceguard.

IPFilter local failover is transparent to users. Network sessions are not disrupted during failover or failback.

You do not need to configure any additional rules in IPFilter. When an interface fails over, the HP-UX IPFilter rules that specify interface names are automatically changed.

For example, a node in a Serviceguard cluster has a primary interface named `lan0` and a standby interface named `lan1`. The following rule is configured for `lan0`:

```
pass in on lan0 proto tcp from any to any port = telnet
```

Upon failover, the rule is automatically modified to:

```
pass in on lan1 proto tcp from any to any port = telnet
```

The rule will be changed back automatically on failback.

All rules that filter on interface names are changed at failover and failback in both the active ruleset and the inactive ruleset. In addition, logging reflects the changes; the standby interface name will appear in logs and reports when it is in use.

Remote Failover

HP-UX IPFilter is a system firewall and as such should be installed on end systems. Connections to an IPFilter system that are lost during a remote failover must be reinitiated.

Install and configure HP-UX IPFilter on each node of a Serviceguard cluster that must be protected. The IPFilter configuration for the primary node might be different from the configuration for the backup nodes.

For example, the backup node might be multi-homed and require a different rule set. Also, rules could be configured to filter on the static IP address of the receiving node that would be different for each node in the cluster. Rules that filter on interface names will also be different on different nodes in a cluster.

Filtering on a Package IP Address

HP-UX IPFilter can filter on a package IP address. The package IP address is an IP address that corresponds to a logical network interface.

For example, a telnet connection is made to the primary cluster node with a package IP address of 17.13.24.105. You want to configure IPFilter to let telnet traffic through. Configure the following rule for incoming telnet connections made to the telnet package:

```
pass in proto tcp from any to 17.13.24.105 flags S keep state
```

You can replace 17.13.24.105 with the package name in this rule if the package has been configured properly and has a DNS entry.

Configure this rule on the backup nodes as well. This ensures that when the telnet package fails to a backup, there is a rule on the backup that lets telnet connections pass through as required.

This type of configuration can be used with any package.

Mandatory Rules

Each node in a Serviceguard cluster has specific rules that must be configured. These rules ensure that:

- Normal remote failovers are not disrupted.
- No false remote failovers occur because traffic is blocked by IPFilter that should not be blocked.

The classes of mandatory rules cover:

- Intra-Cluster Communication
- Quorum Server
- Remote Command Execution
- Cluster Object Manager
- Serviceguard Manager

The following services should not be blocked:

```
hacl-qs  1238/tcp   # High Availability (HA) Quorum Server
clvm-cfg 1476/tcp   # HA LVM configuration
hacl-hb  5300/tcp   # High Availability (HA) Cluster heartbeat
hacl-hb  5300/udp   # High Availability (HA) Cluster heartbeat
hacl-gs  5301/tcp   # HA Cluster General Services
hacl-cfg 5302/tcp   # HA Cluster TCP configuration
hacl-cfg 5302/udp   # HA Cluster UDP configuration
hacl-probe 5303/tcp # HA Cluster TCP probe
hacl-probe 5303/udp # HA Cluster UDP probe
hacl-local 5304/tcp # HA Cluster commands
hacl-test  5305/tcp # HA Cluster test
hacl-dlm   5408/tcp # HA Cluster distributed lock manager
```

NOTE

This list of HA services is not exhaustive. In addition, Serviceguard also uses dynamic ports (typically in the 49152–65535 range) for some cluster services. If you have adjusted the dynamic port range using kernel tunable parameters, alter your rules accordingly.

This list does not include all HA applications (such as Continental Cluster). New HA applications might be developed that use port numbers different from those previously listed. You must add new rules as appropriate to ensure that all HA applications run properly. The current list of ports used by Serviceguard are documented in the *Serviceguard Release Notes*.

Intra-Cluster Communication To ensure proper operation of your Serviceguard cluster, each of the configured Serviceguard heartbeat subnets must allow intra-cluster communication. The following rules must be applied to each subnet.

For a simplified HP-UX IPFilter configuration, use the following rules:

```
pass in quick from <clusternodes> to any

pass out quick from any to <clusternodes>
```

For more restrictive HP-UX IPFilter configurations, use the following rules to allow only the required cluster services to pass through:

```
pass in quick proto tcp from <clusternodes> to <clusternodes>
port 5299 >< 5305 flags S keep state

pass in quick proto udp from <clusternodes> to <clusternodes>
port = 5300 keep state

pass in quick proto udp from <clusternodes> to <clusternodes>
port = 5302 keep state

pass in quick proto tcp from <clusternodes> to <clusternodes>
port = 5408 flags S keep state

pass in quick proto tcp from <clusternodes> to <clusternodes>
port 49151><65536 keep state

pass in quick proto udp from <clusternodes> to <clusternodes>
port 49151><65536 keep state

pass out quick proto tcp from <clusternodes> to <clusternodes>
port 5299 >< 5305 flags S keep state

pass out quick proto udp from <clusternodes> to <clusternodes>
port = 5300 keep state

pass out quick proto udp from <clusternodes> to <clusternodes>
port = 5302 keep state

pass out quick proto tcp from <clusternodes> to <clusternodes>
port = 5408 flags S keep state

pass out quick proto tcp from <clusternodes> to <clusternodes>
port 49151><65536 keep state

pass out quick proto udp from <clusternodes> to <clusternodes>
port 49151><65536 keep state

pass in quick proto udp from <clusternodes> to <clusternodes>
port = 9 keep state

pass out quick proto udp from <clusternodes> to <clusternodes>
port = 9 keep state
```

In the previous set of rules, *<clusternodes>* are all nodes in the cluster, including the local node.

Running the `cmscancl` command requires the “shell” port be open.

Quorum Server If your Serviceguard configuration uses a Quorum Server, each node in the cluster must have the following rule configured:

```
pass out quick proto tcp from <clusternodes> to <quorumserver>
port = 1238 flags S keep state
```

Any node providing Quorum Service for another cluster must have the following rule configured:

```
pass in quick proto tcp from <clusternodes> to <quorumserver>
port = 1238 flags S keep state
```

In the previous set of rules, *<clusternodes>* are all nodes in the cluster utilizing the Quorum Service and *<quorumserver>* is the IP address used to access the Serviceguard Quorum Service software.

