03-02-2022-24064301-Hauptpost-0559 03-02-2022-24064301-HauPtPost-0559

Anlage D9

Cisco Systems GmbH ./. Centripetal Networks, Inc.

Streitgebrauchsmuster:
DE 20 2016 008 885.9

BARDEHLE PAGENBERG
Unser Zeichen: C159362WOEPGBMA1Lö5



SNORT® Users Manual
2.9.0

The Snort Project

March 25, 2011

Copyright ©1998-2003 Martin Roesch
Copyright ©2001-2003 Chris Green
Copyright ©2003-2011 Sourcefire, Inc.



03-02-2022-24064301-HauptPost-0560

Contents

1 Snort Overview 9

1.1 Getting Started . . . 9
1.2 Sniffer Mode . . . 9
1.3 Packet Logger Mode . . . 10
1.4 Network Intrusion Detection System Mode . . . 11
1.4.1 NIDS Mode Output Options . . . 11
1.4.2 Understanding Standard Alert Output . . . 12
1.4.3 High Performance Configuration . . . 12
1.4.4 Changing Alert Order . . . 13
1.5 Packet Acquisition . . . 13
1.5.1 Configuration . . . 13
1.5.2 PCAP . . . 14
1.5.3 AFPACKET . . . 15
1.5.4 NFQ . . . 15
1.5.5 IPQ . . . 16
1.5.6 IPFW . . . 16
1.5.7 Dump . . . 16
1.5.8 Statistics Changes . . . 17
1.6 Reading Pcaps . . . 17
1.6.1 Command line arguments . . . 17
1.6.2 Examples . . . 17
1.7 Basic Output . . . 19
1.7.1 Timing Statistics . . . 19
1.7.2 Packet I/O Totals . . . 19
1.7.3 Protocol Statistics . . . 20
1.7.4 Actions, Limits, and Verdicts . . . 21
1.8 Tunneling Protocol Support . . . 22
1.8.1 Multiple Encapsulations . . . 22
1.8.2 Logging . . . 22
1.9 Miscellaneous . . . 23

16444709564601258531 9

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 3

1.9.1 Running Snort as a Daemon . . . 23
1.9.2 Running in Rule Stub Creation Mode . . . 23
1.9.3 Obfuscating IP Address Printouts . . . 24
1.9.4 Specifying Multiple-Instance Identifiers . . . 24
1.9.5 Snort Modes . . . 24
1.10 More Information . . . 25

2 Configuring Snort **26**
2.1 Includes . . . 26
2.1.1 Format . . . 26
2.1.2 Variables . . . 26
2.1.3 Config . . . 29
2.2 Preprocessors . . . 38
2.2.1 Frag3 . . . 38
2.2.2 Stream5 . . . 41
2.2.3 sfPortscan . . . 46
2.2.4 RPC Decode . . . 51
2.2.5 Performance Monitor . . . 52
2.2.6 HTTP Inspect . . . 55
2.2.7 SMTP Preprocessor . . . 67
2.2.8 FTP/Telnet Preprocessor . . . 70
2.2.9 SSH . . . 76
2.2.10 DNS . . . 78
2.2.11 SSL/TLS . . . 78
2.2.12 ARP Spoof Preprocessor . . . 80
2.2.13 DCE/RPC 2 Preprocessor . . . 81
2.2.14 Sensitive Data Preprocessor . . . 94
2.2.15 Normalizer . . . 97
2.3 Decoder and Preprocessor Rules . . . 100
2.3.1 Configuring . . . 100
2.3.2 Reverting to original behavior . . . 101
2.4 Event Processing . . . 101
2.4.1 Rate Filtering . . . 102
2.4.2 Event Filtering . . . 103
2.4.3 Event Suppression . . . 105
2.4.4 Event Logging . . . 106
2.5 Performance Profiling . . . 107
2.5.1 Rule Profiling . . . 107
2.5.2 Preprocessor Profiling . . . 109



03-02-2022-24064301-HauptPost-0561

2.5.3 Packet Performance Monitoring (PPM) . . . 111
2.6 Output Modules . . . 114
2.6.1 alert_syslog . . . 115
2.6.2 alert_fast . . . 116
2.6.3 alert_full . . . 117
2.6.4 alert_unixsock . . . 117
2.6.5 log_tcpdump . . . 117
2.6.6 database . . . 118
2.6.7 csv . . . 119
2.6.8 unified . . . 120
2.6.9 unified 2 . . . 121
2.6.10 alert_prelude . . . 121
2.6.11 log null . . . 122
2.6.12 alert_aruba_action . . . 122
2.6.13 Log Limits . . . 123
2.7 Host Attribute Table . . . 123
2.7.1 Configuration Format . . . 123
2.7.2 Attribute Table File Format . . . 124
2.7.3 Attribute Table Example . . . 126
2.8 Dynamic Modules . . . 127
2.8.1 Format . . . 127
2.8.2 Directives . . . 127
2.9 Reloading a Snort Configuration . . . 128
2.9.1 Enabling support . . . 128
2.9.2 Reloading a configuration . . . 128
2.9.3 Non-reloadable configuration options . . . 129
2.10 Multiple Configurations . . . 130
2.10.1 Creating Multiple Configurations . . . 130
2.10.2 Configuration Specific Elements . . . 131
2.10.3 How Configuration is applied? . . . 132
2.11 Active Response . . . 132
2.11.1 Enabling Active Response . . . 133
2.11.2 Configure Sniping . . . 133
2.11.3 Flexresp . . . 133
2.11.4 React . . . 134
2.11.5 Rule Actions . . . 135

