endace
accelerated

filter_loader Software Guide

EDM04-28

Protection Against Harmful Interference

When present on equipment this manual pertains to, the statement "This device complies with part 15 of the FCC rules" specifies the equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the Federal Communications Commission [FCC] Rules.

These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment.

This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications.

Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at their own expense.

Extra Components and Materials

The product that this manual pertains to may include extra components and materials that are not essential to its basic operation, but are necessary to ensure compliance to the product standards required by the United States Federal Communications Commission, and the European EMC Directive. Modification or removal of these components and/or materials, is liable to cause non compliance to these standards, and in doing so invalidate the user's right to operate this equipment in a Class A industrial environment.

Disclaimer

Whilst every effort has been made to ensure accuracy, neither Endace Technology Limited nor any employee of the company, shall be liable on any ground whatsoever to any party in respect of decisions or actions they may make as a result of using this information.

Endace Technology Limited has taken great effort to verify the accuracy of this manual, but nothing herein should be construed as a warranty and Endace shall not be liable for technical or editorial errors or omissions contained herein.

In accordance with the Endace Technology Limited policy of continuing development, the information contained herein is subject to change without notice.

Website

http://www.endace.com

Copyright 2008 Endace Technology Ltd. All rights reserved.

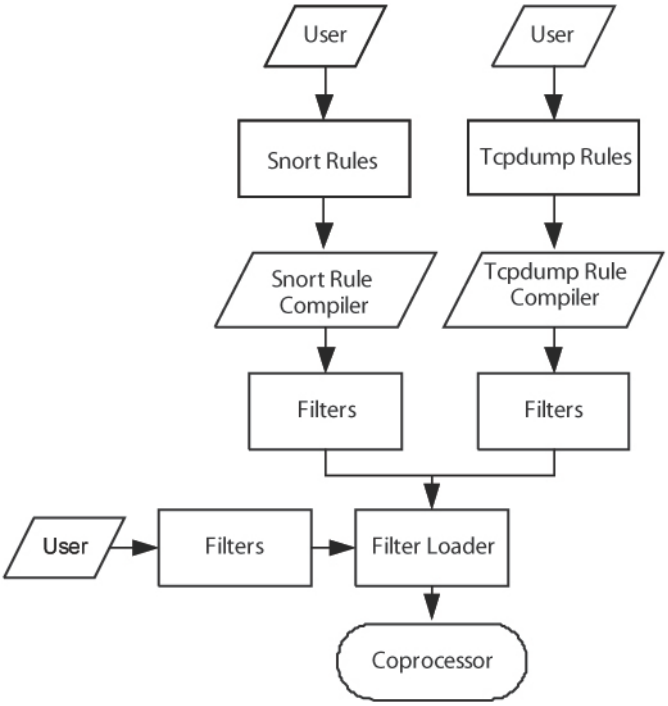
Contents

filter_loader software

The `filter_loader`, `snort_compiler` and `tcpdump_compiler` are command-line applications are used to program the filtering capabilities of the Co-Processor. Libpcap or the native DAG API application deals with capturing packets.

A filter rule set can be:

- manually written in the Endace filter format,
- generated from Snort-style rules by the `snort_compiler` application, or
- generated from tcpdump-style rules by the `tcpdump_compiler` application.



Requirements

The requirements for using the `filter_loader` are:

- An DAG card with Co-Processor.
- DAG software (3.3.1 or greater).
  Customers with a current support contract can download this from the secure Endace website: https://www.endace.com/support.
  Refer to *EDM04-01 DAG Software Installation Guide* for details on how to install and compile the DAG software.

filter_loader operation

The `filter_loader` loads a set of filters (from a text file or standard input) into a Co-Processor equipped DAG card running IP Filter firmware. The filter set determines what packets are captured. The packets are captured in ERF.

A filter rule set can be:

- manually written in the Endace filter format,
- 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 IP Filter firmware can support up to two filter sets per interface, one active and one inactive.

When a new filter rule set is loaded, the `filter_loader` can:

- restart the card and make the new filter set active, or
- load the new filter set in an inactive state and instruct the card to switch between the active and inactive filter sets after the new set is loaded. This allows the filter set to be dynamically modified with zero packet loss.

The following shows the size of the filter sets for different card configurations.

Interfaces	Hot-Swap Ability	SC128 Filters Per Set	SC256 Filters Per Set	Built in Filters Per Set
1	No	16k	32k	32k
1	Yes	8k	16k	16k
2	No	8k	16k	16k
2	Yes	4k	8k	8k

filter_loader operation example

The following are examples of how to load a filter set into a Co-Processor.