Remote Command Execution If you want nodes outside the cluster to execute Serviceguard commands, as specified in the `etc/cmcluster/cmclnodelist` file, additional rules are required.

Each node in the cluster must have the following rules configured:

```
pass in quick proto tcp from <remotenodes> to <clusternodes>
port = 5302 flags S keep state

pass in quick proto udp from <remotenodes> to <clusternodes>
port = 5302 keep state

pass out quick proto tcp from <clusternodes> to <remote node
name> port 49151><65536 keep state

pass out quick proto udp from <clusternodes> to <remote node
name> port 49151><65536 keep state
```

Each remote node must have the following rules configured:

```
pass in quick proto tcp from <clusternodes> to <remote node
name> port 49151 >< 65536 keep state

pass in quick proto udp from <clusternodes> to <remote node
name> port 49151 >< 65536 keep state

pass out quick proto tcp from <remotenodes> to <clusternodes>
port = 5302 flags S keep state

pass out quick proto udp from <remotenodes> to <clusternodes>
port = 5302 keep state
```

In the previous set of rules, *<clusternodes>* are all nodes in the cluster, *<remote node name>* is the specific remote node, and *<remotenodes>* are all other nodes outside the cluster that are designated in the `cmclnodelist` file for remote command access.

Running the `cmscancl` command requires the "shell" port be open.

Cluster Object Manager If you are using a Cluster Object Manager (COM) on a node outside of the cluster to provide connections to Serviceguard Manager clients, each node in the cluster must have the following rules configured:

```
pass in quick proto tcp from <comnode> to <clusternodes> port =
5302 flags S keep state

pass in quick proto udp from <comnode> to <clusternodes> port =
5302 keep state

pass out quick proto tcp from <clusternodes> to <comnode> port
49151 >< 65536 keep state

pass out quick proto udp from <clusternodes> to <comnode> port
49151 >< 65536 keep state
```

The node running COM must have the following rules configured:

```
pass in quick proto tcp from <comclient> to <comnode> port =
5303 flags S keep state

pass in quick proto tcp from <clusternodes> to <comnode> port
49151 >< 65536 keep state

pass in quick proto udp from <clusternodes> to <comnode> port
49151 >< 65536 keep state

pass out quick proto tcp from <comnode> to <clusternodes> port
= 5302 flags S keep state

pass out quick proto udp from <comnode> to <clusternodes> port
= 5302 keep state
```

Each COM client must have the following rules configured:

```
pass out quick proto tcp from <comclient> to <comnode> port =
5303 flags S keep state
```

In the previous set of rules, *<clusternodes>* are all nodes in the cluster, *<comclient>* are nodes that are clients of COM for Serviceguard Manager or Continental Clusters products, and *<comnode>* is the node running the COM.

Serviceguard Manager If you are using the station-management version of Serviceguard Manager, you must configure rules to let SNMP traffic pass between all nodes in the cluster and the Serviceguard Manager node.

Each cluster node must have the following rules configured:

```
pass in quick proto udp from <SGMgr node> to <clusternodes>
port = 161 keep state

pass out quick proto udp from <clusternodes> to <SGMgr node>
port = 162 keep state
```

Each Serviceguard Manager node must have the following rules configured:

```
pass out quick proto udp from <SGMgr node> to <clusternodes>
port = 161 keep state

pass in quick proto udp from <clusternodes> to <SGMgr node>
port = 162 keep state
```

In the previous set of rules, *<clusternodes>* are all nodes in the cluster, including the local node, and *<SGMgr node>* is the node or nodes running Serviceguard Manager.

NOTE The previous sections are examples and meant to serve as guidelines. You might need to modify them to work with your network configuration and the applications you use. Specific applications used within the Serviceguard cluster might require additional ports to be opened.

DCA Remote Failover

Normally, IPFilter `keep state` rules are configured with the `flags S` parameter. This parameter instructs IPFilter to create a TCP state entry only when a SYN packet is parsed.

To enable transparent failover between IPFilter DCA nodes, do not use `flags S` with `keep limit` rules. If incoming TCP/IP traffic is switched from the active to the standby node, DCA can rebuild the previous IPFilter state table and IPFilter DCA limit tables from the data stream. Without `flags S` in the `keep limit` rule, IPFilter creates a new state

entry from any TCP/IP packet, not just a SYN packet. A limit table entry is created. Any new connections that exceed the connection limit are rejected.

After the state table entry is created for a particular IP address source/destination and TCP port source/destination 4-tuple, further packets of this connection are processed in the state table entry. These packets are not processed by the rules' table.

For example, when Serviceguard detects that the primary IPFilter DCA gateway has failed, the IP addresses of the primary systems are switched to the standby DCA system. The standby system contains the same set of configured rules as the primary system. Therefore, the standby system can completely rebuild the TCP state tables and limit entries that were previously on the primary system.

If a client has active connection to an IPFilter system and is attempting to make new connections when Serviceguard fails over, the new connections replace the existing connections in the limit table entry for the client only if the established connections are not generating traffic.

A HP-UX IPFilter Configuration Examples

This appendix provides IPFilter configuration examples. These examples are also included in the `/opt/ipf/examples` directory with HP-UX

IPFilter. You can take useful rules that you find in these examples and copy them into `/etc/opt/ipf/ipf.conf`, which is your HP-UX IPFilter configuration file.

These files are taken from the files provided with the open source IPFilter product.