16444709564601258531 9

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 5

3 **Writing Snort Rules** 136

3.1 The Basics . . . 136

3.2 Rules Headers . . . 136

3.2.1 Rule Actions . . . 136

3.2.2 Protocols . . . 137

3.2.3 IP Addresses . . . 137

3.2.4 Port Numbers . . . 138

3.2.5 The Direction Operator . . . 138

3.2.6 Activate/Dynamic Rules . . . 139

3.3 Rule Options . . . 139

3.4 General Rule Options . . . 140

3.4.1 msg . . . 140

3.4.2 reference . . . 140

3.4.3 gid . . . 141

3.4.4 sid . . . 141

3.4.5 rev . . . 142

3.4.6 classtype . . . 142

3.4.7 priority . . . 143

3.4.8 metadata . . . 144

3.4.9 General Rule Quick Reference . . . 144

3.5 Payload Detection Rule Options . . . 145

3.5.1 content . . . 145

3.5.2 nocase . . . 146

3.5.3 rawbytes . . . 146

3.5.4 depth . . . 147

3.5.5 offset . . . 147

3.5.6 distance . . . 148

3.5.7 within . . . 148

3.5.8 http_client_body . . . 149

3.5.9 http_cookie . . . 149

3.5.10 http_raw_cookie . . . 150

3.5.11 http_header . . . 150

3.5.12 http_raw_header . . . 151

3.5.13 http_method . . . 151

3.5.14 http_uri . . . 151

3.5.15 http_raw_uri . . . 152

3.5.16 http_stat_code . . . 152

3.5.17 http_stat_msg . . . 153

3.5.18 http_encode . . . 153

1644470956460125853 19

3.5.19 fast_pattern . . . 154
3.5.20 uricontent . . . 155
3.5.21 urilen . . . 156
3.5.22 isdataat . . . 157
3.5.23 pcre . . . 157
3.5.24 file_data . . . 158
3.5.25 base64_decode . . . 159
3.5.26 base64_data . . . 160
3.5.27 byte_test . . . 161
3.5.28 byte_jump . . . 162
3.5.29 byte_extract . . . 163
3.5.30 ftpbounce . . . 164
3.5.31 asn1 . . . 164
3.5.32 cvs . . . 165
3.5.33 dce_iface . . . 165
3.5.34 dce_opnum . . . 165
3.5.35 dce_stub_data . . . 165
3.5.36 ssl_version . . . 165
3.5.37 ssl_state . . . 165
3.5.38 Payload Detection Quick Reference . . . 165
3.6 Non-Payload Detection Rule Options . . . 166
3.6.1 fragoffset . . . 166
3.6.2 ttl . . . 166
3.6.3 tos . . . 167
3.6.4 id . . . 167
3.6.5 ipopts . . . 168
3.6.6 fragbits . . . 168
3.6.7 dsize . . . 169
3.6.8 flags . . . 169
3.6.9 flow . . . 170
3.6.10 flowbits . . . 170
3.6.11 seq . . . 171
3.6.12 ack . . . 171
3.6.13 window . . . 171
3.6.14 itype . . . 172
3.6.15 icode . . . 172
3.6.16 icmp_id . . . 172
3.6.17 icmp_seq . . . 172
3.6.18 rpc . . . 173

164447095646012585319

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 7

3.6.19 ip_proto . . . 173
3.6.20 sameip . . . 173
3.6.21 stream_reassemble . . . 174
3.6.22 stream_size . . . 174
3.6.23 Non-Payload Detection Quick Reference . . . 174
3.7 Post-Detection Rule Options . . . 175
3.7.1 logto . . . 175
3.7.2 session . . . 175
3.7.3 resp . . . 176
3.7.4 react . . . 176
3.7.5 tag . . . 176
3.7.6 activates . . . 177
3.7.7 activated_by . . . 177
3.7.8 count . . . 177
3.7.9 replace . . . 177
3.7.10 detection_filter . . . 177
3.7.11 Post-Detection Quick Reference . . . 178
3.8 Rule Thresholds . . . 178
3.9 Writing Good Rules . . . 179
3.9.1 Content Matching . . . 179
3.9.2 Catch the Vulnerability, Not the Exploit . . . 179
3.9.3 Catch the Oddities of the Protocol in the Rule . . . 179
3.9.4 Optimizing Rules . . . 180
3.9.5 Testing Numerical Values . . . 181

4 Dynamic Modules **184**
4.1 Data Structures . . . 184
4.1.1 DynamicPluginMeta . . . 184
4.1.2 DynamicPreprocessorData . . . 184
4.1.3 DynamicEngineData . . . 185
4.1.4 SFSnortPacket . . . 185
4.1.5 Dynamic Rules . . . 185
4.2 Required Functions . . . 190
4.2.1 Preprocessors . . . 191
4.2.2 Detection Engine . . . 191
4.2.3 Rules . . . 192
4.3 Examples . . . 192
4.3.1 Preprocessor Example . . . 192
4.3.2 Rules . . . 194

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 8

5 Snort Development **197**

5.1 Submitting Patches 197

5.2 Snort Data Flow 197

5.2.1 Preprocessors 197

5.2.2 Detection Plugins 197

5.2.3 Output Plugins 197

5.3 The Snort Team 198



Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 9

05-02-2022-24064301-Hauptpost-0563-R

Chapter 1

Snort Overview

This manual is based on *Writing Snort Rules* by Martin Roesch and further work from Chris Green <cmg@snort.org>. It was then maintained by Brian Caswell <bmc@snort.org> and now is maintained by the Snort Team. If you have a better way to say something or find that something in the documentation is outdated, drop us a line and we will update it. If you would like to submit patches for this document, you can find the latest version of the documentation in LaTeX format in the Snort CVS repository at `/doc/snort_manual.tex`. Small documentation updates are the easiest way to help out the Snort Project.

1644470956460125853 19

1.1 Getting Started

Snort really isn't very hard to use, but there are a lot of command line options to play with, and it's not always obvious which ones go together well. This file aims to make using Snort easier for new users.

Before we proceed, there are a few basic concepts you should understand about Snort. Snort can be configured to run in three modes:

- *Sniffer mode,* which simply reads the packets off of the network and displays them for you in a continuous stream on the console (screen).
- *Packet Logger mode,* which logs the packets to disk.
- *Network Intrusion Detection System (NIDS) mode,* the most complex and configurable configuration, which allows Snort to analyze network traffic for matches against a user-defined rule set and performs several actions based upon what it sees.

1.2 Sniffer Mode

First, let's start with the basics. If you just want to print out the TCP/IP packet headers to the screen (i.e. sniffer mode), try this:

```
./snort -v
```

This command will run Snort and just show the IP and TCP/UDP/ICMP headers, nothing else. If you want to see the application data in transit, try the following:

```
./snort -vd
```

This instructs Snort to display the packet data as well as the headers. If you want an even more descriptive display, showing the data link layer headers, do this:

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 10

```
./snort -vde
```

(As an aside, these switches may be divided up or smashed together in any combination. The last command could also be typed out as:

```
./snort -d -v -e
```

and it would do the same thing.)

1.3 Packet Logger Mode

OK, all of these commands are pretty cool, but if you want to record the packets to the disk, you need to specify a logging directory and Snort will automatically know to go into packet logger mode:

```
./snort -dev -l ./log
```

Of course, this assumes you have a directory named `log` in the current directory. If you don't, Snort will exit with an error message. When Snort runs in this mode, it collects every packet it sees and places it in a directory hierarchy based upon the IP address of one of the hosts in the datagram.

If you just specify a plain -l switch, you may notice that Snort sometimes uses the address of the remote computer as the directory in which it places packets and sometimes it uses the local host address. In order to log relative to the home network, you need to tell Snort which network is the home network:

```
./snort -dev -l ./log -h 192.168.1.0/24
```

This rule tells Snort that you want to print out the data link and TCP/IP headers as well as application data into the directory `./log`, and you want to log the packets relative to the 192.168.1.0 class C network. All incoming packets will be recorded into subdirectories of the log directory, with the directory names being based on the address of the remote (non-192.168.1) host.





Note that if both the source and destination hosts are on the home network, they are logged to a directory with a name based on the higher of the two port numbers or, in the case of a tie, the source address.

If you're on a high speed network or you want to log the packets into a more compact form for later analysis, you should consider logging in binary mode. Binary mode logs the packets in tcpdump format to a single binary file in the logging directory:

```
./snort -l ./log -b
```

Note the command line changes here. We don't need to specify a home network any longer because binary mode logs everything into a single file, which eliminates the need to tell it how to format the output directory structure. Additionally, you don't need to run in verbose mode or specify the -d or -e switches because in binary mode the entire packet is logged, not just sections of it. All you really need to do to place Snort into logger mode is to specify a logging directory at the command line using the -l switch—the -b binary logging switch merely provides a modifier that tells Snort to log the packets in something other than the default output format of plain ASCII text.

