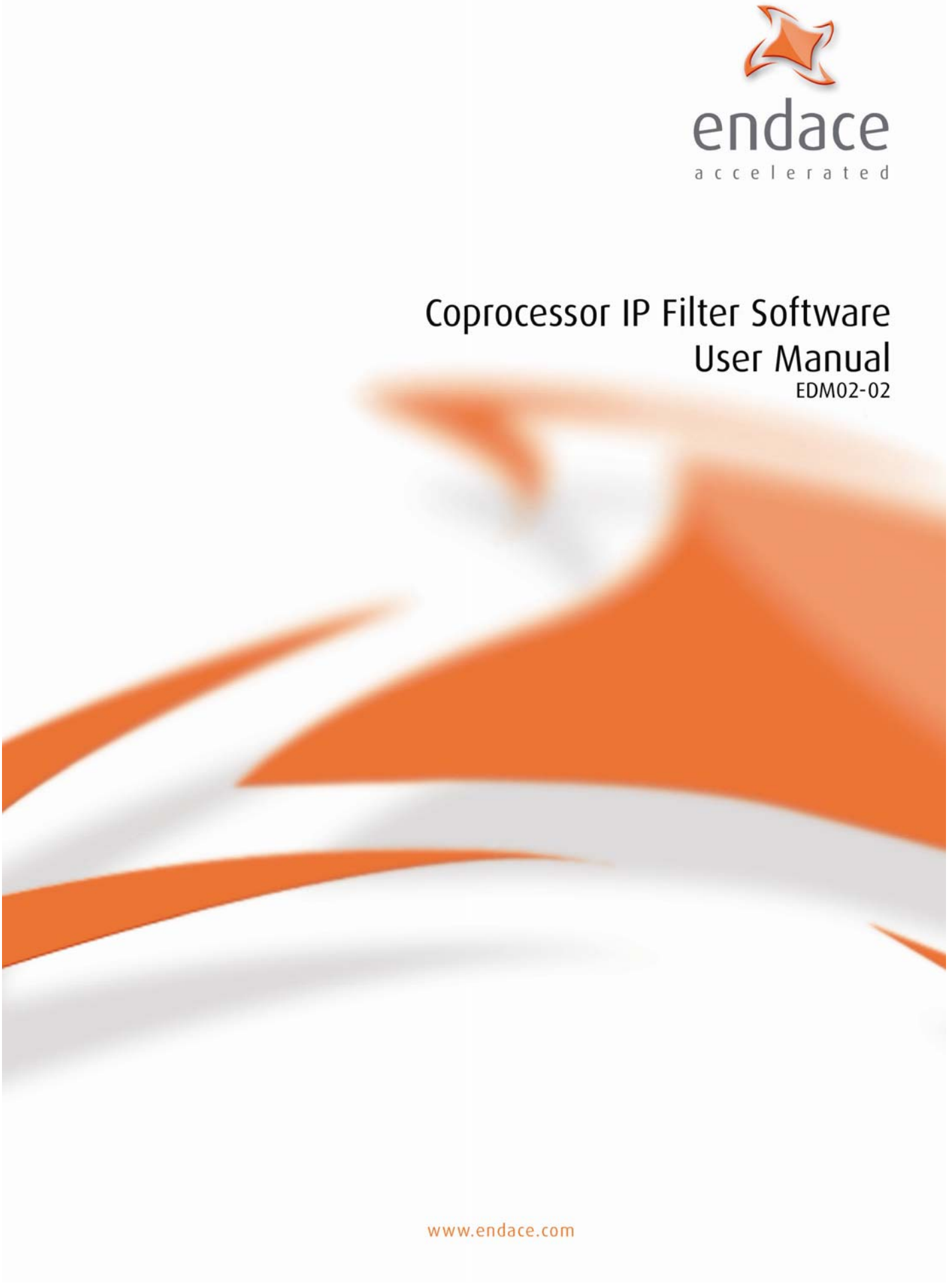

Coprocessor IP Filter Software User Manual

EDM02-02

www.endace.com

Published by:

Endace Measurement Systems® Ltd
Building 7
17 Lambie Drive
PO Box 76802
Manukau City 1702
New Zealand
Phone: +64 9 262 7260
Fax: +64 9 262 7261
support@endace.com
www.endace.com

International Locations

New Zealand
Endace Technology® Ltd
Level 9
85 Alexandra Street
PO Box 19246
Hamilton 2001
New Zealand
Phone: +64 7 839 0540
Fax: +64 7 839 0543
support@endace.com
www.endace.com

Americas
Endace USA® Ltd
Suite 220
11495 Sunset Hill Road
Reston
Virginia 20190
United States of America
Phone: ++1 703 382 0155
Fax: ++1 703 382 0155
support@endace.com
www.endace.com

Europe, Middle East & Africa
Endace Europe® Ltd
Sheraton House
Castle Park
Cambridge CB3 0AX
United Kingdom
Phone: ++44 1223 370 176
Fax: ++44 1223 370 040
support@endace.com
www.endace.com

Conventions Used in this Document

- Command-line examples suitable for entering at command prompts are displayed in `mono-space courier font`.

  The hash # symbol at start of a line represents the prompt and is not entered as part of the command. Results generated by example command-lines are also displayed in `mono-space courier font`.

- Information relating to functions not implemented in this beta version of this product are underlined

Table of Contents

1.0 PREFACE

Introduction

The Endace Filter software package enables IP packets to be filtered based on:

- Ingress interface
- Protocol [IP, ICMP, IGRP, TCP, UDP]
- Source and destination IP address
- TCP and UDP source and destination port number
- TCP flags

The filter software package includes:

- Xilinx images for the DAG 3.8, DAG 4.3S, and DAG 4.3GE cards and Coprocessor
- Support for SC256 and SC128 Co-Processors
- Snort Rule Compiler application for turning Snort-like rules into filters
- Tcpdump Rule Compiler application for turning tcpdump-like rules into filters
- Filter Loader application for loading filters onto the Coprocessor

1.1 Software Manual Purpose

Description

The purpose of the installation manual for the Endace Filter Software package is to identify and explain:

- Installing Endace Filter software
- Filter Loader operation
- Snort Rule Compiler operation
- Tcpdump Rule Compiler operation

1.2 Installation Requirements

Description

The requirements for installing the Endace Filter software are:

- A DAG 3.8, 4.3S or 4.3GE card with an Endace Coprocessor fitted
- For Linux and FreeBSD, version 2.5.6.1 of the DAG software or higher. Customers with a current support contract can download this from the secure Endace website:

  https://www.endace.com/dag-2.5.6.1.tar.gz

- For Windows:

  https://www.endace.com/dag-2.5.6.1-windows.zip

libpcap 0.9.4

libpcap 0.9.4 or higher, required for capture using libpcap which can be downloaded from the following locations:

For Linux and FreeBSD:

http://www.tcpdump.org

For Windows:

http://endace.com/OpenSource_wincap.htm

2.0 INSTALLING ENDACE FILTER SOFTWARE

Introduction

The Snort Rule Compiler, Tcpdump Rule Compiler and Filter Loader are used to program the filtering capabilities of the Coprocessor, with a libpcap or native DAG API application dealing with the received packets.

The installation process also involves building Endace Filter Tools for Linux/FreeBSD. The process is not followed for Windows.

Figure

Figure 1-1 shows the Endace filter operation.

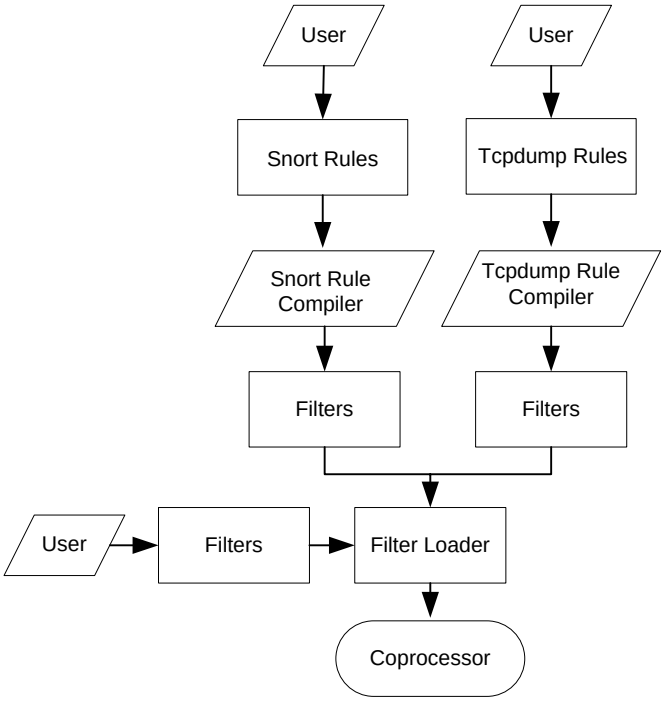

Figure 1-1. Endace Filter Operation.