BASIC_1.FW

```
#!/sbin/ipf -f -
#

# SAMPLE: RESTRICTIVE FILTER RULES
#
# ppp0 - (external) PPP connection to ISP, address a.b.c.d/32
#
# lan0 - (internal) network interface, address w.x.y.z/32
#
# This file contains the basic rules needed to construct a
# firewall for the above connections.
#
#--------------------------------------------------------
# Block short packets which are packets fragmented too short to
# be real packets.
block in log quick all with short
#--------------------------------------------------------
# Group setup.
# ============
# By default, block and log all packets.  This may result in
# too much information to be logged (especially for lan0)
# and needs to be further refined.
#
block in log on ppp0 all head 100
block in log proto tcp all flags S/SA head 101 group 100
block out log on ppp0 all head 150
block in log on lan0 from w.x.y.z/24 to any head 200
block in log proto tcp all flags S/SA head 201 group 200
block in log proto udp all head 202 group 200
block out log on lan0 all head 250
#--------------------------------------------------------
# Localhost packets.
# ==================
# Packets going in/out of network interfaces that aren't on the
# loopback interface should *NOT* exist.
block in log quick from 127.0.0.0/8 to any group 100
block in log quick from any to 127.0.0.0/8 group 100
block in log quick from 127.0.0.0/8 to any group 200
block in log quick from any to 127.0.0.0/8 group 200
#--------------------------------------------------------
# Invalid Internet packets.
# =========================
```

```
#
# Deny reserved addresses.
#
block in log quick from 10.0.0.0/8 to any group 100
block in log quick from 192.168.0.0/16 to any group 100
block in log quick from 172.16.0.0/12 to any group 100
#
# Prevent IP spoofing.
#
block in log quick from a.b.c.d/24 to any group 100
#
#--------------------------------------------------------
# Allow outgoing DNS requests (no named on firewall)
#
pass in quick proto udp from any to any port = 53 keep state
group 202
#
# If you are running named on the firewall and all internal
# hosts talk to it,use the following:
#
pass in quick proto udp from any to w.x.y.z/32 port = 53 keep
state group 202
pass out quick on ppp0 proto udp from a.b.c.d/32 to any port =
53 keep state
#
# Allow outgoing FTP from any internal host to any external FTP
# server.
#
pass in quick proto tcp from any to any port = ftp keep state
group 201
pass in quick proto tcp from any to any port = ftp-data keep
state group 201
pass in quick proto tcp from any port = ftp-data to any port >
1023 keep state group 101
#
# Allow NTP from any internal host to any external NTP server.
#
pass in quick proto udp from any to any port = ntp keep state
group 202
#
# Allow outgoing connections: SSH, TELNET, WWW
#
pass in quick proto tcp from any to any port = 22 keep state
group 201
pass in quick proto tcp from any to any port = telnet keep
state group 201
```

```
pass in quick proto tcp from any to any port = www keep state
group 201
#
#-------------------------------------------------------
block in log proto tcp from any to a.b.c.d/32 flags S/SA head
110 group 100
#
# Allow the following incoming packets types to the external
# firewall interface: mail, WWW, DNS
pass in log quick proto tcp from any to any port = smtp keep
state group 110
pass in log quick proto tcp from any to any port = www keep
state group 110
pass in log quick proto tcp from any to any port = 53 keep
state group 110
pass in log quick proto udp from any to any port = 53 keep
state group 100
#-------------------------------------------------------
# Log these:
# ==========
# * Return RST packets for invalid SYN packets to help the
#other end close
block return-rst in log proto tcp from any to any flags S/SA
group 100
# * Return ICMP error packets for invalid UDP packets
block return-icmp(net-unr) in proto udp all group 100
```

BASIC_2.FW

```
# SAMPLE: PERMISSIVE FILTER RULES
#
# ppp0 - (external) PPP connection to ISP, address a.b.c.d/32
#
# lan0 - (internal) network interface, address w.x.y.z/32
#
# This file contains the basic rules needed to construct a
# firewall for the above situation.
#
#-------------------------------------------------------
# Short packets which are packets fragmented too short to be
# real packets.
block in log quick all with short
#-------------------------------------------------------
# Group setup.
# ============
# By default, block and log all packets.  This may result in
# too much information to be logged (especially for lan0) and
# the rules needs to be further refined.
#
block in log on ppp0 all head 100
block out log on ppp0 all head 150
block in log on lan0 from w.x.y.z/24 to any head 200
block out log on lan0 all head 250
#-------------------------------------------------------
# Invalid Internet packets.
# =========================
#
# Deny reserved addresses.
#
block in log quick from 10.0.0.0/8 to any group 100
block in log quick from 192.168.0.0/16 to any group 100
block in log quick from 172.16.0.0/12 to any group 100
#
# Prevent IP spoofing.
#
block in log quick from a.b.c.d/24 to any group 100
#
#-------------------------------------------------------
# Localhost packets.
# ==================
# packets going in/out of network interfaces that aren't on the
```

```
# loopbackinterface should *NOT* exist
block in log quick from 127.0.0.0/8 to any group 100
block in log quick from any to 127.0.0.0/8 group 100
block in log quick from 127.0.0.0/8 to any group 200
block in log quick from any to 127.0.0.0/8 group 200
#-------------------------------------------------------
# Allow any communication between the inside network and the
# outside only.
#
# Allow all outgoing connections (SSH, TELNET, FTP, WWW,
# gopher, etc)
#
pass in log quick proto tcp all flags S/SA keep state group 200
#
# Support all UDP 'connections' initiated from inside.
#
# Allow ping out
#
pass in log quick proto icmp all keep state group 200
#-------------------------------------------------------
# Log these:
# ==========
# * return RST packets for invalid SYN packets to help the
# other end close
block return-rst in log proto tcp from any to any flags S/SA
group 100
# * return ICMP error packets for invalid UDP packets
block -icmp(net-unr) in proto udp all group 100
```

example.1

```
#
# block all incoming TCP packets on lan0 from host 10.1.1.1 to
# any destination.
#
block in on lan0 proto tcp from 10.1.1.1/32 to any
```

example.2

```
#
# block all outgoing TCP packets on lan0 from any host to port
# 23 of host 10.1.1.2
#
block out on lan0 proto tcp from any to 10.1.1.3/32 port = 23
```

example.3

```
# block all inbound packets.
#
block in from any to any
#
# pass through packets to and from localhost.
#
pass in from 127.0.0.1/32 to 127.0.0.1/32
#
# allow a variety of individual hosts to send any type of IP
# packet to any other host.
#
pass in from 10.1.3.1/32 to any
pass in from 10.1.3.2/32 to any
pass in from 10.1.3.3/32 to any
pass in from 10.1.3.4/32 to any
pass in from 10.1.3.5/32 to any
pass in from 10.1.0.13/32 to any
pass in from 10.1.1.1/32 to any
pass in from 10.1.2.1/32 to any
#
#
# block all outbound packets.
#
block out from any to any
#
# allow any packets destined for localhost out.
#
pass out from any to 127.0.0.1/32
#
# allow any host to send any IP packet out to a limited number
# of hosts.
#
pass out from any to 10.1.3.1/32
pass out from any to 10.1.3.2/32
pass out from any to 10.1.3.3/32
pass out from any to 10.1.3.4/32
pass out from any to 10.1.3.5/32
pass out from any to 10.1.0.13/32
pass out from any to 10.1.1.1/32
pass out from any to 10.1.2.1/32
```