- How to initialize a new rule set for Interface 1:

```
filter_loader -d0 --initialize --init-ports 2 --init-rulesets 2 --iface 1 -l
ethernet -m color -i test-1-1-ifc-1.rule
```

- How to load the rule set for interface 0:

```
filter_loader -d0 --iface 0 -l ethernet -m color -i test-1-1-ifc-0.rule
```

- How to switch the rule set for both interfaces:

```
filter_loader -d0 --iface 0 -l ethernet -m color -i test-1-2-ifc-0.rule
filter_loader -d0 --iface 1 -l ethernet -m color -i test-1-2-ifc-1.rule
```

filter_loader options

The following table explains the `filter_loader` command line short and long options.

Short Option	Long Option	Explanation
`-d`	`--device`	Followed by device name of the DAG card to configure, such as `d0`.
`-l`	`--linktype`	Followed by the type of link being monitored. Valid options are 'ethernet', 'pos4chdlc' and 'pos4ppp'.
`-m`	`--mapping`	Where to place the packet classification in the received packets. Valid values are: 'color', 'colour' 'rxerror', 'lcntr', 'flags', 'hdlcheader' (PoS links only), and 'padoffset' (Ethernet links only). Also 'padoffset0', 'hdlcheader0', and 'colour0'. For further details, see *Options for –m* (page 2a). 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.
	`--initialise` `--initialize`	If this flag is present then the Co-Processor is initialized before filters are loaded. If this flag is not present, then the filters will be hot-swapped with the supplied filters.
	`--init-ports` `--init-ifaces`	Followed by the number of interfaces. This value is used to configure the Co-Processor 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.
	`--init-rulesets`	Followed by the number of rulesets per interface. This value is used to configure the Co-Processor 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.

Options for –m

The `--mapping` option to filter loader determines what ERF format is used for capture to the host.

The effect of the `--mapping` 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
`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

For further details on ERF types, see *EDM11-01 ERF types*.

Filter rules

Filter rules are 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 rules can be grouped together into sets. Filter rule sets are loaded into the Co-Processor in the order they are presented, and the ultimate filter should be a catch-all accept or reject filter. IP Filter supports filtering on:

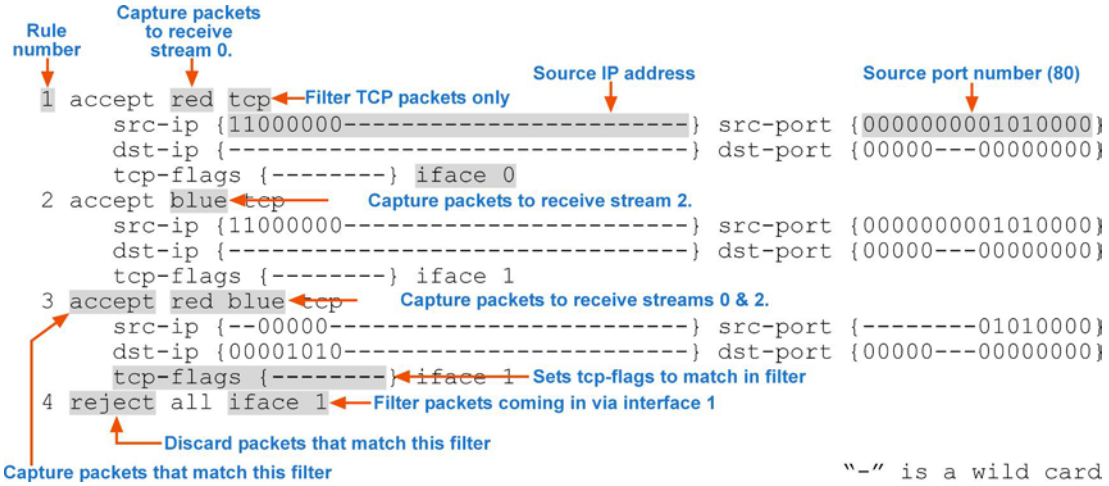
- Ingress interface
- Protocol [ICMP, IGRP, RawIP/TCP or UDP]
- Source and destination IP addresses
- TCP and UDP source and destination port numbers
- TCP flags

Each bit in the IP address, port number, and TCP flags fields in the filter rules can take values "0", "1" or "-" (wildcard). The *classification* of the packet is an integer in the range 0 to 16383 which is written into the ERF records. For details on which ERF type is received see *Options for –m* (page).

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 zero	3 = both buffers
2 = buffer two	0 = neither buffer

Filter rule example



Note: The above example has been wrapped onto several lines for best presentation. In the filter set each rule **must** be on a single line.