In this chapter

This chapter covers the following sections of information.

- Install Endace Filter Software
- Build Endace Filter Tools for Linux/FreeBSD
- Perform Self-test
- Configuring DAG Cards and Coprocessor

2.1 Install Endace Filter Software

Description Installing the Endace Filter software involves installation of the base DAG software distribution and then the Endace Filter software distribution.

In the following examples, all software is installed into the home directory of the superuser, `root`.

For the installation, Step 1 below is for Linux/FreeBSD. Step 2 is for Windows.

Procedure Follow these steps to install the Endace Filter software.

Step 1. **Install DAG Software for Linux/FreeBSD**

Install the `dag-2.5.6.1.tar.gz` tarball.

Change the working directory to be the parent directory of the DAG software tree.

```
dag@endace:# cd /root

dag@endace:# ls -l
drwxr-xr-x 9 root root 4096 Mar 17 14:49 dag-2.5.6.1
```

Step 2. **Install DAG Software for Windows**

Install the dag-2.5.6.1-windows.msi.

Use the document EDM04.02 Endace Windows Software User Manual to assist with this Step.

2.2 Build Endace Filter Tools for Linux/FreeBSD

Description The Filter Loader, Snort Rule Compiler and Tcpdump Rule Compiler are built using the standard `configure` script from within the `dag-2.5.6.1` directory:

Contact `support@endace.com` for help with problems that occur in building the tools.

NOTE: The following procedure does not apply for a Windows installation.

Continued on next page

2.2 Build Endace Filter Tools for Linux/FreeBSD, continued

Procedure Follow these steps to build the Endace filter tools for Linux/FreeBSD.

Step 1. Build Tools

Using a standard `configure` script from within the `dag-2.5.6.1` directory:

```
dag@endace:# ./configure
dag@endace:# make
```

After building, the tools are created as:

```
dag-2.5.6.1/filtering/filter_loader/filter_loader
dag-2.5.6.1/filtering/snort_compiler/snort_compiler
dag-2.5.6.1/filtering/tcpdump_compiler/tcpdump_compiler
```

Step 2. Install Tools

By default, these tools can be installed to `/usr/local/bin` using:

```
dag@endace:# make install
```

NOTE: The install directory can be set using:

`--prefix` command-line argument to the `configure` script

For more information about available options, run the `configure` script:

```
dag@endace:# ./configure –help
```

2.3 Perform Self-test

Description The Tcpdump Rule Compiler comes with a self-test script that can be used to verify the basic functionality of the binary.

Procedure Follow this step to perform the self-test.

Step 1. Run test

Via the `run_tests.sh` script in the `dag-2.5.6.1/filtering/tcpdump_compiler` directory:

```
dag@endace:# cd dag-2.5.6.1/filtering/tcpdump_compiler

dag@endace:# ./run_tests.sh
```

The following message is displayed indicating all 32 tests succeeded:

```
End of tests
```

2.4 Configuring DAG Cards and Coprocessor

Description

The DAG card and Coprocessor configuration feedback provided by a local system may vary from that shown in the following steps.

When changing the filters on the Coprocessor, there is a delay before the new filters take effect. For example, the first few hundred packets after the change are filtered according to the previous filter set.

Please note that there are different Co-Processor firmware images for the SC128 and SC256 Co-Processors.

How to tell if you have a SC128 or SC256 Co-Processor

You can tell the difference between a SC128 and SC256 Co-Processor by checking the output of the `daginf` command

SC128 Co-Processor

```
qa11:~# daginf
id       0
model    DAG 3.8S
device   0x3800
phy addr 369098752
buf size 134217728 (128MB)
iom size 65536 (64kB)
copro    SC128
```

SC256 Co-Processor

```
qa3:~# daginf
id       0
model    DAG 4.3S
device   0x4300
phy addr 780140544
buf size 134217728 (128MB)
iom size 65536 (64kB)
copro    SC256 Rev C
```

Standard feature

The filters can be configured so that packets are placed into two memory buffers, with each memory buffer capable of being read by a separate process.

If one of the reading processes blocks for a sufficient period of time it is possible for both processes to see packet loss, as opposed to only the blocked process seeing packet loss.

In this section

This section covers the following topics of information.

- Configure DAG 3.8 Card and Coprocessor
- Configure DAG 4.3S Card and Coprocessor
- Configure DAG 4.3GE Card and Coprocessor

2.4.1 Configure DAG 3.8 Card and Coprocessor

Description To configure the DAG 3.8S card and coprocessor, the DAG card filter image and coprocessor image are loaded, followed by configuring the card. The DAG 3.8S supports the SC128 Coprocessor only.

Procedure Follow these steps to configure a DAG 3.8S card and Coprocessor.

Step 1. **Load DAG card filter image.**

The Endace Filter firmware image for the DAG card is loaded by:

```
dag@endace:# dagrom -d dag0 -rpvy -f dag-2.5.6.1/xilinx/dag38s-ipf.bit

dagrom: verbose: ROM version 0, base 0x0158
dagrom: verbose: cfi_identify 546: Detected 8-bit device
dagrom: verbose: CFI: Vendor AMD/Fujitsu, interface async x8 only, size 0x15
dagrom: verbose: ROM.0 Am29LV017D 16MBit.
dagrom: verbose: Size of image (bytes) : 469202
dagrom: verbose: read  0x0007f000 0xff
dagrom: verbose: prog required
dagrom: verbose: dag0 dev 0x3800 pos 0 got idx 9
dagrom: verbose: erase blitz start...
dagrom: verbose: erase blitz accepted...
dagrom: verbose: erase blitz complete
dagrom: verbose: write 0x00072000 0x17
dagrom: verbose: read  0x0007f000 0xff
dagrom: verbose: verify ok
current:edag38spci_cp1-ipf_v2_2 2v1000fg456 2005/08/18 12:24:05 *
stable:  edag38spci_erf_v2_13 2v1000fg456 2005/04/21 16:18:14
Card Serial: 3351
```

Step 2. **Load Coprocessor Image**

The Endace Filter firmware images for the Coprocessor are loaded by:

```
dag@endace:# dagld -d dag0 -x dag-2.5.6.1/xilinx/dag38pp-
terf.bit:dag-2.5.6.1/xilinx/copro-ipf38s.bit

Waiting for Xilinx1 (dag38spp_erf_v2_8 2s300eft256) to
program...
FPGA Initialized.
Starting to program
.....
File loaded.
Done.
Waiting for Xilinx2 (ec10gcp_ipf_v2_2 2v2000ff896) to
program...
FPGA Initialized.
Starting to program
.....
File loaded.
Done.
```

Step 3. Configure DAG card.

A reasonable initial configuration for OC-12c operation is obtained with the following arguments to `dagthree`:

```
  dag@endace:# dagthree –d dag0 default oc12 slen=1540

linkA   PoS noreset OC12c nolt0 fcl noeql enablea
linkB   PoS noreset OC12c nolt0 fcl noeql enableb
sonetA  noscramble slave
sonetB  noscramble slave
posA    nocrc pscramble
posB    nocrc pscramble
packet  varlen slen=1540 align64
packetA drop=0
packetB drop=0
ipf     nodrop steer=stream0
pcix    66MHz 64-bit buf=128MiB rxstreams=2 txstreams=0
mem=64:0:64:0
```

2.4.2 Configure DAG 4.3S Card and Coprocessor

Description To configure the DAG 4.3S card and coprocessor, the DAG card filter image and coprocessor image are loaded, followed by configuring the card. Please note the DAG 4.3S has different Coprocessor firmware images for the SC128 and SC256 Co-Processors.

Procedure Follow these steps to configure a DAG 4.3S card and Coprocessor.

Step 1. Load DAG card filter image.