example.4

```
#
# block all ICMP packets.
#
block in proto icmp from any to any
#
```

example.5

```
#
# test ruleset
#
# allow packets coming from foo to bar through.
#
pass in from 10.1.1.2 to 10.2.1.1
#
# allow any TCP packets from the same subnet as foo is on
# through to host 10.1.1.2 if they are destined for port 6667.
#
pass in proto tcp from 10.2.2.2/24 to 10.1.1.2/32 port = 6667
#
# allow in UDP packets that are NOT from port 53 and are
# destined for localhost
#
pass in proto udp from 10.2.2.2 port != 53 to localhost
#
# block all ICMP unreachables.
#
block in proto icmp from any to any icmp-type unreach
#
# allow packets through that have a non-standard IP header
# length (ie there are IP options such as source-routing
# present).
#
pass in from any to any with ipopts
#
```

example.6

```
#
# block all TCP packets with only the SYN flag set (this is the
# first packet sent to establish a connection) out of the
# SYN-ACK pair.
#
block in proto tcp from any to any flags S/SA
```

example.7

```
# block all ICMP packets.
#
block in proto icmp all
#
# allow in ICMP echos and echo-replies.
#
pass in on lan1 proto icmp from any to any icmp-type echo
pass in on lan1 proto icmp from any to any icmp-type echorep
#
# block all ICMP destination unreachable packets which are
# port-unreachables
#
block in on lan1 proto icmp from any to any icmp-type unreach
code 3
```

example.8

```
#
# block all incoming TCP connections but send back a TCP-RST
# for ones to the ident port
#
block in proto tcp from any to any flags S/SA
block return-rst in quick proto tcp from any to any port = 113
flags S/SA
#
# block all inbound UDP packets and send back an ICMP error.
#
block return-icmp in proto udp from any to any
```

example.9

```
# drop all packets without IP security options
#
block in all
pass in all with opt sec
#
# only allow packets in and out on lan0 which are top secret
#
block out on lan0 all
pass out on lan0 all with opt sec-class topsecret
block in on lan0 all
pass in on lan0 all with opt sec-class topsecret
```

example.10

```
#
# pass ack packets (ie established connection)
#
pass in proto tcp from 10.1.0.0/16 port = 23 to 10.2.0.0/16 ...
     flags A/A
pass out proto tcp from 10.1.0.0/16 port = 23 to 10.2.0.0/16...
     flags A/A
#
# block incoming connection requests to my internal network
# from the internet.
#
block in on lan0 proto tcp from any to 10.1.0.0/16 flags S/SA
#  block the replies:
block out on lan0 proto tcp from 10.1.0.0 to any flags SA/SA
```

example.11

```
#
# allow any TCP packets from the same subnet as foo is on
# through to host 10.1.1.2 if they are destined for port 6667.
#
pass in proto tcp from 10.2.2.2/24 to 10.1.1.2/32 port = 6667
#
# allow in UDP packets which are NOT from port 53 and are
# destined for localhost
#
pass in proto udp from 10.2.2.2 port != 53 to localhost
#
# block any packet trying to get to X terminal ports, X:0 to
# X:9
#
block in proto tcp from any to any port 5999 >< 6010
#
# allow any connections to be made,except to BSD
# print/r-services this will also protect syslog.
#
block in proto tcp/udp all
pass in proto tcp/udp from any to any port 512 <> 515
#
# allow any connections to be made, except to BSD
# print/r-services
# this will also protect syslog.
#
pass in proto tcp/udp all
block in proto tcp/udp from any to any port 511 >< 516
```

example.12

```
#
# get rid of all short IP fragments (too small for valid
# comparison)
#
block in proto tcp all with short
#
# drop and log any IP packets with options set in them.
#
block in log all with ipopts
#
# log packets with BOTH ssrr and lsrr set
#
log in all with opt lsrr,ssrr
#
# drop any source routing options
#
block in quick all with opt lsrr
block in quick all with opt ssrr
```

example.13

```
#
# log all short TCP packets to lan3, with 10.3.3.3 as the
# intended destination for the packet.
#
block in on lan0 to lan3:10.3.3.3 proto tcp all with short
#
# log all connection attempts for TCP
#
pass in on lan0 dup-to lan1:10.3.3.3 proto tcp all flags S/SA
#
# route all UDP packets through transparently.
#
pass in on ppp0 fastroute proto udp all
#
# route all ICMP packets to network 10 out through lan1, to
# 10.3.3.1
#
pass in on lan0 to lan1:10.3.3.1 proto icmp all
```

example.sr

```
# log all inbound packets on lan0 which has IP options present
# log in on lan0 from any to any with ipopts
#
# block any inbound packets on lan0 which are fragmented and
# "too short" to do any meaningful comparison on. This actually
# only applies to TCP packets which can be missing the
# flags/ports (depending on which part of the fragment you
# see).
#
  block in log quick on lan0 from any to any with short frag
#
# log all inbound TCP packets with the SYN flag (only) set
#  (NOTE: if it were an inbound TCP packet with the SYN flag
#set and it had IP options present, this rule and the above
#would cause it to be logged twice).
#
log in on lan0 proto tcp from any to any flags S/SA

block and log any inbound ICMP unreachables

block in log on lan0 proto icmp from any to any icmp-type
unreach

block and log any inbound UDP packets on lan0 which are going
to port 2049 (the NFS port).

block in log on lan0 proto udp from any to any port = 2049
#
# quickly allow any packets to/from a particular pair of hosts
#
pass in quick from any to 10.1.3.2/32
pass in quick from any to 10.1.0.13/32
pass in quick from 10.1.3.2/32 to any
pass in quick from 10.1.0.13/32 to any
#
# block (and stop matching) any packet with IP options present.
#
block in quick on lan0 from any to any with ipopts
#
# allow any packet through
#
pass in from any to any
```

```
#
# block any inbound UDP packets destined for these subnets.
#
block in on lan0 proto udp from any to 10.1.3.0/24
block in on lan0 proto udp from any to 10.1.1.0/24
block in on lan0 proto udp from any to 10.1.2.0/24
#
# block any inbound TCP packets with only the SYN flag set that
# are destined for these subnets.
#
block in on lan0 proto tcp from any to 10.1.3.0/24 flags S/SA
block in on lan0 proto tcp from any to 10.1.2.0/24 flags S/SA
block in on lan0 proto tcp from any to 10.1.1.0/24 flags S/SA
#
# block any inbound ICMP packets destined for these subnets.
#
block in on lan0 proto icmp from any to 10.1.3.0/24
block in on lan0 proto icmp from any to 10.1.1.0/24
block in on lan0 proto icmp from any to 10.1.2.0/24
```