Once the packets have been logged to the binary file, you can read the packets back out of the file with any sniffer that supports the tcpdump binary format (such as tcpdump or Ethereal). Snort can also read the packets back by using the -r switch, which puts it into playback mode. Packets from any tcpdump formatted file can be processed through Snort in any of its run modes. For example, if you wanted to run a binary log file through Snort in sniffer mode to dump the packets to the screen, you can try something like this:

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 11

```
./snort -dv -r packet.log
```

You can manipulate the data in the file in a number of ways through Snort's packet logging and intrusion detection modes, as well as with the BPF interface that's available from the command line. For example, if you only wanted to see the ICMP packets from the log file, simply specify a BPF filter at the command line and Snort will only see the ICMP packets in the file:

```
./snort -dvr packet.log icmp
```

For more info on how to use the BPF interface, read the Snort and tcpdump man pages.

1.4 Network Intrusion Detection System Mode

To enable Network Intrusion Detection System (NIDS) mode so that you don't record every single packet sent down the wire, try this:

```
./snort -dev -l ./log -h 192.168.1.0/24 -c snort.conf
```



where `snort.conf` is the name of your snort configuration file. This will apply the rules configured in the `snort.conf` file to each packet to decide if an action based upon the rule type in the file should be taken. If you don't specify an output directory for the program, it will default to `/var/log/snort`.

One thing to note about the last command line is that if Snort is going to be used in a long term way as an IDS, the -v switch should be left off the command line for the sake of speed. The screen is a slow place to write data to, and packets can be dropped while writing to the display.

It's also not necessary to record the data link headers for most applications, so you can usually omit the -e switch, too.

```
./snort -d -h 192.168.1.0/24 -l ./log -c snort.conf
```

This will configure Snort to run in its most basic NIDS form, logging packets that trigger rules specified in the `snort.conf` in plain ASCII to disk using a hierarchical directory structure (just like packet logger mode).

1.4.1 NIDS Mode Output Options

There are a number of ways to configure the output of Snort in NIDS mode. The default logging and alerting mechanisms are to log in decoded ASCII format and use full alerts. The full alert mechanism prints out the alert message in addition to the full packet headers. There are several other alert output modes available at the command line, as well as two logging facilities.

Alert modes are somewhat more complex. There are seven alert modes available at the command line: full, fast, socket, syslog, console, cmg, and none. Six of these modes are accessed with the -A command line switch. These options are:

Option	Description
`-A fast`	Fast alert mode. Writes the alert in a simple format with a timestamp, alert message, source and destination IPs/ports.
`-A full`	Full alert mode. This is the default alert mode and will be used automatically if you do not specify a mode.
`-A unsock`	Sends alerts to a UNIX socket that another program can listen on.
`-A none`	Turns off alerting.
`-A console`	Sends "fast-style" alerts to the console (screen).
`-A cmg`	Generates "cmg style" alerts.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 12

Packets can be logged to their default decoded ASCII format or to a binary log file via the -b command line switch. To disable packet logging altogether, use the -N command line switch.

For output modes available through the configuration file, see Section 2.6.



Command line logging options override any output options specified in the configuration file. This allows debugging of configuration issues quickly via the command line.

To send alerts to syslog, use the -s switch. The default facilities for the syslog alerting mechanism are LOG_AUTHPRIV and LOG_ALERT. If you want to configure other facilities for syslog output, use the output plugin directives in snort.conf. See Section 2.6.1 for more details on configuring syslog output.

For example, use the following command line to log to default (decoded ASCII) facility and send alerts to syslog:

```
./snort -c snort.conf -l ./log -h 192.168.1.0/24 -s
```

As another example, use the following command line to log to the default facility in /var/log/snort and send alerts to a fast alert file:

```
./snort -c snort.conf -A fast -h 192.168.1.0/24
```



16444709564601258531 9

1.4.2 Understanding Standard Alert Output

When Snort generates an alert message, it will usually look like the following:

```
[**] [116:56:1] (snort_decoder): T/TCP Detected [**]
```

The first number is the Generator ID, this tells the user what component of Snort generated this alert. For a list of GIDs, please read etc/generators in the Snort source. In this case, we know that this event came from the "decode" (116) component of Snort.

The second number is the Snort ID (sometimes referred to as Signature ID). For a list of preprocessor SIDs, please see etc/gen-msg.map. Rule-based SIDs are written directly into the rules with the *sid* option. In this case, *56* represents a T/TCP event.

The third number is the revision ID. This number is primarily used when writing signatures, as each rendition of the rule should increment this number with the *rev* option.

1.4.3 High Performance Configuration

If you want Snort to go *fast* (like keep up with a 1000 Mbps connection), you need to use unified logging and a unified log reader such as *barnyard*. This allows Snort to log alerts in a binary form as fast as possible while another program performs the slow actions, such as writing to a database.

If you want a text file that's easily parsed, but still somewhat fast, try using binary logging with the "fast" output mechanism.

This will log packets in tcpdump format and produce minimal alerts. For example:

```
./snort -b -A fast -c snort.conf
```


Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 13

1.4.4 Changing Alert Order

The default way in which Snort applies its rules to packets may not be appropriate for all installations. The Pass rules are applied first, then the Drop rules, then the Alert rules and finally, Log rules are applied.

⚠NOTE

Sometimes an errant pass rule could cause alerts to not show up, in which case you can change the default ordering to allow Alert rules to be applied before Pass rules. For more information, please refer to the `--alert-before-pass` option.

Several command line options are available to change the order in which rule actions are taken.

- `--alert-before-pass` option forces alert rules to take affect in favor of a pass rule.
- `--treat-drop-as-alert` causes drop and reject rules and any associated alerts to be logged as alerts, rather then the normal action. This allows use of an inline policy with passive/IDS mode. The sdrop rules are not loaded.
- `--process-all-events` option causes Snort to process every event associated with a packet, while taking the actions based on the rules ordering. Without this option (default case), only the events for the first action based on rules ordering are processed.



⚠NOTE

Pass rules are special cases here, in that the event processing is terminated when a pass rule is encountered, regardless of the use of `--process-all-events`.

1.5 Packet Acquisition

Snort 2.9 introduces the DAQ, or Data Acquisition library, for packet I/O. The DAQ replaces direct calls to PCAP functions with an abstraction layer that facilitates operation on a variety of hardware and software interfaces without requiring changes to Snort. It is possible to select the DAQ type and mode when invoking Snort to perform PCAP readback or inline operation, etc.