The Endace Filter firmware image for the DAG card is loaded by:

```
dag@endace:# dagrom -d dag0 –rpvy -f dag-2.5.6.1/xilinx/dag43s-
ipf.bit

dagrom: verbose: ROM version 0, base 0x0158
dagrom: verbose: cfi_identify 546: Detected 8-bit device
dagrom: verbose: CFI: Vendor AMD/Fujitsu, interface async x8 only,
size 0x15
dagrom: verbose: ROM.0 Am29LV017D 16MBit.
dagrom: verbose: Size of image (bytes) : 469199
dagrom: verbose: read  0x0007f000 0xff
dagrom: verbose: prog required
dagrom: verbose: dag0 dev 0x4300 pos 0 got idx 14
dagrom: verbose: erase blitz start...
dagrom: verbose: erase blitz accepted...
dagrom: verbose: erase blitz complete
dagrom: verbose: write 0x00072000 0xaf
dagrom: verbose: read  0x0007f000 0xff
dagrom: verbose: verify ok
current: edag43spcix_ipf_v2_3 2v1000ff896 2005/11/17 15:51:05 *
stable:  edag43spcix_erf_v2_12 2v1000ff896 2004/04/13 14:53:33
Card Serial: 3864
```

Step 2. Load Coprocessor Image

The Endace Filter firmware image for the Coprocessor is loaded by:

For the SC128 Co-Processor

```
dag@endace:# dagld -d dag0 –x dag-2.5.6.1/xilinx/copro-
ipf43s.bit

Waiting for Xilinx1 (ec10gcp_ipf32_cp_v2_1 2v2000ff896) to
program ...
FPGA Initialized.
Starting to program
.................................................
File loaded.
Done.
```

For the SC256 Co-Processor

```
dag@endace:# dagld -d dag0 –x dag-2.5.6.1/xilinx/copro2-
ipf43s.bit

Waiting for Xilinx1 (ec20gcp_ipf32_cp_v2_5 2v2000ff896) to program
...
FPGA Initialized.
Starting to program
.................................................
File loaded.
Done.
```

Step 3. Configure DAG card.

A reasonable initial configuration is obtained with the following arguments:

```
dag@endace:~/end# dagfour -d dag0 default slen=1540

light   nolaser
link    noreset OC48c nofcl noeql
sonet   master scramble
POS     crc32 nocrcstrip short=16 long=1536 discard pscramble
rxpkts txpkts
packet  varlen slen=1540 align64
packetA drop=0
ipf     nodrop steer=colour
pcix    133MHz 64-bit buf=128MiB rxstreams=2 txstreams=0
mem=64:0:64:0
```

2.4.3 Configure DAG 4.3GE Card and Coprocessor

Description To configure the DAG 4.3GE card and coprocessor, the DAG card filter image and coprocessor image are loaded, followed by configuring the card.

Procedure Follow these steps to configure a DAG 4.3GE card and Coprocessor.
Please note the DAG 4.3GE has different Coprocessor firmware images for the SC128 and SC256 Co-Processors.

Step 1. Load DAG card filter image.

The Endace Filter firmware image for the DAG card is loaded by:

```
  dag@endace:# dagrom -d dag0 -rpvy -f dag-2.5.6.1/xilinx/dag43ge-ipf.bit

dagrom: verbose: ROM version 0, base 0x0158
dagrom: verbose: cfi_identify 548: Detected 8-bit device
dagrom: verbose: CFI: Vendor AMD/Fujitsu, interface async x8 only, size 0x15
dagrom: verbose: ROM.0 Am29LV017D 16MBit.
dagrom: verbose: romid 456
dagrom: verbose: base 344
dagrom: verbose: rom_version 0
dagrom: verbose: device_code 0x430e
dagrom: verbose: ident Am29LV017D 16MBit
dagrom: verbose: size 1048576
dagrom: verbose: sector 65536
dagrom: verbose: bstart 0
dagrom: verbose: bsize 524288
dagrom: verbose: tstart 524288
dagrom: verbose: tsize 524288
dagrom: verbose: pstart 0
dagrom: verbose: psize 0
dagrom: verbose: mpu_id -1
dagrom: verbose: mpu_rom 0
dagrom: verbose: read  0x0007f000 0xff
dagrom: verbose: reprog ok
current:        edag43epci_ipf_v2_4 2v1000ff896 2005/11/16 16:55:24 *
stable:         edag43epci_erf_v2_11 2v1000ff896 2005/04/21 16:22:35
Card Serial: 4001
```

Step 2. Load Coprocessor Image

The Endace Filter firmware image for the Coprocessor is loaded by:

For the SC128 Co-Processor

```
dag@endace:# dagld -d dag0 -x dag-2.5.6.1/xilinx/copro-ipf43ge.bit

Waiting for Xilinx1 (ec10gcp_ipf_v2_2 2v2000ff896) to program...
FPGA Initialized.
Starting to program
.....................................................................
File loaded.
Done.
```

For the SC256 Co-Processor

```
dag@endace:# dagld -d dag0 –x dag-2.5.6.1/xilinx/copro2-ipf43ge.bit

Waiting for Xilinx1 (ec20gcp_ipf32_cp_v2_5 2v2000ff896) to program ...
FPGA Initialized.
Starting to program
.................................................
File loaded.
Done.
```

Step 3. Configure DAG card.

A reasonable initial configuration is obtained with the following arguments to `dagfour`:

```
dag@endace:# dagfour –d dag0 default slen=1540

linkA nonic noeql rxpkts txpkts crc long=1518 enablea
linkB nonic noeql rxpkts txpkts crc long=1518 enableb
packet varlen slen=1540 align64
packetA drop=0
packetB drop=0
ipf  nodrop steer=steam0
pcix 133MHz 64-bit buf=128MiB rxstreams=2 txstreams=0
mem=64:0:64:0
```

3.0 FILTER LOADER OPERATION

Introduction

The Filter Loader loads a set of filters from a text file or standard input into a Coprocessor-equipped DAG card running Endace Filter firmware.

The filter set can be written by hand, generated from Snort-style rules by the Snort Rule Compiler application, or generated from tcpdump-style rules by the Tcpdump Rule Compiler application.

A DAG card running Endace Filter firmware can support up to two filter sets per interface, one active and one inactive.

When a new filter set is loaded the Filter Loader can restart the card and make the new filter set active. Or, the Filter Loader can load the new filter set in an inactive state and instruct the card to atomically switch the active and inactive filter sets. This allows the filter set to be dynamically modified with zero packet loss.

Table

Table 3-1 shows the size of the filter sets for different card configurations.

Interfaces	Hot-Swap Ability	Filters Per Set
1	No	16384
1	Yes	8192
2	No	8192
2	Yes	4096

Table 3-1. Size of Filter Sets for Different Card Configurations.

Standard feature

This Endace Filter module version includes the ability to steer packets to one of two memory buffers. These memory buffers are referred to as `red` and `blue` in the filter rules, and are accessed through the DAG API as receive streams 0 and 2 respectively.

For example, for a DAG card located at `dag0`, the memory buffer containing the red packets can be referred to as `dag0:0` when using the standard DAG utilities. Similarly, the memory buffer containing the blue packets can be referred to as `dag0:2` when using the standard DAG utilities.

In this chapter

This chapter covers the following sections of information.

- Simple to Complex Filters
- Filter Loader Command-line Options and Flags
- `--mapping` Command Line Option Effect on Endace Record Format

3.1 Simple to Complex Filters

Description

Filters are simple, one-line specifications used to describe characteristics of packets considered to be a "match", together with an action to take for matching packets. Two actions currently supported are:

accept a packet.	Accepted packets are passed to the host computer where they can be received using libpcap [2] or the native DAG API and further processed in software,
reject a packet.	Dropped packets are not sent to the host, saving valuable CPU cycles for analyzing the packets that are of most interest.

Filters are loaded into the Coprocessor in the order they are presented, and the ultimate filter should be a catch-all accept or reject filter. Endace Filter supports filtering on:

- Ingress interface
- Protocol [ICMP, IGRP, RawIP/TCP or UDP]
- Source and destination IP addresses
- TCP and UDP source and destination port numbers
- TCP flags

IP addresses, port numbers, and TCP flags in the filter entries can take values 0, 1 and "don't care", which is represented by a dash ('-'). The *classification* of the packet is an integer in the range 0, 16383 written into the ERF, as described in the ERF Record table within Section 3.3 – `mapping` Command Line Option Effect on Endace Record Format.

To retrieve the classification, these bytes are considered as a single 16-bit quantity in network byte order, with the classification being the most significant 14 bits. The least significant 2 bits encode the memory buffer into which the packet was steered, as shown below:

1 = buffer one	3 = both buffers
2 = buffer two	0 = neither buffer

Within the filter examples in the following sections of information, filter lines have been wrapped to fit on the printed page. Filters actually presented to the Filter Loader must be written one per line.