firewall

```
#Configuring IP Filter for firewall usage.
=========================================

Step 1 - Block out "bad" IP packets.
------------------------------------

Run the perl script "mkfilters".  This will generate a list of
blocking rules which:
a) blocks all packets which might belong to an IP Spoofing
attack;
b) blocks all packets with IP options;
c) blocks all packets which have a length which is too short
for any legal packet;

Step 2 - Convert Network Security Policy to filter rules.
---------------------------------------------------------

Draw up a list of which services you want to allow users to use
on the Internet (e.g. WWW, ftp, etc).  Draw up a separate list
for what you want each host that is part of your firewall to be
allowed to do, including communication with internal hosts.

Step 3 - Create TCP "keep state" rules.
---------------------------------------

For each service that uses TCP, create a rule as follows:

pass in on <int-a> proto tcp from <int-net> to any port
<ext-service> flags S/SA keep state

where
* "int-a" is the internal interface of the firewall.  That is,
it is the closest to your internal network in terms of network
hops.

* "int-net" is the internal network IP# subnet address range.
This might be something like 10.1.0.0/16, or 128.33.1.0/24

* "ext-service" is the service to which you wish to connect or
if it doesn't have a proper name, a number can be used.  The
translation of "ext-service" as a name to a number is
controlled with the /etc/services file.
```

server

```
#
# For a network server, which has two interfaces, 128.1.40.1
#(lan0) and 128.1.2.1 (lan1), we want to block all IP spoofing
# attacks.  lan1 is connected to the majority of the network,
# while lan0 is connected to a leaf subnet.
# We're not concerned about filtering individual services
#
#
pass in quick on lan0 from 128.1.40.0/24 to any
block in log quick on lan0 from any to any
block in log quick on lan1 from 128.1.1.0/24 to any
pass in quick on lan1 from any to any
```

tcpstate

```
#
# Only allow TCP packets in/out of lan0 if there is an outgoing
# connection setup somewhere, waiting for it.
#
pass out quick on lan0 proto tcp from any to any flags S/SAFR
keep state
block out on lan0 proto tcp all
block in on lan0 proto tcp all
#
# allow nameserver queries and replies to pass through, but no
# other UDP
#
pass out quick on lan0 proto udp from any to any port = 53
keep state
block out on lan0 proto udp all
block in on lan0 proto udp all
```

BASIC.NAT

```
#!/sbin/ipnat -f -
#
# THIS EXAMPLE IS WRITTEN FOR IP FILTER 3.3
#
# ppp0 - (external) PPP connection to ISP, address a.b.c.d/32
#
# lan0 - (internal) network interface, address w.x.y.z/32
#
# If only one valid IP address from the ISP, then use this
# rule:
#
map ppp0 w.x.y.z/24 -> a.b.c.d/32 portmap tcp/udp 40000:60000
map ppp0 w.x.y.z/24 -> a.b.c.d/32
#
# If a different dialup IP address is assigned each time, then
# use this rule:
map ppp0 w.x.y.z/24 -> 0/32 portmap tcp/udp 40000:60000
map ppp0 w.x.y.z/24 -> 0/32
#
# If using a class C address space of valid IP addresses from
# an ISP, then use this rule:
#
map ppp0 w.x.y.z/24 -> a.b.c.d/24 portmap tcp/udp 40000:60000
map ppp0 w.x.y.z/24 -> a.b.c.d/24
#
# If using a small number of PCs, use this rule:
#
map ppp0 w.x.y.v/32 -> a.b.c.E/32 portmap tcp/udp 40000:60000
map ppp0 w.x.y.v/32 -> a.b.c.E/32
map ppp0 w.x.y.u/32 -> a.b.c.F/32 portmap tcp/udp 40000:60000
map ppp0 w.x.y.u/32 -> a.b.c.F/32
map ppp0 w.x.y.t/32 -> a.b.c.G/32 portmap tcp/udp 40000:60000
map ppp0 w.x.y.t/32 -> a.b.c.G/32
map ppp0 w.x.y.s/32 -> a.b.c.H/32 portmap tcp/udp 40000:60000
map ppp0 w.x.y.s/32 -> a.b.c.H/32
map ppp0 w.x.y.r/32 -> a.b.c.I/32 portmap tcp/udp 40000:60000
map ppp0 w.x.y.r/32 -> a.b.c.I/32
map ppp0 w.x.y.q/32 -> a.b.c.J/32 portmap tcp/udp 40000:60000
map ppp0 w.x.y.q/32 -> a.b.c.J/32
map ppp0 w.x.y.p/32 -> a.b.c.K/32 portmap tcp/udp 40000:60000
map ppp0 w.x.y.p/32 -> a.b.c.K/32
#
```

```
# For ftp to work using the internal ftp proxy, use the
# following rule:
#
map ppp0 w.x.y.z/24 -> a.b.c.d/32 proxy port ftp ftp/tcp
```

nat.eg

```
# map all tcp connections from 10.1.0.0/16 to 240.1.0.1,
# changing the source
# port number to something between 10,000 and 20,000 inclusive.
# For all other
# IP packets, allocate an IP # between 240.1.0.0 and
# 240.1.0.255, temporarily
# for each new user.
#
map lan1 10.1.0.0/16 -> 240.1.0.1/32 portmap tcp 10000:20000
map lan1 10.1.0.0/16 -> 240.1.0.0/24
#
# Redirection is triggered for input packets.
# For example, to redirect FTP connections through this box, to
the local ftp
# port, forcing them to connect through a proxy, you would use:
#
rdr lan0 0.0.0.0/0 port ftp -> 127.0.0.1 port ftp
```