1.5.1 Configuration

Assuming that you did not disable static modules or change the default DAQ type, you can run Snort just as you always did for file readback or sniffing an interface. However, you can select and configure the DAQ when Snort is invoked as follows:

```
./snort \
    [--daq <type>] \
    [--daq-mode <mode>] \
    [--daq-dir <dir>] \
    [--daq-var <var>]

config daq: <type>
config daq_dir: <dir>
config daq_var: <var>
config daq_mode: <mode>

<type> ::= pcap | afpacket | dump | nfq | ipq | ipfw
```


Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 14

```
<mode> ::= read-file | passive | inline
<var> ::= arbitrary <name>=<value> passed to DAQ
<dir> ::= path where to look for DAQ module so's
```

The DAQ type, mode, variable, and directory may be specified either via the command line or in the conf file. You may include as many variables and directories as needed by repeating the arg / config. DAQ type may be specified at most once in the conf and once on the command line; if configured in both places, the command line overrides the conf.

If the mode is not set explicitly, -Q will force it to inline, and if that hasn't been set, -r will force it to read-file, and if that hasn't been set, the mode defaults to passive. Also, -Q and –daq-mode inline are allowed, since there is no conflict, but -Q and any other DAQ mode will cause a fatal error at start-up.

Note that if Snort finds multiple versions of a given library, the most recent version is selected. This applies to static and dynamic versions of the same library.

```
./snort [--daq-list <dir>]
```

The above command searches the specified directory for DAQ modules and prints type, version, and attributes of each. This feature is not available in the conf.



1.5.2 PCAP

pcap is the default DAQ. if snort is run w/o any DAQ arguments, it will operate as it always did using this module. These are equivalent:

```
./snort -i <device>
./snort -r <file>

./snort --daq pcap --daq-mode passive -i <device>
./snort --daq pcap --daq-mode read-file -r <file>
```

You can specify the buffer size pcap uses with:

```
./snort --daq pcap --daq-var buffer_size=<#bytes>
```

Note that the pcap DAQ does not count filtered packets.

MMAPed pcap

On Linux, a modified version of libpcap is available that implements a shared memory ring buffer. Phil Woods (cpw@lanl.gov) is the current maintainer of the libpcap implementation of the shared memory ring buffer. The shared memory ring buffer libpcap can be downloaded from his website at `http://public.lanl.gov/cpw/`.

Instead of the normal mechanism of copying the packets from kernel memory into userland memory, by using a shared memory ring buffer, libpcap is able to queue packets into a shared buffer that Snort is able to read directly. This change speeds up Snort by limiting the number of times the packet is copied before Snort gets to perform its detection upon it.

Once Snort linked against the shared memory libpcap, enabling the ring buffer is done via setting the environment variable *PCAP_FRAMES. PCAP_FRAMES* is the size of the ring buffer. According to Phil, the maximum size is 32768, as this appears to be the maximum number of iovecs the kernel can handle. By using *PCAP_FRAMES=max*, libpcap will automatically use the most frames possible. On Ethernet, this ends up being 1530 bytes per frame, for a total of around 52 Mbytes of memory for the ring buffer alone.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 15

1.5.3 AFPACKET

afpacket functions similar to the memory mapped pcap DAQ but no external library is required:

```
./snort --daq afpacket -i <device>
        [--daq-var buffer_size_mb=<#MB>]
        [--daq-var debug]
```

If you want to run afpacket in inline mode, you must set device to one or more interface pairs, where each member of a pair is separated by a single colon and each pair is separated by a double colon like this:

```
eth0:eth1
```

or this:

```
eth0:eth1::eth2:eth3
```

By default, the afpacket DAQ allocates 128MB for packet memory. You can change this with:

```
--daq-var buffer_size_mb=<#MB>
```

Note that the total allocated is actually higher, here's why. Assuming the default packet memory with a snaplen of 1518, the numbers break down like this:

1. The frame size is 1518 (snaplen) + the size of the AFPacket header (66 bytes) = 1584 bytes.
2. The number of frames is 128 MB / 1518 = 84733.
3. The smallest block size that can fit at least one frame is 4 KB = 4096 bytes @ 2 frames per block.
4. As a result, we need 84733 / 2 = 42366 blocks.
5. Actual memory allocated is 42366 * 4 KB = 165.5 MB.

16444709564601258531 9

1.5.4 NFQ

NFQ is the new and improved way to process iptables packets:

```
./snort --daq nfq \
    [--daq-var device=<dev>] \
    [--daq-var proto=<proto>] \
    [--daq-var queue=<qid>] \
[--daq-var queue_len=<qlen>]

<dev> ::= ip | eth0, etc; default is IP injection
<proto> ::= ip4 | ip6 | ip*; default is ip4
<qid> ::= 0..65535; default is 0
<qlen> ::= 0..65535; default is 0
```

Notes on iptables are given below.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 16

03-02-2022-24064301-HauptPost-0567

1.5.5 IPQ

IPQ is the old way to process iptables packets. It replaces the inline version available in pre-2.9 versions built with this:

```
./configure --enable-inline / -DGIDS
```

Start the IPQ DAQ as follows:

```
./snort --daq ipq \
    [--daq-var device=<dev>] \
    [--daq-var proto=<proto>] \

<dev> ::= ip | eth0, etc; default is IP injection
<proto> ::= ip4 | ip6; default is ip4
```

Notes on iptables are given below.

16444709564601258531 9

1.5.6 IPFW

IPFW is available for BSD systems. It replaces the inline version available in pre-2.9 versions built with this:

```
./configure --enable-ipfw / -DGIDS -DIPFW
```

This command line argument is no longer supported:

```
./snort -J <port#>
```

Instead, start Snort like this:

```
./snort --daq ipfw [--daq-var port=<port>]

<port> ::= 1..65535; default is 8000
```

* IPFW only supports ip4 traffic.

1.5.7 Dump

The dump DAQ allows you to test the various inline mode features available in 2.9 Snort like injection and normalization.

```
./snort -i <device> --daq dump
./snort -r <pcap> --daq dump
```

By default a file named inline-out.pcap will be created containing all packets that passed through or were generated by snort. You can optionally specify a different name.

```
./snort --daq dump --daq-var file=<name>
```

dump uses the pcap daq for packet acquisition. It therefore does not count filtered packets.

Note that the dump DAQ inline mode is not an actual inline mode. Furthermore, you will probably want to have the pcap DAQ acquire in another mode like this:

```
./snort -r <pcap> -Q --daq dump --daq-var load-mode=read-file
./snort -i <device> -Q --daq dump --daq-var load-mode=passive
```


Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 17

1.5.8 Statistics Changes

The Packet Wire Totals and Action Stats sections of Snort's output include additional fields:

- `Filtered` count of packets filtered out and not handed to Snort for analysis.
- `Injected` packets Snort generated and sent, eg TCP resets.
- `Allow` packets Snort analyzed and did not take action on.
- `Block` packets Snort did not forward, eg due to a block rule.
- `Replace` packets Snort modified.
- `Whitelist` packets that caused Snort to allow a flow to pass w/o inspection by any analysis program.
- `Blacklist` packets that caused Snort to block a flow from passing.
- `Ignore` packets that caused Snort to allow a flow to pass w/o inspection by this instance of Snort.

The action stats show "blocked" packets instead of "dropped" packets to avoid confusion between dropped packets (those Snort didn't actually see) and blocked packets (those Snort did not allow to pass).