In this section

This section covers the following topics of information.

- Simple Filter Example
- Less Simple Filter Example
- Complex Filter Example
- Filtering on TCP Flags
- Interface-specific Filters
- Packet-steering Filters Optional Feature

3.1.1 Simple Filter Example

Description The simple filter example in Table 3-2 captures TCP packets with a source IP address of 192.168.0.1 sent from any source port to the IP address 192.168.0.2 on port 80.

Table Table 3-2 shows a typical simple filter example.

```
1 accept tcp  src-ip {11000000101010000000000000000001} src-port {----------------}
              dst-ip {11000000101010000000000000000010} dst-port {0000000001010000}
0 reject all  src-ip {--------------------------------} src-port {----------------}
              dst-ip {--------------------------------} dst-port {----------------}
```

Table 3-2. Typical Simple Filter Example.

In the Table 3-2 example the second filter uses the keyword `'all'` in place of a specific protocol to reject packets that do not match the first filter.

The initial integer is a tag that can be used by packet-sniffing software to determine which filter caused the packet to be accepted.

Because a snap length has not been specified for filter 1, the effective snap length for packets that match the first filter will be determined by:

(a) The argument to the command-line option `--snap`, if given;
(b) The default snap length otherwise [65536].

Because an interface has not been specified for the filters, they will be applied to the interface specified by:

(a) The argument to the command-line option `--iface`, if given;
(b) The default interface otherwise (0).

The snap length and interface explanation applies to all examples in this section of information.

3.1.2 Less Simple Filter Example

Description The less simple filter example in Table 3-3 captures TCP packets with a source IP address in the subnet 192.168.0.0/16 and any source port to the IP address 192.168.0.2 on ports 80 or 81.

Traffic to/from an entire subnet can be captured by a single filter through the use of "don’t care" entries.

Table Table 3-3 shows a less simple filter example.

```
 1  accept   tcp  src-ip  {1100000010101000----------------}   src-port  {----------------}
                  dst-ip  {11000000101010000000000000000010}   dst-port  {000000000101000-}
 0  reject   all  src-ip  {--------------------------------}   src-port  {----------------}
                  dst-ip  {--------------------------------}   dst-port  {----------------}
```

Table 3-3. Less Simple Filter Example.

3.1.3 Complex Filter Example

Description The complex filter example in Table 3-4 captures TCP packets with a source IP address in the subnet 192.168.0.0/16 and any source port to the IP address 192.168.1.2 on ports 80 or 81, except for packets to or from the IP address 192.168.1.1.

The filters are evaluated in order. In the Table 3-4 example below, packets with a source IP address of 192.168.1.1 for the first filter, or destination IP address of 192.168.1.1 for the second filter, are discarded before reaching the third filter.

Table Table 3-4 shows a complex filter example.

```
2  reject   tcp  src-ip {11000000101010000000000100000001}   src-port {----------------}
                  dst-ip {--------------------------------}   dst-port {----------------}
2  reject   tcp  src-ip {--------------------------------}   src-port {----------------}
                  dst-ip {11000000101010000000000100000001}   dst-port {----------------}
1  accept   tcp  src-ip {1100000010101000----------------}   src-port {----------------}
                  dst-ip {11000000101010000000000100000010}   dst-port {000000000101000-}
0  reject   all  src-ip {--------------------------------}   src-port {----------------}
                  dst-ip {--------------------------------}   dst-port {----------------}
```

Table 3-4. Complex Filter Example.

3.1.4 Filtering on TCP Flags

Description The TCP flags filter example in Table 3-5 captures TCP packets from any source to any destination that have the SYN flag set.

The `tcp-flags` field may be present for non-TCP rules, in which case it is ignored, as for the second filter below.

If the `tcp-flags` field is not present for a TCP rule, then it is considered to be all "don't care" entries.

Table Table 3-5 shows the TCP flags filter example.

```
1 accept tcp src-ip {------------------------------} src-port {----------------}
             dst-ip {------------------------------} dst-port {----------------} tcp-flags {------1-}
0 reject all src-ip {------------------------------} src-port {----------------}
             dst-ip {------------------------------} dst-port {----------------} tcp-flags {--------}
```

Table 3-5. Filtering on TCP Flags Example.

3.1.5 Interface-specific Filters

Description The interface-specific filters example in Table 3-6 captures TCP packets from 192.168.0.1 to 192.168.0.2 on interface 0, and TCP packets from 192.168.0.3 to 192.168.0.4 on interface 1.

If an interface is specified for any filter, then an interface must be specified for all filters. The Filter Loader will either apply all filters to the interface given by the command-line option `--iface`, or it will apply the filters to the interfaces specified on a per-filter basis as in the Table 3-6 example below.

To minimize potential for confusion, the Filter Loader reports an error if these two modes of operation are mixed, such as attempting to load a filter file in which some filters have per-filter interfaces and others do not.

To have a default reject filter in place, it is included once for each interface.

Table Table 3-6 shows an interface-specific filters example.

```
2 accept  tcp  src-ip {11000000101010000000000000000001} src-port {----------------}
               dst-ip {11000000101010000000000000000010} dst-port {0000000001010000} iface 0
1 accept  tcp  src-ip {11000000101010000000000000000011} src-port {----------------}
               dst-ip {11000000101010000000000000000100} dst-port {0000000001010000} iface 1
0 reject  all  src-ip {------------------------------} src-port {----------------}
               dst-ip {------------------------------} dst-port {----------------} iface 0
0 reject  all  src-ip {------------------------------} src-port {----------------}
               dst-ip {------------------------------} dst-port {----------------} iface 1
```

Table 3-6. Interface-specific Filters Example.

3.1.6 Packet-steering Filters Optional Feature

Description The packet steering filter example in Table 3-7 sends all TCP packets to the red memory buffer, such as receive stream 0, and all UDP packets to the blue memory buffer, such as receive stream 2.

It is possible to have a packet sent to both memory buffers by including both the red and blue keywords.

When a memory buffer is not specified for an accept rule, packets matching the rule will be sent to the red memory buffer, receive stream 0.

Table Table 3-7 shows a packet steering filters example.

```
1 accept red  tcp  src-ip {-------------------------------} src-port {---------------}
                   dst-ip {-------------------------------} dst-port {---------------}
2 accept blue udp  src-ip {-------------------------------} src-port {---------------}
                   dst-ip {-------------------------------} dst-port {---------------}
0 reject      all  src-ip {-------------------------------} src-port {---------------}
                  dst-ip {-------------------------------} dst-port {---------------}
```

Table 3-7. Packet Steering Filters Example.

3.2 Filter Loader Command-line Options and Flags

Description There are a number of command-line flags and options recognised by the Filter Loader.

Command-line flags. The following table explains the short and long options command-line flags.

Short Option	Long Option	Explanation
-d	**--device**	Followed by device name of the DAG card to configure, such as `dag0`.
-l	**--linktype**	Followed by the type of link being monitored. Valid options are ‘ethernet’ and ‘pos4’.
-m	**--mapping**	Where to place the packet classification in the received packets. Valid values are: ’color’, ‘colour’ ‘rxerror’, ‘lcntr’, ‘flags’, ‘entire’, ‘hdlcheader’ (PoS links only), and ‘padoffset’ (Ethernet links only). Also ‘padoffset0’, ‘hdlcheader0’, and ‘colour0’. Mapping that end with 0 send all ERF records to receive stream 0. For example `dag 0:0`
-i	**--infile**	Followed by an input file name which contains Snort-like rules, one per line. If this option is not present the rules are read from standard input.
	--initialise	If this flag is present then the Coprocessor is initialised before filters are loaded. If this flag is not present, then the filters will be hot-swapped with the currently active filters.
	--init-ports	
	--init-ifaces	Followed by the number of interfaces. This value is used to configure the Coprocessor by dividing the filters into sets that apply to each interface. The default is 1. This option can only be specified when the `--initialise` flag is present.

Continued on next page

3.2 Filter Loader Command-line Options and Flags, continued

Command-line flags. (continued)

Short Option	Long Option	Explanation
	--init-rulesets	Followed by the number of rulesets per interface. This value is used to configure the Coprocessor by dividing the filters into rulesets that apply to each interface. The default is 1. This option can only be specified when the `--initialise` flag is present.
-p	**--port**	
	--iface	Followed by the identifier for the interface that the rules should apply to, such as `0` or `1`. If no interface is specified then the filters are applied to all interfaces, unless the filters file specifies per-filter interfaces with the 'iface' command.
-s	**--snap**	Followed by the number of bytes to be captured from the payload of the packet. This option sets a default snap-length for filters which do not explicitly contain a snap-length. If this option is not present then filters which do not explicitly contain a snap-length will capture entire packets.
-h	**--usage**	
-?	**--help**	If this flag is present the Filter Loader displays a help message and then exits.
	--version	Display version information.