nat-setup

```
Configuring NAT on your network.
================================
To start setting up NAT, we need to define which is your
"internal" interface and which is your "external" interface.
The "internal" interface is the network adapter connected to
the network with private IP addresses which you need to change
for communicating on the Internet.  The "external" interface is
configured with a valid internet address.
For example, your internal interface might have an IP address
of 10.1.1.1 and be connected to your ethernet, whilst your
external interface might be a PPP connection with an IP number
of 204.51.62.176.
Thus your network might look like this:
<Internal Network>
[pc]      [pc]
|         |
+-+---------+------+
                   |
               [firewall]
                   |
                   |
                Internet
<External Network>
Writing the map-rule.
---------------------
When you're connected to the Internet, you will either have a
block of IP addresses assigned to you, maybe several different
blocks, or you use a single IP address, i.e. with dialup PPP.
If you have a block of addresses assigned, these can be used to
create either a 1:1 mapping (if you have only a few internal IP
addresses) or N:1 mappings, where groups of internal addresses
map to a single IP address and unless you have enough Internet
addresses for a 1:1 mapping, you will want to do "portmapping"
for TCP and UDP port numbers.
For an N:1 situation, you might have:
map ppp0 10.1.0.0/16 -> 209.23.1.5/32 portmap tcp/udp
10000:40000
map ppp0 10.1.0.0/16 -> 209.23.1.5/32 portmap
where if you had 16 addresses available, you could do:
map ppp0 10.1.0.0/16 -> 209.23.1.0/28 portmap tcp/udp
10000:40000
map ppp0 10.1.0.0/16 -> 209.23.1.0/28 portmap
```

```
Or if you wanted to allocate subnets to each IP#, you might do:
map ppp0 10.1.1.0/24 -> 209.23.1.2/32 portmap tcp/udp
10000:40000
map ppp0 10.1.2.0/24 -> 209.23.1.3/32 portmap tcp/udp
10000:40000
map ppp0 10.1.3.0/24 -> 209.23.1.4/32 portmap tcp/udp
10000:40000
map ppp0 10.1.1.0/24 -> 209.23.1.2/32 portmap
map ppp0 10.1.2.0/24 -> 209.23.1.3/32 portmap
map ppp0 10.1.3.0/24 -> 209.23.1.4/32 portmap
*** NOTE: NAT rules are used on a first-match basis only!
Filtering with NAT.
-------------------
IP Filter will always translate addresses in a packet _BEFORE_
it checks its access list for inbound packets and translates
addresses _AFTER_ it has checked the access control lists for
outbound packets.
For example (using the above NAT rules), if you wanted to
prevent all hosts in the 10.1.2.0/24 subnet from using NAT, you
might use the following rule with ipf:
block out on ppp0 from 10.1.2.0/24 to any
block in on ppp0 from any to 10.1.2.0/24
and use these with ipnat:
map ppp0 10.1.0.0/16 -> 209.23.1.0/28 portmap tcp/udp
10000:40000
map ppp0 10.1.0.0/16 -> 209.23.1.0/28 portmap
```

B HP-UX IPFilter Static Linking

This appendix provides instructions for statically linking the HP-UX IPFilter kernel modules to the kernel for HP-UX 11i v1 and HP-UX 11i v2.

Static Linking

IPFilter has two kernel modules, `pfil`, a streams module and `ipf`, a WSIO pseudo driver. These are dynamically loadable kernel modules. When IPFilter is installed on an HP-UX system using `swinstall`, these two modules are loaded and configured as dynamically linked modules. They can be loaded and unloaded when required without shutting down the system as long as the modules are not currently in use.

Static Linking of HP-UX IPFilter on HP-UX 11i v1

As with any other DLKM modules for HP-UX 11i v1, these modules can be statically linked to the kernel. Follow these steps to statically link the IPFilter modules to the kernel:

1. Use the `kmadmin` command to find out if the modules have been loaded dynamically. See the *kmadmin* (1M) manpage for usage information. For example:

   ```
   $ kmadmin -s
   ```

Table B-1

Name	ID	Status	Type
pfil	1	LOADED	STREAMS
ipf	2	LOADED	WSIO

2. Use the `kmsystem` command to find the status of each module. See the *kmsystem* (1M) manpage for more detail. For example:

   ```
   $ kmsystem -q pfil
   ```

Table B-2

Module	Configured	Loadable
pfil	Y	Y

The output is similar for the `ipf` module. This output shows that the `pfil` module is loadable.

3. Use the `kmsystem` command to set the loadable parameter to N.

```
$ kmsystem -l N -c Y ipf
$ kmsystem -q ipf
```

Table B-3

Module	Configured	Loadable
ipf	Y	N

```
$ kmsystem -l N -c Y pfil
```

4. Use the following command to build the new kernel with the modified configuration:

```
$config /stand/system
```

5. Use the `kmupdate` command to prepare the system to boot from the new kernel during the next system shutdown.

```
$ kmupdate /stand/build/vmunix_test
$ shutdown -r 0  # Shutdown the system now
```

This boots the system using the new kernel that has both IPFilter modules statically linked.

CAUTION If you need to remove or update IPFilter software, you must reconfigure the `ipf` and `pfil` modules to link dynamically into the kernel. The install and remove scripts for IPFilter assume the IPFilter modules to be dynamically linked. Do not try installing a newer version or removing the existing IPFilter product if it is statically linked to the kernel.

Static Linking of HP-UX IPFilter on HP-UX 11i v2

Use the following steps to statically link the IPFilter modules to the kernel with HP-UX 11i v2:

1. Set up the IPFilter modules to be statically linked to the kernel using the `kcmodule` command. The modules will be statically linked at the next system boot. See the `kcmodule` (1M) manpage for further details. For example:

   ```
   $ kcmodule -K -h -s pfil=static
   $ kcmodule -K -h -s ipf=static
   ```

2. Reboot the system.

Use the following steps to return the system back to dynamic linking.

1. Set up the IPFilter modules to be dynamically linked to the kernel using the following commands:

   ```
   $ kcmodule -K -h -s pfil=auto
   $ kcmodule -K -h -s ipf=auto
   ```

2. Reboot the system.

CAUTION If you need to remove or update IPFilter software, you must reconfigure the `ipf` and `pfil` modules to link dynamically into the kernel. The install and remove scripts for IPFilter assume the IPFilter modules to be dynamically linked. Do not try installing a newer version or removing the existing IPFilter product if it is statically linked to the kernel.