164447095646012585319

1.6 Reading Pcaps

Instead of having Snort listen on an interface, you can give it a packet capture to read. Snort will read and analyze the packets as if they came off the wire. This can be useful for testing and debugging Snort.

1.6.1 Command line arguments

Any of the below can be specified multiple times on the command line (`-r` included) and in addition to other Snort command line options. Note, however, that specifying `--pcap-reset` and `--pcap-show` multiple times has the same effect as specifying them once.

Option	Description
`-r <file>`	Read a single pcap.
`--pcap-single=<file>`	Same as -r. Added for completeness.
`--pcap-file=<file>`	File that contains a list of pcaps to read. Can specify path to pcap or directory to recurse to get pcaps.
`--pcap-list="<list>"`	A space separated list of pcaps to read.
`--pcap-dir=<dir>`	A directory to recurse to look for pcaps. Sorted in ASCII order.
`--pcap-filter=<filter>`	Shell style filter to apply when getting pcaps from file or directory. This filter will apply to any `--pcap-file` or `--pcap-dir` arguments following. Use `--pcap-no-filter` to delete filter for following `--pcap-file` or `--pcap-dir` arguments or specify `--pcap-filter` again to forget previous filter and to apply to following `--pcap-file` or `--pcap-dir` arguments.
`--pcap-no-filter`	Reset to use no filter when getting pcaps from file or directory.
`--pcap-reset`	If reading multiple pcaps, reset snort to post-configuration state before reading next pcap. The default, i.e. without this option, is not to reset state.
`--pcap-show`	Print a line saying what pcap is currently being read.

1.6.2 Examples

Read a single pcap

```
$ snort -r foo.pcap
$ snort --pcap-single=foo.pcap
```


Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 18

03-02-2022-24064301-HauptPost-0568

Read pcaps from a file

```
$ cat foo.txt
fool.pcap
foo2.pcap
/home/foo/pcaps

$ snort --pcap-file=foo.txt
```

This will read fool.pcap, foo2.pcap and all files under /home/foo/pcaps. Note that Snort will not try to determine whether the files under that directory are really pcap files or not.

Read pcaps from a command line list

```
$ snort --pcap-list="fool.pcap foo2.pcap foo3.pcap"
```

This will read fool.pcap, foo2.pcap and foo3.pcap.

Read pcaps under a directory

```
$ snort --pcap-dir="/home/foo/pcaps"
```

This will include all of the files under /home/foo/pcaps.

Using filters

```
$ cat foo.txt
fool.pcap
foo2.pcap
/home/foo/pcaps

$ snort --pcap-filter="*.pcap" --pcap-file=foo.txt
$ snort --pcap-filter="*.pcap" --pcap-dir=/home/foo/pcaps
```

The above will only include files that match the shell pattern "*.pcap", in other words, any file ending in ".pcap".

```
$ snort --pcap-filter="*.pcap --pcap-file=foo.txt \
> --pcap-filter="*.cap" --pcap-dir=/home/foo/pcaps
```

In the above, the first filter "*.pcap" will only be applied to the pcaps in the file "foo.txt" (and any directories that are recursed in that file). The addition of the second filter "*.cap" will cause the first filter to be forgotten and then applied to the directory /home/foo/pcaps, so only files ending in ".cap" will be included from that directory.

```
$ snort --pcap-filter="*.pcap --pcap-file=foo.txt \
> --pcap-no-filter --pcap-dir=/home/foo/pcaps
```

In this example, the first filter will be applied to foo.txt, then no filter will be applied to the files found under /home/foo/pcaps, so all files found under /home/foo/pcaps will be included.

```
$ snort --pcap-filter="*.pcap --pcap-file=foo.txt \
> --pcap-no-filter --pcap-dir=/home/foo/pcaps \
> --pcap-filter="*.cap" --pcap-dir=/home/foo/pcaps2
```

In this example, the first filter will be applied to foo.txt, then no filter will be applied to the files found under /home/foo/pcaps, so all files found under /home/foo/pcaps will be included, then the filter "*.cap" will be applied to files found under /home/foo/pcaps2.

16444709564601258531 9

Resetting state

```
$ snort --pcap-dir=/home/foo/pcaps --pcap-reset
```

The above example will read all of the files under /home/foo/pcaps, but after each pcap is read, Snort will be reset to a post-configuration state, meaning all buffers will be flushed, statistics reset, etc. For each pcap, it will be like Snort is seeing traffic for the first time.

Printing the pcap

```
$ snort --pcap-dir=/home/foo/pcaps --pcap-show
```

The above example will read all of the files under /home/foo/pcaps and will print a line indicating which pcap is currently being read.

1.7 Basic Output



Snort does a lot of work and outputs some useful statistics when it is done. Many of these are self-explanatory. The others are summarized below. This does not include all possible output data, just the basics.

1.7.1 Timing Statistics

This section provides basic timing statistics. It includes total seconds and packets as well as packet processing rates. The rates are based on whole seconds, minutes, etc. and only shown when non-zero.

Example:

```
===============================================================================
Run time for packet processing was 175.856509 seconds
Snort processed 3716022 packets.
Snort ran for 0 days 0 hours 2 minutes 55 seconds
   Pkts/min:       1858011
   Pkts/sec:         21234
===============================================================================
```

1.7.2 Packet I/O Totals

This section shows basic packet acquisition and injection peg counts obtained from the DAQ. If you are reading pcaps, the totals are for all pcaps combined, unless you use –pcap-reset, in which case it is shown per pcap.

- Outstanding indicates how many packets are buffered awaiting processing. The way this is counted varies per DAQ so the DAQ documentation should be consulted for more info.
- Filtered packets are not shown for pcap DAQs.
- Injected packets are the result of active response which can be configured for inline or passive modes.

Example:

```
===============================================================================
Packet I/O Totals:
   Received:       3716022
   Analyzed:       3716022 (100.000%)
```


Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 20

03-02-2022-24064301-HauPtPost-0569

```
    Dropped:            0 (  0.000%)
   Filtered:            0 (  0.000%)
Outstanding:            0 (  0.000%)
   Injected:            0
===============================================================================
```

1.7.3 Protocol Statistics

Traffic for all the protocols decoded by Snort is summarized in the breakdown section. This traffic includes internal "pseudo-packets" if preprocessors such as frag3 and stream5 are enabled so the total may be greater than the number of analyzed packets in the packet I/O section.

- Disc counts are discards due to basic encoding integrity flaws that prevents Snort from decoding the packet.
- Other includes packets that contained an encapsulation that Snort doesn't decode.
- S5 G 1/2 is the number of client/server sessions stream5 flushed due to cache limit, session timeout, session reset.