3.3 `--mapping` Command Line Option Effect on Endace Record Format

Description The `--mapping` command line option recognized by the Filter Loader can have a number of effects on the ERF record. These can include setting of the RX error bit for packets, writing of classifications, the results register being written into the ERF Record payload, and setting destination streams.

ERF Record The effect of the `--mapping` filter loader command line option is shown in the following table.

Name	Effect on ERF Record	ERF Type[s] Received	Link	Force Stream0
rxerror	The RX error bit is set for packets that would be dropped.	`TYPE_ETH` `TYPE_HDLC_POS`	Both	No
color colour lcntr	The 16-bit color [14-bit classification and 2-bit destination stream field] is written into the color field of the ERF record.	`TYPE_COLOR_ETH` `TYPE_COLOR_HDLC_POS`	Both	No
padoffset	The color is written into the pad and offset bytes of the Ethernet ERF record.	`TYPE_ETH`	Eth	No
hdlcheader	The color is written into the first two bytes of the four-byte HDLC header.	`TYPE_HDLC_POS`	PoS	No
entire	The results register is written into the ERF Record payload.	`TYPE_ETH` `TYPE_HDLC_POS`	Both	No
color0 colour0	As for ‘color’, but all packets are sent to stream 0. This includes those packets that would normally have been dropped, which have a destination stream of 0.	`TYPE_COLOR_ETH` `TYPE_COLOR_HDLC_POS`	Both	Yes
padoffset0	As for ‘padoffset’, but all packets are sent to stream 0. This includes those packets that would normally have been dropped, which will have a destination stream of 0.	`TYPE_ETH`	Eth	Yes
hdlcheader0	As for ‘hdlcheader’, but all packets are sent to stream 0. This includes those packets that would normally have been dropped, which have a destination stream of 0.	`TYPE_HDLC_POS`	PoS	Yes

4.0 SNORT RULE COMPILER

Introduction The Snort Rule Compiler forms part of the Endace Filter system.

Snort is an Open Source Network Intrusion Detection System [NIDS] controlled by a set of pattern/action rules residing in a configuration file of a specific format.

The Snort Rule Compiler takes rules from a text file or passed in via standard input and produces a set of filters that correspond to those rules. This set of filters can then be loaded into a Coprocessor-equipped DAG card running Endace firmware by the Filter Loader application.

Rules are specified using a Snort-like syntax that specifies the protocol [ICMP, IP, TCP or UDP], source/destination IP addresses and source destination ports for TCP and UDP.

The actual filter lines produced by the Snort Rule Compiler are written one per line. The examples in this chapter of information are wrapped for printing purposes.

In this chapter This chapter covers the following sections of information.

- Snort Rule Compiler Examples
- Snort Grammar Specification
- Snort Grammar Variables
- Snort Grammar Variable Examples
- Snort Command-line Options and Flags

4.1 Snort Rule Compiler Examples

Description Using user-defined rules from a compiler, Snort examines all packets going through a specific network that it is set up to monitor and alerts when it finds specific predefined patterns that could be malicious.

The Snort rules can range from the simple and less simple through to more complex ones.

In this section This section covers the following topics of information.

- Simple Snort Rule Output
- Less Simple Snort Rule Output
- Complex Snort Rule Output

4.1.1 Simple Snort Rule Output

Description The simple Snort rule shown in Table 4-1 results in the output from the Snort Rule Compiler. The Snort Rule Compiler expresses both destination ports by using a "don't care" entry in the destination port.

The second filter is present because a default filter is always added to accept or reject packets that do not match any other rules. Unless the command-line flag `--accept` is given, the default filter will reject packets.

Table Table 4-1 shows a simple Snort rule output.

Rule:

```
   accept tcp 192.168.1.1 any -> 192.168.1.2 80:81
```

Output:

```
# Filter file created by snort_compiler at Tue Mar 16 16:47:49 2004.
   2 accept tcp src-ip {11000000101010000000000100000001} src-port {----------------}
                dst-ip {11000000101010000000000100000010} dst-port {000000000101000-}
   0 reject  ip src-ip {--------------------------------} src-port {----------------}
                dst-ip {--------------------------------} dst-port {----------------}
```

Table 4-1. Simple Snort Rule Output.

4.1.2 Less Simple Snort Rule Output

Description In the less simple Snort rule shown in Table 4-2, the Snort Rule Compiler was able to combine the 192.168.1.1 and 192.168.3.1 source IP addresses into a single filter with a "don't care" entry.

Table Table 4-2. A less simple Snort rule output.

Rule:

```
    accept tcp [192.168.1.1,192.168.2.1,192.168.3.1] any -> 192.168.1.2 80
```

Output:

```
# Filter file created by snort_compiler at Wed Mar 17 11:51:27 2004.
   2 accept tcp src-ip {1100000010101000000000-100000001} src-port {----------------}
                dst-ip {11000000101010000000000100000010} dst-port {0000000001010000}
   2 accept tcp src-ip {11000000101010000000001000000001} src-port {----------------}
                dst-ip {11000000101010000000000100000010} dst-port {0000000001010000}
   0 reject  ip src-ip {--------------------------------} src-port {----------------}
                dst-ip {--------------------------------} dst-port {----------------}
```

Table 4-2. Less Simple Snort Rule Output.

4.1.3 Complex Snort Rule Output

Description The complex Snort rule output shown in Table 4-3 can be read as "accept the headers and first 50 bytes of payload for all TCP traffic destined for host 192.168.1.24 on ports 80 to 90 inclusive that does not have a source IP address from the subnets 10.0.0.0/8 and 127.0.0.0/16".

The output expands to 58 filters with one default rule plus nineteen source IP addresses combined with three destination port numbers. The last seven filters for the complex rule output are shown in Table 4-3.

The Snort Rule Compiler encoded the eleven destination port numbers as three entries covering ports 80–87 with three "don’t care" bits, 88–89 with one "don’t care" bit, and 90.

Table Table 4-3 shows a complex Snort rule output.

Rule:

```
accept tcp ![10.0.0.0/8,127.0.0.0/16] any -> 192.168.1.24/32 80:90 snap 50
```

Output:

```
...
   2 accept tcp src-ip {0111111101----------------------} src-port {----------------}
                dst-ip {11000000101010000000000100011000} dst-port {0000000001010---} snap 50
   2 accept tcp src-ip {0111111101----------------------} src-port {----------------}
                dst-ip {11000000101010000000000100011000} dst-port {000000000101100-} snap 50
   2 accept tcp src-ip {0111111101----------------------} src-port {----------------}
                dst-ip {11000000101010000000000100011000} dst-port {0000000001011010} snap 50
   2 accept tcp src-ip {011111111-----------------------} src-port {----------------}
                dst-ip {11000000101010000000000100011000} dst-port {0000000001010---} snap 50
   2 accept tcp src-ip {011111111-----------------------} src-port {----------------}
                dst-ip {11000000101010000000000100011000} dst-port {000000000101100-} snap 50
   2 accept tcp src-ip {011111111-----------------------} src-port {----------------}
                dst-ip {11000000101010000000000100011000} dst-port {0000000001011010} snap 50
   0 reject ip src-ip {---------------------------------} src-port {----------------}
                dst-ip {---------------------------------} dst-port {----------------} snap 50
```

Table 4-3. Complex Snort Rule Output.

4.2 Snort Grammar Specification

Description A formal specification for the grammar is:

- Pipe character (|) indicates choice
- Speech marks (") indicate literal data
- Braces ({}) indicate optional elements
- em-dash (--) indicates a range
- Literal exclamation marks (!) used in a rule indicate a logical negation

rule::= keyword protocol source direction target snaplength {body}

keyword ::= "accept" | "reject"
protocol ::= "tcp" | "udp" | "ip" | "icmp"
direction ::= "->" | "<>"
snaplength ::= 0–65535

source ::= source_ip source_port
target ::= target_ip target_port

source_ip ::= ip_address
target_ip ::= ip_address

ip_address ::= {"!"} ip_set
ip_set ::= single_ip_address | "["single_ip_address "," ip_set"]"
single_ip_address ::= octet "." octet "." octet "." octet { "/" mask }
octet ::= 0–255
mask ::= 1–32

source port ::= {"!"} port_range
target_port ::= {"!"} port_range

port_range ::= single_port | ":" single_port | single_port ":"
| single_port ":" single_port
single_port ::= 1–65535

Body ::= "(" ASCII_text")"

At this stage the body of the rule is optional, and if present has no effect.