C Performance Guidelines

This appendix provides performance guidelines for the use of HP-UX IPFilter.

You must take operating environment limits in to account when you configure HP-UX IPFilter. HP-UX does not enforce maximum configuration limits to provide flexibility. However, you must take care not to overburden HP-UX IPFilter systems or unpredictable consequences may result.

This appendix contains the following sections:

- System Configuration
- Rule Loading
- Rule Configuration
- Traffic
- Performance Monitoring

System Configuration

The following are four suggestions for HP-UX system configuration for optimal performance:

Figure C-1 Processing packets through a system

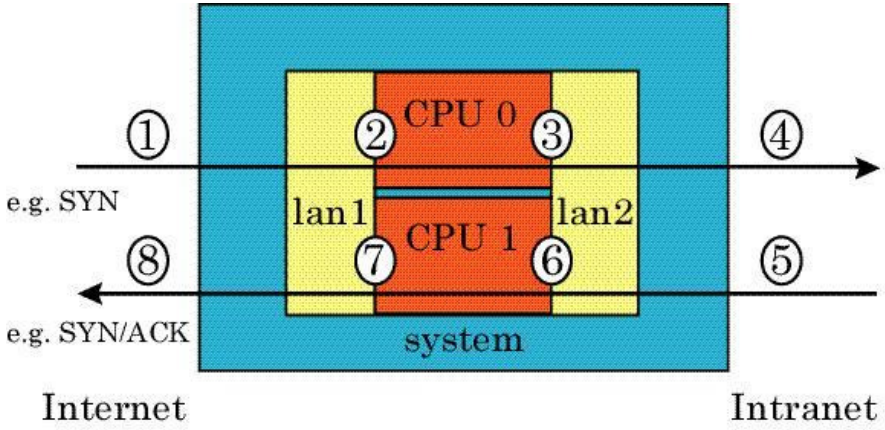


Table C-1 Processing Packets through a System

Packets from the Internet		Packets to the Internet	
1	Packets enter the system	5	Packets enter the system
2	Processed by inbound IPFilter processing	6	Processed by inbound IPFilter processing
3	Processed by outbound IPFilter processing	7	Processed by outbound IPFilter processing
4	Packets leave the system	8	Packets leave the system
Packets are processed twice (2 and 3)		Packets are processed twice (6 and 7)	

1. On an intermediate system, disable the interface on the intranet side. By default, there is redundant processing for each packet through an intermediate system, as shown in Figure C-1. By

disabling the intranet interface, using `ipf -D lan2` in this example, each packet is processed only once in each direction (2 and 7). Do not disable any interface on an end system.

2. If your system has multiple CPUs and LAN cards, be sure traffic is divided evenly between the CPUs. Interrupt migration and PerfView utilities can be used to determine that traffic is spread evenly between CPUs.
3. Dedicate a CPU to each LAN card, if possible. Avoid configuring one CPU to share an application and a LAN, especially if the application is data or computationally intensive. Use the HP-UX Processor Set (PSET) utility to separate applications and LAN processing.
4. If you are configuring an intermediate system, dedicate that system to HP-UX IPFilter. Do not share the system with other standalone applications.

Rule Loading

When you load a large number of new rules to a ruleset, the system must search existing rulesets for duplicate rules. This slows down the loading process.

For example, if there is no `group` rule and there are 5000 rules on the system, the system searches through all 5000 rules to be sure there is no duplication before adding each new rule.

HP-UX IPFilter searches for duplicate rules by group. To speed the search process when loading rules, divide the rules into groups. See “Improving Performance with Rule Groups” on page 72 for information on rule groups. HP recommends configuring a maximum of 5000 rules per group and 5000 groups per system.

You do not need to flush and reload an entire ruleset to modify some rules within the ruleset. Adding rules that already exist slows processing. If you are modifying a large ruleset, follow these steps:

1. Find the difference between the new rule set and the current rule set using the `diff` command.
2. Delete the old rules using the `ipf -rf` command.
3. If your ruleset contains `keep limit` rules, modify the rules with the `ipf -f` command.
4. Add the new rules using the `ipf -f` command. If a rule must be in a specific place in the ruleset, specify the rule number using `@<`*`rule_number`*`>` before the rule.

You can also modify an inactive ruleset and then switch the inactive ruleset for the active ruleset with the `ipf -s` command.

Rule Configuration

To configure IPFilter rules for optimal system performance:

- Avoid using `return-rst` whenever possible.

  From both security and performance perspectives, it is better for IPFilter to block packets anonymous rather than returning a reset packet with a known address.

- Avoid logging whenever possible.

  Excessive logging can impact both storage and CPU performance on the system. Determine the appropriate logging level for your environment.

- Use the `quick` keyword whenever possible.

  The `quick` keyword stops the rule search for a packet a rule matches. Otherwise, IPFilter searches the entire ruleset, which can impact performance if there are a large number of rules.

- Use `keep state` or `keep limit` rules whenever possible.

  Each connection that matches the `keep state` or `keep limit` rule searches through the rule set only once. The following packets for that connection will match the existing state entry and not search the rest of the ruleset.

- Use `group` rules whenever possible.

  For more information, see “Improving Performance with Rule Groups” on page 72.