Example:




```
===============================================================================
Breakdown by protocol (includes rebuilt packets):
        Eth:      3722347 (100.000%)
       VLAN:            0 (  0.000%)
        IP4:      1782394 ( 47.884%)
       Frag:         3839 (  0.103%)
       ICMP:        38860 (  1.044%)
        UDP:       137162 (  3.685%)
        TCP:      1619621 ( 43.511%)
        IP6:      1781159 ( 47.850%)
    IP6 Ext:      1787327 ( 48.016%)
   IP6 Opts:         6168 (  0.166%)
      Frag6:         3839 (  0.103%)
      ICMP6:         1650 (  0.044%)
       UDP6:       140446 (  3.773%)
       TCP6:      1619633 ( 43.511%)
     Teredo:           18 (  0.000%)
    ICMP-IP:            0 (  0.000%)
      EAPOL:            0 (  0.000%)
    IP4/IP4:            0 (  0.000%)
    IP4/IP6:            0 (  0.000%)
    IP6/IP4:            0 (  0.000%)
    IP6/IP6:            0 (  0.000%)
        GRE:          202 (  0.005%)
    GRE Eth:            0 (  0.000%)
   GRE VLAN:            0 (  0.000%)
    GRE IP4:            0 (  0.000%)
    GRE IP6:            0 (  0.000%)
GRE IP6 Ext:            0 (  0.000%)
   GRE PPTP:          202 (  0.005%)
    GRE ARP:            0 (  0.000%)
    GRE IPX:            0 (  0.000%)
   GRE Loop:            0 (  0.000%)
       MPLS:            0 (  0.000%)
        ARP:       104840 (  2.817%)
```


Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 21

```
        IPX:           60 (  0.002%)
   Eth Loop:            0 (  0.000%)
   Eth Disc:            0 (  0.000%)
   IP4 Disc:            0 (  0.000%)
   IP6 Disc:            0 (  0.000%)
   TCP Disc:            0 (  0.000%)
   UDP Disc:         1385 (  0.037%)
  ICMP Disc:            0 (  0.000%)
All Discard:         1385 (  0.037%)
      Other:        57876 (  1.555%)
Bad Chk Sum:        32135 (  0.863%)
    Bad TTL:            0 (  0.000%)
     S5 G 1:         1494 (  0.040%)
     S5 G 2:         1654 (  0.044%)
      Total:      3722347
==============================================================================
```

1.7.4 Actions, Limits, and Verdicts

Action and verdict counts show what Snort did with the packets it analyzed. This information is only output in IDS mode (when snort is run with the `-c <conf>` option).

- Alerts is the number of activate, alert, and block actions processed as determined by the rule actions. Here block includes block, drop, and reject actions.

Limits arise due to real world constraints on processing time and available memory. These indicate potential actions that did not happen:

- Match Limit counts rule matches were not processed due to the `config detection: max_queue_events` setting. The default is 5.
- Queue Limit counts events couldn't be stored in the event queue due to the `config event_queue: max_queue` setting. The default is 8.
- Log Limit counts events were not alerted due to the `config event_queue: log` setting. The default is 3.
- Event Limit counts events not alerted due to `event_filter` limits.

Verdicts are rendered by Snort on each packet:

- Allow = packets Snort analyzed and did not take action on.
- Block = packets Snort did not forward, eg due to a block rule. "Block" is used instead of "Drop" to avoid confusion between dropped packets (those Snort didn't actually see) and blocked packets (those Snort did not allow to pass).
- Replace = packets Snort modified, for example, due to normalization or replace rules. This can only happen in inline mode with a compatible DAQ.
- Whitelist = packets that caused Snort to allow a flow to pass w/o inspection by any analysis program. Like blacklist, this is done by the DAQ or by Snort on subsequent packets.
- Blacklist = packets that caused Snort to block a flow from passing. This is the case when a block TCP rule fires. If the DAQ supports this in hardware, no further packets will be seen by Snort for that session. If not, snort will block each packet and this count will be higher.
- Ignore = packets that caused Snort to allow a flow to pass w/o inspection by this instance of Snort. Like blacklist, this is done by the DAQ or by Snort on subsequent packets.

644709564601258531 9

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 22

03-02-2022-24064301-HauPtPost-0570

Example:

```
===============================================================================
Action Stats:
     Alerts:            0 (  0.000%)
     Logged:            0 (  0.000%)
     Passed:            0 (  0.000%)
Match Limit:            0
Queue Limit:            0
  Log Limit:            0
Event Limit:            0
Verdicts:
      Allow:      3716022 (100.000%)
      Block:            0 (  0.000%)
    Replace:            0 (  0.000%)
  Whitelist:            0 (  0.000%)
  Blacklist:            0 (  0.000%)
     Ignore:            0 (  0.000%)
===============================================================================
```

1644709564601258531 9

1.8 Tunneling Protocol Support

Snort supports decoding of GRE, IP in IP and PPTP. To enable support, an extra configuration option is necessary:

```
$ ./configure --enable-gre
```

To enable IPv6 support, one still needs to use the configuration option:

```
$ ./configure --enable-ipv6
```

1.8.1 Multiple Encapsulations

Snort will not decode more than one encapsulation. Scenarios such as

```
Eth IPv4 GRE IPv4 GRE IPv4 TCP Payload
```

or

```
Eth IPv4 IPv6 IPv4 TCP Payload
```

will not be handled and will generate a decoder alert.

1.8.2 Logging

Currently, only the encapsulated part of the packet is logged, e.g.

```
Eth IP1 GRE IP2 TCP Payload
```

gets logged as

```
Eth IP2 TCP Payload
```


Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 23

and

```
Eth IP1 IP2 TCP Payload
```

gets logged as

```
Eth IP2 TCP Payload
```

NOTE

Decoding of PPTP, which utilizes GRE and PPP, is not currently supported on architectures that require word alignment such as SPARC.

1.9 Miscellaneous

1.9.1 Running Snort as a Daemon

If you want to run Snort as a daemon, you can the add -D switch to any combination described in the previous sections. Please notice that if you want to be able to restart Snort by sending a SIGHUP signal to the daemon, you *must* specify the full path to the Snort binary when you start it, for example:

```
/usr/local/bin/snort -d -h 192.168.1.0/24 \
    -l /var/log/snortlogs -c /usr/local/etc/snort.conf -s -D
```

Relative paths are not supported due to security concerns.

Snort PID File

When Snort is run as a daemon , the daemon creates a PID file in the log directory. In Snort 2.6, the `--pid-path` command line switch causes Snort to write the PID file in the directory specified.

Additionally, the `--create-pidfile` switch can be used to force creation of a PID file even when not running in daemon mode.

The PID file will be locked so that other snort processes cannot start. Use the `--nolock-pidfile` switch to not lock the PID file.

1.9.2 Running in Rule Stub Creation Mode

If you need to dump the shared object rules stub to a directory, you might need to use the –dump-dynamic-rules option. These rule stub files are used in conjunction with the shared object rules. The path can be relative or absolute.

```
/usr/local/bin/snort -c /usr/local/etc/snort.conf \
    --dump-dynamic-rules=/tmp
```

This path can also be configured in the snort.conf using the config option dump-dynamic-rules-path as follows:

```
config dump-dynamic-rules-path: /tmp/sorules
```

The path configured by command line has precedence over the one configured using dump-dynamic-rules-path.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 24

16444709564601258531 9

```
/usr/local/bin/snort -c /usr/local/etc/snort.conf \
    --dump-dynamic-rules

snort.conf:
config dump-dynamic-rules-path: /tmp/sorules
```

In the above mentioned scenario the dump path is set to /tmp/sorules.