Filter rule keywords

Keyword	Description
`accept`	Sets the filter to capture all packets that match the following filter.
`all`	Sets the filter to look at all Layer 3 protocol field (IPv4).
`blue`	Sets the filter to steer packets to receive stream 2. One of two receive streams used with packet steering. Accessed through the DAG API as receive stream 2. For example, for a DAG card identified as `dag0`, the blue memory buffer containing the blue packets can be referred to as `dag0:2` when using the standard DAG utilities. 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. See also `red`.
`dst-ip`	A 32 bit binary representation of the destination IP address. An 8 bit representation for each part of the IP address. Can be either "`0`", "`1`" or "`-`" (wildcard). Examples: 192.168.0.1 = `11000000101010000000000000000001` 192.168.0.0/28 = `11000000101010000000000000000----`
`dst-port`	16 bit binary representation of the destination port. Can be either "`0`", "`1`" or "`-`" (wildcard).
`icmp`	Sets the filter to look for `icmp` packets when processing this filter.
`iface <num>`	Sets the filter to look at packets that match the selected interface. If an interface is specified for any filter, 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 • apply the filters to the interfaces specified on a per-filter basis. 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. If a default reject filter is used, it must be included for each interface.
`igrp`	Sets the filter to look for `igrp` packets when processing this filter.
`ip`	Sets the filter to capture IPv4 packets. For Ethernet or cHDLC the value will be 0x0800 and for PPP the value will be 0x0021.
`ip-proto`	Sets the filter to look for the specified Layer 3 protocol field (IPv4 header protocol)
`l2-proto`	Sets the filter to look for the specified layer 2 protocol field (Ether type, PPP or cHDLC protocols).
`non-ip`	Sets the filter to look for all layer 2 protocol fields.
`red`	Sets the filter to steer packets to receive stream 0. One of two receive streams used with packet steering. Accessed through the DAG API as receive stream 0. For example, for a DAG card identified as `dag0`, the red memory buffer containing the red packets can be referred to as `dag0:0` when using the standard DAG utilities. 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. See also `blue`.
`reject`	Discards any packets that match the rest of the filter. To have a default reject filter in place, you must included one for each interface.
`src-ip`	A 32 bit binary representation of the source IP address. An 8 bit representation for each part of the IP address. Can be either "`0`", "`1`" or "`-`" (wildcard). Example: 192.168.0.2 = `11000000101010000000000000000010`
`src-port`	16 bit binary representation of the source port. Can be either "`0`", "`1`" or "`-`" (wildcard).
`tcp`	Sets the filter to look for `tcp` packets (Layer 3 (IPv4) protocol).

`tcp-flags`	Sets which tcp-flags the filter needs to look at when processing this filter. Can be either "0", "1" or "-" (wildcard). The tcp-flags are from (left to right): • C, CWR • E, ECE, ECN-Echo • U, URG – Urgent pointer valid flag • A, ACK - Acknowledgment number valid flag. • P, PSH - Push flag. • R, RST - Reset connection flag. • S, SYN - Synchronize sequence numbers flag. • F, FIN - End of data flag. **Note:** If the tcp-flags field is not present for a TCP rule, then it is considered to be all "wildcard" entries. The tcp-flags field may be present for non-TCP rules, in which case it is ignored.
`udp`	Sets the filter to look for `udp` packets when processing this filter.

Note: Filters rules are evaluated in from the top of the filter rule set to the bottom.
Missing tokens are assumed to be wildcards.
The first matching filter is the one selected.

Note: Layer 2 protocol field is Ether type in most networks.
Layer 3 protocol field is IPv4 protocol type.

Filter examples

Note: The following examples have been wrapped onto several lines for best presentation.
In the filter rule set each rule **must** be on a single line.

Example 1

The following filter rule set captures TCP packets with:

- a source IP address of 192.168.0.1 sent from any source port
- to the destination IP address 192.168.0.2 on port 80,
- all other packets are rejected.

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

Example 2

The following filter rule set captures TCP packets with:

- a source IP address in the subnet 192.168.0.0/16 and any port
- to the IP address 192.168.0.2 on ports 80 or 81,
- all other packets are rejected

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

Example 3

The following filter rule set 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,
- all other packets are rejected

The filters are evaluated in order. In the following example, 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.

```
1 reject tcp src-ip   {11000000101010000000000100000001}
             src-port {----------------}
             dst-ip   {--------------------------------}
             dst-port {----------------}
2 reject tcp src-ip   {--------------------------------}
             src-port {----------------}
             dst-ip   {11000000101010000000000100000001}
             dst-port {----------------}
3 accept tcp src-ip   {1100000010101000----------------}
             src-port {----------------}
             dst-ip   {11000000101010000000000100000010}
             dst-port {0000000000101000-}
4 reject all src-ip   {--------------------------------}
             src-port {----------------}
             dst-ip   {--------------------------------}
             dst-port {----------------}
```