4.3 Snort Grammar Variables

Description The Snort grammar supports use of variables for IP addresses and ports. Variables are defined before they are used.

variable_definition ::= "var" variable_name (ip_variable | port_variable)

ip_variable ::= {"!"} ip_address
port_variable ::= {"!"} port_range

Once defined, an IP address variable can be used by prepending its name with a dollar sign "$" wherever an IP address is expected. A port variable can be used wherever a port is expected, including in a subsequent variable definition.

In this section This section covers the following topics of information.

- Simple Snort Grammar Variable Use
- Snort Grammar Less Simple Variable Use
- Snort Grammar Complex Variable Use

4.3.1 Simple Snort Grammar Variable Use

Description The simple Snort grammar variable example in Table 4-4 is used to assign a meaningful name to a port number, which helps make the rule intention clear.

Table Table 4-4 shows simple Snort grammar variable use.

```
var SSH_PORT 22

accept tcp 192.168.0.0/16 any -> 192.168.0.1/32 $SSH_PORT
```

Table 4-4. Simple Snort Grammar Variable Use.

4.3.2 Snort Grammar Less Simple Variable Use

Description The less simple Snort grammar variable use example in Table 4-5 shows all external IP addresses are defined by taking the complement, logical negation, of the easily defined internal IP address space.

Table Table 4-5 shows less simple Snort grammar variable use.

```
var INTERNAL_NETWORK 192.168.0.0/16
var EXTERNAL_NETWORK !$INTERNAL_NETWORK

reject tcp $EXTERNAL_NETWORK any -> $INTERNAL_NETWORK 22
```

Table 4-5. Less Simple Snort Grammar Variable Use.

4.3.3 Snort Grammar Complex Variable Use

Description The Snort grammar complex variable use example in Table 4-6 is of several variables used to make the rules both independent of specific IP addresses and more readable.

Table Table 4-6 shows a complex Snort grammar variable use.

```
# Set up some variables.
var MY_HOST 192.168.1.24
var NOT_MY_HOST !192.168.1.24
var INTERNAL_NETWORK 192.168.0.0/16
var EXTERNAL_NETWORK !$INTERNAL_NETWORK
var SSH_PORT 22
var PROXY_PORTS 80:81

# Test rules to exercise the parser.
accept ip $INTERNAL_NETWORK any -> $MY_HOST any
accept icmp $INTERNAL_NETWORK any -> $MY_HOST any
accept tcp $INTERNAL_NETWORK any -> $MY_HOST any
accept udp $INTERNAL_NETWORK any -> $MY_HOST any
reject ip $INTERNAL_NETWORK any -> $MY_HOST any
reject icmp $INTERNAL_NETWORK any -> $MY_HOST any
reject tcp $INTERNAL_NETWORK any -> $MY_HOST any
reject udp $INTERNAL_NETWORK any -> $MY_HOST any

# Port numbers and negations.
accept ip $INTERNAL_NETWORK $SSH_PORT -> $MY_HOST $PROXY_PORTS
accept ip $INTERNAL_NETWORK !$SSH_PORT -> $MY_HOST $PROXY_PORTS
accept ip $INTERNAL_NETWORK $SSH_PORT -> $MY_HOST !$PROXY_PORTS
```

Table 4-6. Complex Snort Grammar Variable Use.

4.4 Snort Command-line Options and Flags

Description There are a number of command-line flags and options recognised by the Snort Rule Compiler.

Command-line flags. The following table explains the short and long option command-line flags recognized by the Snort Rule Compiler.

Short Option	Long Option	Explanation
-i	**--infile**	Followed by an input file name which contains Snort-like rules, one per line. If this option is not present the rules are read from standard input.
-o	**--outfile**	Followed by name of output file to be written with filters, one per line. If this option is not present the output filters are written to standard output. If the specified file exists, it will be overwritten, otherwise it will be created.
-a	**--accept**	If this flag is present the default filter added to the end of the output will accept all packets.
-r	**--reject**	If this flag is present the default filter added to the end of the output will reject all packets. This is the default.
-s	**--snap**	Followed by the number of bytes to be captured from the payload of the packet. This option sets a default snap-length for filters created from the rules which do not explicitly contain a snap-length. If this option is not present then rules which do not explicitly contain a snap-length will produce filters that capture entire packets.
-h **-?**	**--help**	If this flag is present the Snort Rule Compiler displays a help message and then exits.

5.0 TCPDUMP RULE COMPILER

Introduction

The Tcpdump Rule Compiler application forms part of the Endace Filter system.

The Tcpdump Rule Compiler takes a single tcpdump rule contained in a text file or passed in via standard input and produces a set of filters that correspond to that rule.

In this chapter

This chapter covers the following sections of information.

- Tcpdump Rule Compiler Examples
- Tcpdump Rule Compiler Grammar Specifications
- Tcpdump Grammar Command-line Options and Flags

5.1 Tcpdump Rule Compiler Examples

Description

A rule is specified using a tcpdump-like syntax [2] that specifies combinations of:

• Protocol [ICMP, IGRP, Raw/IP, TCP or UDP]
• Source and destination IP addresses
• TCP and UDP source and destination ports
• TCP flags [TCP]

In addition to explicitly specified rules, the compiler adds default rules for each protocol [ICMP, IGRP, TCP, UDP, IP] according to the following scheme:

1. For each Layer 4 protocol (ICMP, IGRP, TCP and UDP), if any rules were specified then a default rule will be added that has the opposite ‘sense’ to those rules. For example, if TCP rules are given that reject specific packets (“tcp and not port 80”) then a default rule will be added that accepts all other TCP packets.

2. The default rule for each Layer 4 protocol is added to the filters so that it is applied after all specific filters for that protocol.

3. A final accept/reject rule is added according to the settings given on the command-no command-line flag is given then this final filter will reject all packets.)

4. If the default rule for a Layer 4 protocol has the same sense (accept/reject) as the final catch-all rule, then it is omitted.

Continued on next page

5.1 Tcpdump Rule Compiler Examples, continued

In this section This section covers the following topics of information.

- Tcpdump Rule Compiler Simple Rule Example 1
- Tcpdump Rule Compiler Simple Rule Example 2
- Tcpdump Rule Compiler Less Simple Rule Example 1
- Tcpdump Rule Compiler Less Simple Rule Example 2
- Tcpdump Rule Compiler Complex Rule Example

5.1.1 Tcpdump Rule Compiler Simple Rule Example 1

Description Given the simple tcpdump rule in Table 5-1, only rules involving TCP potentially have values in the `tcp-flags` fields, and only rules involving TCP or UDP potentially have values in the `src-port` and `dst-port` fields.

In the following example in Table 5-1, because the port was not qualified with a `src` or `dst` prefix, two filters have been created. One catches port 80 in the source port of a TCP packet, the other catches port 80 in the destination port of a TCP packet.

The third filter is the default TCP filter. Because there was a TCP rule that excluded packets, the compiler has added a default TCP filter that accepts all other TCP packets.

The final filter in Table 5-1 is present because a default filter is always added to accept or reject packets that do not match any other rules. Unless the command-line option `--accept` is given, the default filter will reject packets.

For all Tcpdump Rule Compiler examples in the following sections of information, filter lines are wrapped to fit on the printed page.

The actual filters produced by the Tcpdump Rule Compiler are written one per line.

Continued on next page

5.1.1 Tcpdump Rule Compiler Simple Rule Example 1 ,continued

Table Table 5-1 shows a simple rule output of the Tcpdump Rule Compiler.

Rule:

```
ip and
(
     tcp and
     (
     not (port 80)
     )
)
```

Output:

```
# Filter file created by ./tcpdump_compiler at Thu Jul 15 08:35:13 2004.
  1 reject tcp src-ip {--------------------------------} src-port {0000000001010000}
              dst-ip {--------------------------------} dst-port {----------------} tcp-flags {--------}
  2 reject tcp src-ip {--------------------------------} src-port {----------------}
              dst-ip {--------------------------------} dst-port {0000000001010000} tcp-flags {--------}
  3 accept tcp src-ip {--------------------------------} src-port {----------------}
               dst-ip {--------------------------------} dst-port {----------------} tcp-flags {--------}
  4 reject  ip src-ip {--------------------------------} src-port {----------------}
              dst-ip {--------------------------------} dst-port {----------------} tcp-flags {--------}
```

Table 5-1. Tcpdump Simple Rule Output.