1.9.3 Obfuscating IP Address Printouts

If you need to post packet logs to public mailing lists, you might want to use the -O switch. This switch obfuscates your IP addresses in packet printouts. This is handy if you don't want people on the mailing list to know the IP addresses involved. You can also combine the -O switch with the -h switch to only obfuscate the IP addresses of hosts on the home network. This is useful if you don't care who sees the address of the attacking host. For example, you could use the following command to read the packets from a log file and dump them to the screen, obfuscating only the addresses from the 192.168.1.0/24 class C network:

```
./snort -d -v -r snort.log -O -h 192.168.1.0/24
```



1.9.4 Specifying Multiple-Instance Identifiers

In Snort v2.4, the `-G` command line option was added that specifies an instance identifier for the event logs. This option can be used when running multiple instances of snort, either on different CPUs, or on the same CPU but a different interface. Each Snort instance will use the value specified to generate unique event IDs. Users can specify either a decimal value (`-G 1`) or hex value preceded by 0x (`-G 0x11`). This is also supported via a long option `--logid`.

1.9.5 Snort Modes

Snort can operate in three different modes namely tap (passive), inline, and inline-test. Snort policies can be configured in these three modes too.

Explanation of Modes

- `Inline`

  When Snort is in Inline mode, it acts as an IPS allowing drop rules to trigger. Snort can be configured to run in inline mode using the command line argument -Q and snort config option `policy_mode` as follows:

  ```
  snort -Q
  config policy_mode:inline
  ```

- `Passive`

  When Snort is in Passive mode, it acts as a IDS. Drop rules are not loaded (without –treat-drop-as-alert). Snort can be configured to passive mode using the snort config option `policy_mode` as follows:

  ```
  config policy_mode:tap
  ```

- `Inline-Test`

  Inline-Test mode simulates the inline mode of snort, allowing evaluation of inline behavior without affecting traffic. The drop rules will be loaded and will be triggered as a Wdrop (Would Drop) alert. Snort can be configured to run in inline-test mode using the command line option (–enable-inline-test) or using the snort config option `policy_mode` as follows:

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 25

```
snort --enable-inline-test
config policy_mode:inline_test
```

NOTE

Please note –enable-inline-test cannot be used in conjunction with -Q.

Behavior of different modes with rule options

Rule Option	Inline Mode	Passive Mode	Inline-Test Mode
reject	Drop + Response	Alert + Response	Wdrop + Response
react	Blocks and send notice	Blocks and send notice	Blocks and send notice
normalize	Normalizes packet	Doesn't normalize	Doesn't normalize
replace	replace content	Doesn't replace	Doesn't replace
respond	close session	close session	close session

Behavior of different modes with rules actions

Adapter Mode	Snort args	config policy_mode	Drop Rule Handling
Passive	snort --treat-drop-as-alert	tap	Alert
Passive	snort	tap	Not Loaded
Passive	snort --treat-drop-as-alert	inline_test	Alert
Passive	snort	inline_test	Would Drop
Passive	snort --treat-drop-as-alert	inline	Alert
Passive	snort	inline	Not loaded + warning
Inline Test	snort --enable-inline-test --treat-drop-as-alert	tap	Alert
Inline Test	snort --enable-inline-test	tap	Would Drop
Inline Test	snort --enable-inline-test --treat-drop-as-alert	inline_test	Alert
Inline Test	snort --enable-inline-test	inline_test	Would Drop
Inline Test	snort --enable-inline-test --treat-drop-as-alert	inline	Alert
Inline Test	snort --enable-inline-test	inline	Would Drop
Inline	snort -Q --treat-drop-as-alert	tap	Alert
Inline	snort -Q	tap	Alert
Inline	snort -Q --treat-drop-as-alert	inline_test	Alert
Inline	snort -Q	inline_test	Would Drop
Inline	snort -Q --treat-drop-as-alert	inline	Alert
Inline	snort -Q	inline	Drop

16444709564601258531 9

1.10 More Information

Chapter 2 contains much information about many configuration options available in the configuration file. The Snort manual page and the output of `snort -?` or `snort --help` contain information that can help you get Snort running in several different modes.

NOTE

In many shells, a backslash (\) is needed to escape the ?, so you may have to type `snort -\?` instead of `snort -?` for a list of Snort command line options.

The Snort web page (`http://www.snort.org`) and the Snort Users mailing list:

`http://marc.theaimsgroup.com/?l=snort-users`

at `snort-users@lists.sourceforge.net` provide informative announcements as well as a venue for community discussion and support. There's a lot to Snort, so sit back with a beverage of your choosing and read the documentation and mailing list archives.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 26

Chapter 2

Configuring Snort

2.1 Includes

The `include` keyword allows other snort config files to be included within the snort.conf indicated on the Snort command line. It works much like an #include from the C programming language, reading the contents of the named file and adding the contents in the place where the include statement appears in the file.

2.1.1 Format

```
include <include file path/name>
```



Note that there is no semicolon at the end of this line.

Included files will substitute any predefined variable values into their own variable references. See Section 2.1.2 for more information on defining and using variables in Snort config files.

2.1.2 Variables

Three types of variables may be defined in Snort:

- var
- portvar
- ipvar



Note: 'ipvar's are only enabled with IPv6 support. Without IPv6 support, use a regular 'var'.

These are simple substitution variables set with the `var`, `ipvar`, or `portvar` keywords as follows:

```
var RULES_PATH rules/
portvar MY_PORTS [22,80,1024:1050]
ipvar MY_NET [192.168.1.0/24,10.1.1.0/24]
alert tcp any any -> $MY_NET $MY_PORTS (flags:S; msg:"SYN packet";)
include $RULE_PATH/example.rule
```


Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 27

1644470956460125853 19

IP Variables and IP Lists

IPs may be specified individually, in a list, as a CIDR block, or any combination of the three. If IPv6 support is enabled, IP variables should be specified using 'ipvar' instead of 'var'. Using 'var' for an IP variable is still allowed for backward compatibility, but it will be deprecated in a future release.

IPs, IP lists, and CIDR blocks may be negated with '!'. Negation is handled differently compared with Snort versions 2.7.x and earlier. Previously, each element in a list was logically OR'ed together. IP lists now OR non-negated elements and AND the result with the OR'ed negated elements.

The following example list will match the IP 1.1.1.1 and IP from 2.2.2.0 to 2.2.2.255, with the exception of IPs 2.2.2.2 and 2.2.2.3.

```
[1.1.1.1,2.2.2.0/24,![2.2.2.2,2.2.2.3]]
```

The order of the elements in the list does not matter. The element 'any' can be used to match all IPs, although '!any' is not allowed. Also, negated IP ranges that are more general than non-negated IP ranges are not allowed.