Example 4

The following TCP flags filter rule set captures TCP packets from:

- any source to any destination that have the SYN flag set.
- all other packets are rejected

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

Example 5

The following filter rule set captures TCP packets

- from 192.168.0.1 to 192.168.0.2 on interface 0 with a destination IP address of 192.168.0.2 on port 80
- and TCP packets from 192.168.0.3 to 192.168.0.4 on interface 1 with a destination IP address of 192.168.0.4 on port 80
- all other packets are rejected

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

Example 6

The following filter rule set sends all TCP packets to

- the red receive stream, receive stream 0, and
- all UDP packets to the blue receive stream, receive stream 2.
- all other packets are rejected

```
1 accept red  tcp
2 accept blue udp
3 reject      all
```


The `snort_compiler` application forms part of the IP Filter system.

Snort is an Open Source Network *Intrusion Detection System* [IDS] 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.

snort_compiler options

The following explains the short and long options recognized by `snort_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`	**Note:** This option is Obsolete - Do not use. 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.

snort_compiler examples

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 pre-defined patterns that could be malicious.

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

Example 1

The following Snort rule results in the output from the Snort Rule Compiler. The Snort Rule Compiler expresses both destination ports by using a "wildcard" 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.

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 {----------------}
```

Example 2

In the following Snort rule, 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 "wildcard" entry.

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   {11000000101010000000000-100000001}
             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 {----------------}
```

Example 3

The following Snort rule output 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 rule output are shown in below.

The Snort Rule Compiler encoded the eleven destination port numbers as three entries covering ports 80–87 with three "wildcard" bits, 88–89 with one "wildcard" bit, and 90.

Rule:

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

Output:

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

snort_compiler grammar

A formal specification for the `snort_compiler` 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.

snort_compiler grammar variables

Snort grammar supports the 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.

Example 1

The following Snort grammar variable example is used to assign a meaningful name to a port number, which helps make the rule intention clear.

```
var SSH_PORT 22
accept tcp 192.168.0.0/16 any -> 192.168.0.1/32 $SSH_PORT
```

Example 2

The following Snort grammar variable example shows all external IP addresses are defined by taking the complement, logical negation, of the easily defined internal IP address space.

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

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

Example 3

The following Snort grammar variable example is used to make the rules both independent of specific IP addresses and more readable.

```
# 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
```


The `tcpdump_compiler` application forms part of the IP Filter system.

The `tcpdump_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.

tcpdump_compiler options

There are a number of options recognized by the `tcpdump_compiler`.

The `tcpdump_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 "wildcard" entries in those locations. An optimization pass will be included in a future revision.

The following table explains the short and long options recognized by the `tcpdump_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_compiler` displays a help message and then exits.

Perform Self-test

The `tcpdump_compiler` comes with a self-test script that can be used to verify the basic functionality of the binary.

Procedure

To perform the self-test, complete the following steps:

1. Change to the following directory:
   `dag-<dag_version>/filtering/tcpdump_compiler`
2. Run `run_tests.sh` script **in the directory**:
   `run_tests.sh`
   The following message is displayed indicating all 32 tests succeeded:
   `End of tests`

tcpdump_compiler grammar specifications

A formal specification for the `tcpdump_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_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 ")"
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 ")"
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"

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

tcpdump_compiler examples

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 (if 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.

Note: For all `tcpdump_compiler` examples in the following sections, filter lines are wrapped to fit on the printed page. The actual filters produced by the `tcpdump_compiler` are written one per line.

Example 1

In the following example, two filters have been created because the port was not qualified with a `src` or `dst` prefix. 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 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.

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 {--------}
```

Example 2

In the following rule, filters have been created to catch both destination and source ports because the port was not qualified with a `src` or `dst` prefix,.

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.

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 {--------}
```

Example 3

In the following rule, eight filters have been created for each combination of the port and two hosts, because neither the port nor host was not qualified with a `src` or `dst` prefix,

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.

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 {--------}
```

Example 4

In the following example, filters have been created to catch all combinations of port and host 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.

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 {--------}
```

Example 5

The following example 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".

In the following example, filters have been created for each combination because neither the ports nor hosts were qualified with a `src` or `dst` prefix.

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.

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 {--------}
```

Version History

Version	Date	Reason
1	November 2008	Split from 02-02 Co-Processor IP Filter Software Manual because this information referenced by two documents now EDM02-02 and EDM 04-26.