5.1.2 Tcpdump Rule Compiler Simple Rule Example 2

Description Given another simple rule Tcpdump Rule Compiler example in Table 5-2, because the port was not qualified with a `src` or `dst` prefix, filters have been created to catch both destination and source ports.

The third filter is the default TCP filter. Because there was a TCP rule that excluded packets, the compiler has added a default TCP filter that accepts all other TCP packets.

Table Table 5-2 shows a second example of a tcpdump simple rule output.

Rule:

```
ip and
(
     tcp and
     (
     not (port 80 and (tcp[13] & 2 = 0))
     )
)
```

Output:

```
  1 reject tcp src-ip {--------------------------------} src-port {0000000001010000}
              dst-ip {--------------------------------} dst-port {----------------} tcp-flags {------0-}
  2 reject tcp src-ip {--------------------------------} src-port {----------------}
             dst-ip {--------------------------------} dst-port {0000000001010000} tcp-flags {------0-}
  3 accept tcp src-ip {--------------------------------} src-port {----------------}
             dst-ip {--------------------------------} dst-port {----------------} tcp-flags {--------}
  4 reject  ip src-ip {--------------------------------} src-port {----------------}
              dst-ip {--------------------------------} dst-port {----------------} tcp-flags {--------}
```

Table 5-2. Second Example of Tcpdump Simple Rule Output.

5.1.3 Tcpdump Rule Compiler Less Simple Rule Example 1

Description Given the Tcpdump Rule Compiler example of a less simple rule in Table 5-3, because neither the port nor host was not qualified with a `src` or `dst` prefix, eight filters have been created for each combination of the port and two hosts.

The ninth filter is the default TCP filter. Because there was a TCP rule that excluded packets, the compiler has added a default TCP filter that accepts all other TCP packets.

Table Table 5-3 shows a Tcpdump Rule Compiler less simple rule output.

Rule:

```
ip and
(
     tcp and
     (
     not(port 80 and host (127.0.0.1 or 192.168.0.1))
     )
)
```

Output:

```
# Filter file created by ./tcpdump_compiler at Thu Jul 15 08:42:53 2004.

  1 reject tcp src-ip {11000000101010000000000000000001} src-port {----------------}
              dst-ip {--------------------------------} dst-port {0000000001010000} tcp-flags {--------}
  2 reject tcp src-ip {11000000101010000000000000000001} src-port {0000000001010000}
              dst-ip {--------------------------------} dst-port {----------------} tcp-flags {--------}
  3 reject tcp src-ip {--------------------------------} src-port {0000000001010000}
              dst-ip {11000000101010000000000000000001} dst-port {----------------} tcp-flags {--------}
  4 reject tcp src-ip {--------------------------------} src-port {----------------}
              dst-ip {11000000101010000000000000000001} dst-port {0000000001010000} tcp-flags {--------}
  5 reject tcp src-ip {01111111000000000000000000000001} src-port {----------------}
              dst-ip {--------------------------------} dst-port {0000000001010000} tcp-flags {--------}
  6 reject tcp src-ip {01111111000000000000000000000001} src-port {0000000001010000}
               dst-ip {--------------------------------} dst-port {----------------} tcp-flags {--------}
  7 reject tcp src-ip {--------------------------------} src-port {0000000001010000}
              dst-ip {01111111000000000000000000000001} dst-port {----------------} tcp-flags {--------}
  8 reject tcp src-ip {--------------------------------} src-port {----------------}
               dst-ip {01111111000000000000000000000001} dst-port {0000000001010000} tcp-flags {--------}
  9 accept tcp src-ip {--------------------------------} src-port {----------------}
              dst-ip {--------------------------------} dst-port {----------------} tcp-flags {--------}
  10 reject ip src-ip {--------------------------------} src-port {----------------}
              dst-ip {--------------------------------} dst-port {----------------} tcp-flags {--------}
```

Table 5-3. Tcpdump Rule Compiler Less Simple Rule Output.

5.1.4 Tcpdump Rule Compiler Less Simple Rule Example 2

Description Given the Tcpdump Rule Compiler example of a less simple rule in Table 5-4, because neither the port nor host were qualified with a `src` or `dst` prefix, filters have been created to catch all combinations of port and host.

The ninth filter is the default TCP filter. As there is a TCP rule excluding packets, the compiler added a default TCP filter that accepts all other TCP packets.

The tenth and eleventh filters are included because of the second clause in the tcpdump rule that specifically excluded IGRP and TCP packets. This implies that UDP and ICMP packets should be captured, so accept filters have been created for these protocols.

Table Table 5-4. Tcpdump Rule Compiler less simple rule output.

Rule:

```
    ip and
  (
     tcp and not
     (
        port 1234 and host (192.168.0.1 or 192.168.0.2)
     )
  )
  or
  (
    ip and not igrp and not tcp
  )
```

Output:

```
# Filter file created by ./tcpdump_compiler at Thu Jul 15 08:44:22 2004.

 1 reject tcp   src-ip {11000000101010000000000000000010} src-port {----------------}
                dst-ip {--------------------------------} dst-port {0000010011010010} tcp-flags {--------}
 2 reject tcp   src-ip {11000000101010000000000000000010} src-port {0000010011010010}
                dst-ip {--------------------------------} dst-port {----------------} tcp-flags {--------}
 3 reject tcp   src-ip {--------------------------------} src-port {0000010011010010}
                dst-ip {11000000101010000000000000000010} dst-port {----------------} tcp-flags {--------}
 4 reject tcp   src-ip {--------------------------------} src-port {----------------}
                dst-ip {11000000101010000000000000000010} dst-port {0000010011010010} tcp-flags {--------}
 5 reject tcp   src-ip {11000000101010000000000000000001} src-port {----------------}
                dst-ip {--------------------------------} dst-port {0000010011010010} tcp-flags {--------}
 6 reject tcp   src-ip {11000000101010000000000000000001} src-port {0000010011010010}
                dst-ip {--------------------------------} dst-port {----------------} tcp-flags {--------}
 7 reject tcp   src-ip {--------------------------------} src-port {0000010011010010}
                dst-ip {11000000101010000000000000000001} dst-port {----------------} tcp-flags {--------}
 8 reject tcp   src-ip {--------------------------------} src-port {----------------}
                dst-ip {11000000101010000000000000000001} dst-port {0000010011010010} tcp-flags {--------}
 9 accept tcp   src-ip {--------------------------------} src-port {----------------}
                dst-ip {--------------------------------} dst-port {----------------} tcp-flags {--------}
10 accept udp   src-ip {--------------------------------} src-port {----------------}
                dst-ip {--------------------------------} dst-port {----------------} tcp-flags {--------}
11 accept icmp  src-ip {--------------------------------} src-port {----------------}
                dst-ip {--------------------------------} dst-port {----------------} tcp-flags {--------}
12 reject ip    src-ip {--------------------------------} src-port {----------------}
                dst-ip {--------------------------------} dst-port {----------------} tcp-flags {--------}
```

Table-5-4. Tcpdump Rule Compiler Less Simple Rule Output.

5.1.5 Tcpdump Rule Compiler Complex Rule Example

Description

The complex rule in Table 5-5 is the result of reading "accept all TCP packets except those involving port 80 and host 127.0.0.80, or port 81 and host 127.0.0.81, and reject all UDP packets except those involving either port 3128 or port 8080".

Given the Tcpdump Rule Compiler complex rule example in Table 5-5, because neither the ports nor hosts were qualified with a `src` or `dst` prefix, filters have been created for each combination.

The ninth filter is the default TCP filter. Because there was a TCP rule that excluded packets, the compiler has added a default TCP filter that accepts all other TCP packets.

The fourteenth filter is the final catch-all filter. Because there was a UDP rule that included packets, the usual default UDP filter would have rejected UDP packets not matched by rules ten through thirteen. However, in this case the final catch-all filter rejects all packets and the default UDP filter was superfluous.

Continued on next page

5.1.5 Tcpdump Rule Compiler Complex Rule Example ,continued

Description (continued)

Table Table 5-5 Tcpdump Rule Compiler complex rule output.