  In the following example, a connection from 15.13.104.72 must search 102 rules before finding a match.

```
pass in quick proto tcp from 15.13.2.1 to any port = 23 keep
limit 1
pass in quick proto tcp from 15.13.2.2 to any port = 23 keep
limit 2
.
(15.13.2.3 to 15.13.2.99)
.
pass in quick proto tcp from 15.13.2.100 to any port = 23
keep limit 100
pass in quick proto tcp from 15.13.103.0/24 to any port = 23
```

```
keep limit 500
pass in quick proto tcp from 15.13.104.0/24 to any port = 23
keep limit 500
pass in quick proto tcp from 15.13.105.0/24 to any port = 23
keep limit 500
pass in quick proto tcp from 15.13.106.0/24 to any port = 23
keep limit 500
pass in log limit freq 20 quick proto tcp from any to any
port = 23 keep limit 4
```

If the ruleset in the previous example is modified to use the `group` keyword, it is only necessary for the packet to search four rules before finding a match. For example:

```
pass in quick proto tcp from 15.13.2.1-15.13.2.100 to any
port = 23 head 1
pass in quick proto tcp from 15.13.2.1 to any port = 23 keep
limit 1 group 1
pass in quick proto tcp from 15.13.2.2 to any port = 23 keep
limit 2 group 1
.
(15.13.2.3 to 15.13.2.99)
.
pass in quick proto tcp from 15.13.2.100 to any port = 23
keep limit 100 group 1
pass in quick proto tcp from 15.13.103.0/24 to any port = 23
keep limit 500
pass in quick proto tcp from 15.13.104.0/24 to any port = 23
keep limit 500
pass in quick proto tcp from 15.13.105.0/24 to any port = 23
keep limit 500
pass in quick proto tcp from 15.13.106.0/24 to any port = 23
keep limit 500
pass in log limit freq 20 quick proto tcp from any to any
port = 23 keep limit 4
```

- Consolidate rules whenever possible, to minimize searching. For example:

```
pass in quick proto tcp from 15.13.103.72 to any keep limit
80
pass in quick proto tcp from 15.13.103.0-15.13.103.6 to any
keep limit 44
pass in quick proto tcp from 15.13.103.7 to any keep limit
33
pass in quick proto tcp from 15.13.103.8 to any keep limit
33
pass in quick proto tcp from 15.13.103.9 to any keep limit
33
pass in quick proto tcp from 15.13.103.10-15.13.103.255 to
any keep limit 44
pass in quick proto tcp from 15.13.104.0/24 to any keep
limit 44
pass in quick proto tcp from 15.13.105.0/24 to any keep
limit 44
pass in quick proto tcp from 15.13.106.0/24 to any keep
limit 44
pass in quick proto tcp from 15.13.107.0-15.13.107.78 to
any keep limit 44
```

  The previous ruleset can be condensed to the following:

```
pass in quick proto tcp from 15.13.103.0-15.13.107.78 to
any keep limit 33 head 1
pass in quick proto tcp from 15.13.103.72 to any keep limit
80 group 1
pass in quick proto tcp from !15.13.103.7-15.13.103.9 to
any keep limit 44 group 1
```

- For `keep limit` rules, avoid the `cumulative` rule whenever possible.

  If a large number of connections have the same source IP, destination IP, and destination port, system performance is impacted by cumulative rules. Non-cumulative `keep limit` rules keep a cache based on the source IP, destination IP, and destination port. Cumulative rules do not keep a cache based on these parameters.

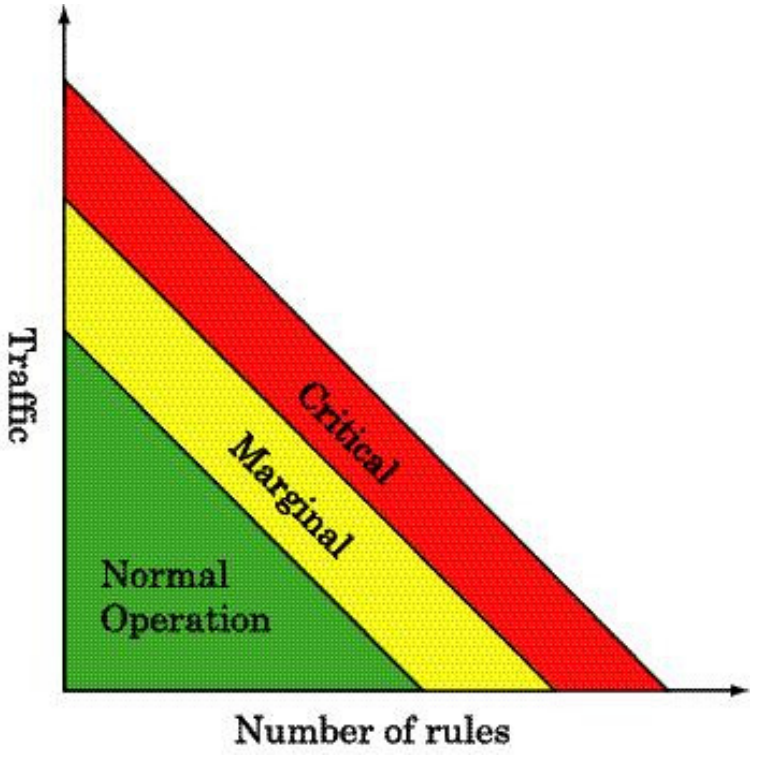
Traffic

To manage IPFilter for optimal system performance:

- Keep the state entries at a manageable level. Many state entries require many CPU cycles to process them. Too many state entries can cause noticeable degradation on a system.
- Keep packet searches on rulesets as short as possible. On a 750-MHz PA-RISC system, a 1000 to 2000 rule search is acceptable. If IPFilter traffic is light, a 5000 rule search is the recommended maximum. The optimal number of rules is dependent on your specific operating environment, including factors such as type of rules and amount of traffic.
- Keep IPFilter traffic at a manageable level. Do not run at peak load all the time. Keep the average CPU usage rate at around 60% to accommodate unexpected peak loads. At peak load times the system compensates with schemes such as dropping packets. However, it is never a good idea to push a system beyond its intended capacity.

For example, the normal region in Figure C-2 shows normal system operation. The system should not operate in the marginal region for a long period of time. Configure your system to raise an alarm if the system reaches the critical level. Define these criteria based your operating environments.

Figure C-2 System Operation



Performance Monitoring

The performance of an IPFilter system depends primarily on four major factors:

- Number and length of rule searches (rule organization)
- Types of rules
- Network traffic
- System configuration

Monitor your system performance to ensure proper operation. HP recommends they following:

- Use `ipfstat -ioh` to monitor the rule searches. If a rule has a high hit count, this indicates that the rule can be optimized.
- Use other `ipfstat` options to monitor IPFilter activities. If you know the normal operating behavior statistics, you can diagnose problems during peak hours more easily.
- Use a performance tool, such as PerfView, to monitor system usage.

Most performance problems can be resolved by changing the system configuration and the IPFilter rule configuration. In some instances, systems may be overwhelmed by network traffic. In these cases, you can implement other traffic-sharing alternatives, such as APA, cluster, and load balancer.

K

L