Rule:

```
ip and
   (
      tcp and
      (
         not (port (80) and host 127.0.0.80)
         and not (port 81 and host (127.0.0.81))
      )
      or udp and
      (
         port (3128 or 8080)
      )
   )
```

Output:

```
# Filter file created by ./tcpdump_compiler at Thu Jul 15 08:45:21 2004.

  1 reject tcp  src-ip {01111111000000000000000001010000} src-port {----------------}
                dst-ip {--------------------------------} dst-port {0000000001010000} tcp-flags {--------}
  2 reject tcp  src-ip {01111111000000000000000001010000} src-port {0000000001010000}
                dst-ip {--------------------------------} dst-port {----------------} tcp-flags {--------}
  3 reject tcp  src-ip {--------------------------------} src-port {0000000001010000}
                dst-ip {01111111000000000000000001010000} dst-port {----------------} tcp-flags {--------}
  4 reject tcp  src-ip {--------------------------------} src-port {----------------}
                dst-ip {01111111000000000000000001010000} dst-port {0000000001010000} tcp-flags {--------}
  5 reject tcp  src-ip {01111111000000000000000001010001} src-port {----------------}
                dst-ip {--------------------------------} dst-port {0000000001010001} tcp-flags {--------}
  6 reject tcp  src-ip {01111111000000000000000001010001} src-port {0000000001010001}
                dst-ip {--------------------------------} dst-port {----------------} tcp-flags {--------}
  7 reject tcp  src-ip {--------------------------------} src-port {0000000001010001}
                dst-ip {01111111000000000000000001010001} dst-port {----------------} tcp-flags {--------}
  8 reject tcp  src-ip {--------------------------------} src-port {----------------}
                dst-ip {01111111000000000000000001010001} dst-port {0000000001010001} tcp-flags {--------}
  9 accept tcp  src-ip {--------------------------------} src-port {----------------}
                dst-ip {--------------------------------} dst-port {----------------} tcp-flags {--------}
 10 accept udp src-ip {--------------------------------} src-port {0001111110010000}
                dst-ip {--------------------------------} dst-port {----------------} tcp-flags {--------}
 11 accept udp src-ip {--------------------------------} src-port {----------------}
                dst-ip {--------------------------------} dst-port {0001111110010000} tcp-flags {--------}
 12 accept udp src-ip {--------------------------------} src-port {0000110000111000}
                dst-ip {--------------------------------} dst-port {----------------} tcp-flags {--------}
 13 accept udp src-ip {--------------------------------} src-port {----------------}
                dst-ip {--------------------------------} dst-port {0000110000111000} tcp-flags {--------}
 14 reject  ip src-ip {--------------------------------} src-port {----------------}
                dst-ip {--------------------------------} dst-port {----------------} tcp-flags {--------}
```

Table 5-5. Tcpdump Rule Compiler Complex Rule Output.

5.2 Tcpdump Rule Compiler Grammar Specifications

Description A formal specification for the Tcpdump Rule Compiler grammar is:

- A pipe character (|) indicates choice
- Speech marks (") indicate literal data
- Braces ({}) indicate optional elements
- em-dash (--) indicates a range
- Literal exclamation mark (!) is used in a rule to indicate a logical negation

Grammar The following list describes the Tcpdump Rule Compiler grammar.

Tcpdump Rule Compiler Grammar

```
rule ::= "ip" "and" protocol_list
        | "ip" "and" protocol_reject_list
        | rule "or" rule
        | "(" rule ")"

protocol_reject_list ::= single_protocol_reject
        | single_protocol_reject "and" protocol_reject_list
        | "(" protocol_reject_list ")"

single_protocol_reject ::= "not" "udp"
        | "not" "tcp"
        | "not" "igrp"
        | "(" single_protocol_reject ")"

protocol list ::= protocol_tree
        | protocol_tree "or" protocol_list
        | "(" protocol_list ")"
protocol tree ::= "tcp" tcp_tree
        | "udp" udp_tree
        | "icmp" icmp_tree
        | "ip" "and" protocol_reject_list
        | "(" protocol_tree ")"
```

Continued on next page

5.2 Tcpdump Rule Compiler Grammar Specifications, continued

Tcpdump Rule Compiler Grammar (continued)

Tcpdump Rule Compiler Grammar

```
icmp_tree ::=
        | "and" icmp_tree
        | "(" icmp_tree ")"

udp_tree ::=
        | "not" udp_reject_tree
        | udp_accept_tree
        | "and" udp_tree
        | "(" udp_tree ")"

udp_reject tree ::= udp_expression
        | udp_expression "and" "not" udp_reject_tree

udp_accept tree ::= udp expression
        | udp_expression "or" udp_accept_tree

udp expression ::= udp clause
        | udp_expression "and" udp_expression
        | "(" udp_expression ")"

udp clause ::= port_primitive
        | host_primitive

tcp tree ::=
        | "not" tcp_reject_tree
        | tcp_accept_tree
        | "and" tcp_tree
        | "(" tcp_tree ")"

tcp reject tree ::= tcp expression
        | tcp_expression "and" "not" tcp_reject_tree

tcp accept tree ::= tcp expression
        | tcp_expression "or" tcp_accept_tree

tcp expression ::= tcp_clause
        | "(" tcp_and_expression ")"
        | "(" tcp_or_expression ")"
```

Continued on next page

5.2 Tcpdump Rule Compiler Grammar Specifications, continued

Tcpdump Rule Compiler Grammar (continued)

Tcpdump Rule Compiler Grammar

```
tcp_and-expression ::= tcp_expression
        | tcp_expression "and" tcp_expression
        | "(" tcp_or_expression "or" tcp_or_expression ")"
        | "not" tcp_clause

tcp_or_expression ::= tcp_expression
        | tcp_or_expression "or" tcp_or_expression
        | "(" tcp_and_expression "and" tcp_and_expression ")"
        | "not" tcp_clause

tcp_clause ::= port_primitive
        | host_primitive
        | tcp_flags_primitive

qualifiers ::=
        | "src"
        | "dst

host_primitive ::= qualifiers host_keyword host_list
        | qualifiers host keyword "(" host list "and" host list ")"
        | "(" qualifiers host_keyword host_list "and" qualifiers_host
          keyword_host list ")"

host keyword ::=
        | "host"
        | "net"

host list ::= single_host
        | single_host "or" host_list
        | "(" host_list ")"

single host ::= hostname
        | netname2
        | netname3
        | "(" single_host ")"

hostname ::= "1--255.0--255.0--255.0--255"

netname2 ::= "1--255.0--255"

netname3 ::= "1--255.0--255.0--255"
```

Continued on next page

5.2 Tcpdump Rule Compiler Grammar Specifications, continued

Tcpdump Rule Compiler Grammar (continued)

Tcpdump Rule Compiler Grammar

```
port_primitive ::= qualifiers "port" port_list
        | qualifiers "port" "(" port_list "and" port_list ")"
        | "(" port_primitive ")"

port_list ::= number
        | number "or" port_list
        | "(" port_list ")"

number ::= "0--65535"
        | "(" "0--65535" ")"

tcp_flags primitive ::= "tcp[13]" "&" number tcp_flags_relop
number
        | "(" tcp_flags_primitive ")"

tcp_flags_relop ::= "="
        | "!="
```

5.3 Tcpdump Grammar Command-line Options and Flags

Description

There are a number of command-line flags and options recognised by the Tcpdump Rule Compiler.

The Tcpdump Rule Compiler performs little optimization on the generated filters, and so the number of filters created may be greater than strictly necessary.

For example, in some cases it may be possible to combine filters that differ only in a few bit locations by using "don’t care" entries in those locations. An optimization pass will be included in a future revision.

Command-line flags.

The following table explains the short option and long option command-line flags recognized by the Tcpdump Rule Compiler.

Short Option	Long Option	Explanation
-i	**--infile**	Followed by an input file name which contains a single tcpdump-like rule. If this option is not present the rules are read from standard input.
-o	**--outfile**	Followed by name of output file to be written with filters, one per line. If this option is not present the output filters are written to standard output. If the specified file exists, it will be overwritten, otherwise it will be created.
-a	**--accept**	If this flag is present the default filter added to the end of the output will accept all packets.
-r	**--reject**	If this flag is present the default filter added to the end of the output will reject all packets. This is the default.
-s	**--obfuscate**	If this flag is present then the IP addresses and port numbers in the input rule will be obfuscated before the rule is processed. The result of the obfuscation is written to the file `obfuscated_rule_N.txt` in the current working directory, where `N` is the first positive integer that makes the filename unique. This enables the obfuscated filters to be compared to an obfuscated rule for accuracy.
-h	**--usage**	
-?	**--help**	If this flag is present the Tcpdump Rule Compiler displays a help message and then exits.