See below for some valid examples if IP variables and IP lists.




```
ipvar EXAMPLE [1.1.1.1,2.2.2.0/24,![2.2.2.2,2.2.2.3]]

alert tcp $EXAMPLE any -> any any (msg:"Example"; sid:1;)

alert tcp [1.0.0.0/8,!1.1.1.0/24] any -> any any (msg:"Example";sid:2;)
```

The following examples demonstrate some invalid uses of IP variables and IP lists.

Use of !any:

```
ipvar EXAMPLE any
alert tcp !$EXAMPLE any -> any any (msg:"Example";sid:3;)
```

Different use of !any:

```
ipvar EXAMPLE !any
alert tcp $EXAMPLE any -> any any (msg:"Example";sid:3;)
```

Logical contradictions:

```
ipvar EXAMPLE [1.1.1.1,!1.1.1.1]
```

Nonsensical negations:

```
ipvar EXAMPLE [1.1.1.0/24,!1.1.0.0/16]
```

Port Variables and Port Lists

Portlists supports the declaration and lookup of ports and the representation of lists and ranges of ports. Variables, ranges, or lists may all be negated with '!'. Also, 'any' will specify any ports, but '!any' is not allowed. Valid port ranges are from 0 to 65535.

Lists of ports must be enclosed in brackets and port ranges may be specified with a ':', such as in:

```
[10:50,888:900]
```


Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 28

Port variables should be specified using 'portvar'. The use of 'var' to declare a port variable will be deprecated in a future release. For backwards compatibility, a 'var' can still be used to declare a port variable, provided the variable name either ends with '_PORT' or begins with 'PORT_'.

The following examples demonstrate several valid usages of both port variables and port lists.

```
portvar EXAMPLE1 80

var EXAMPLE2_PORT [80:90]

var PORT_EXAMPLE2 [1]

portvar EXAMPLE3 any

portvar EXAMPLE4 [!70:90]

portvar EXAMPLE5 [80,91:95,100:200]

alert tcp any $EXAMPLE1 -> any $EXAMPLE2_PORT (msg:"Example"; sid:1;)

alert tcp any $PORT_EXAMPLE2 -> any any (msg:"Example"; sid:2;)

alert tcp any 90 -> any [100:1000,9999:20000] (msg:"Example"; sid:3;)
```



Several invalid examples of port variables and port lists are demonstrated below:

Use of !any:

```
portvar EXAMPLE5 !any
var EXAMPLE5 !any
```

Logical contradictions:

```
portvar EXAMPLE6 [80,!80]
```

Ports out of range:

```
portvar EXAMPLE7 [65536]
```

Incorrect declaration and use of a port variable:

```
var EXAMPLE8 80
alert tcp any $EXAMPLE8 -> any any (msg:"Example"; sid:4;)
```

Port variable used as an IP:

```
alert tcp $EXAMPLE1 any -> any any (msg:"Example"; sid:5;)
```

Variable Modifiers

Rule variable names can be modified in several ways. You can define meta-variables using the $ operator. These can be used with the variable modifier operators ? and -, as described in the following table:

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 29

Variable Syntax	Description
var	Defines a meta-variable.
$(var) or $var	Replaces with the contents of variable var.
$(var:-default)	Replaces the contents of the variable var with "default" if var is undefined.
$(var:?message)	Replaces with the contents of variable var or prints out the error message and exits.

Here is an example of advanced variable usage in action:

```
ipvar MY_NET 192.168.1.0/24
log tcp any any -> $(MY_NET:?MY_NET is undefined!) 23
```

Limitations

When embedding variables, types can not be mixed. For instance, port variables can be defined in terms of other port variables, but old-style variables (with the 'var' keyword) can not be embedded inside a 'portvar'.

Valid embedded variable:

```
portvar pvar1 80
portvar pvar2 [$pvar1,90]
```

Invalid embedded variable:

```
var pvar1 80
portvar pvar2 [$pvar1,90]
```

Likewise, variables can not be redefined if they were previously defined as a different type. They should be renamed instead:

Invalid redefinition:

```
var pvar 80
portvar pvar 90
```

2.1.3 Config

Many configuration and command line options of Snort can be specified in the configuration file.

Format

```
config <directive> [: <value>]
```


Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 30

16444709564601258531 9

03-02-2022-24064301-Hauptpost-0574

Config Directive	Description
config alert_with_interface_name	Appends interface name to alert (snort -I).
config alertfile: <filename>	Sets the alerts output file.
config asn1: <max-nodes>	Specifies the maximum number of nodes to track when doing ASN1 decoding. See Section 3.5.31 for more information and examples.
config autogenerate_preprocessor _decoder_rules	If Snort was configured to enable decoder and preprocessor rules, this option will cause Snort to revert back to it's original behavior of alerting if the decoder or preprocessor generates an event.
config bpf_file: <filename>	Specifies BPF filters (snort -F).
config checksum_drop: <types>	Types of packets to drop if invalid checksums. Values: none, noip, notcp, noicmp, noudp, ip, tcp, udp, icmp or all (only applicable in inline mode and for packets checked per checksum_mode config option).
config checksum_mode: <types>	Types of packets to calculate checksums. Values: none, noip, notcp, noicmp, noudp, ip, tcp, udp, icmp or all.
config chroot: <dir>	Chroots to specified dir (snort -t).
config classification: <class>	See Table 3.2 for a list of classifications.
config daemon	Forks as a daemon (snort -D).
config decode_data_link	Decodes Layer2 headers (snort -e).
config default_rule_state: <state>	Global configuration directive to enable or disable the loading of rules into the detection engine. Default (with or without directive) is enabled. Specify disabled to disable loading rules.
config daq: <type>	Selects the type of DAQ to instantiate. The DAQ with the highest version of the given type is selected if there are multiple of the same type (this includes any built-in DAQs).
config daq_mode: <mode>	Select the DAQ mode: passive, inline, or read-file. Not all DAQs support modes. See the DAQ distro README for possible DAQ modes or list DAQ capabilities for a brief summary.
config daq_var: <name=value>	Set a DAQ specific variable. Snort just passes this information down to the DAQ. See the DAQ distro README for possible DAQ variables.
config daq_dir: <dir>	Tell Snort where to look for available dynamic DAQ modules. This can be repeated. The selected DAQ will be the one with the latest version.
config daq_list: [<dir>]	Tell Snort to dump basic DAQ capabilities and exit. You can optionally specify a directory to include any dynamic DAQs from that directory. You can also preceed this option with extra DAQ directory options to look in multiple directories.

16444709564601258531 9

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 31

`config detection: [search-method <method>]`	Select type of fast pattern matcher algorithm to use. • `search-method <method>` – Queued match search methods - Matches are queued until the fast pattern matcher is finished with the payload, then evaluated. This was found to generally increase performance through fewer cache misses (evaluating each rule would generally blow away the fast pattern matcher state in the cache). * `ac` and `ac-q` - Aho-Corasick Full (high memory, best performance). * `ac-bnfa` and `ac-bnfa-q` - Aho-Corasick Binary NFA (low memory, high performance) * `lowmem` and `lowmem-q` - Low Memory Keyword Trie (low memory, moderate performance) * `ac-split` - Aho-Corasick Full with ANY-ANY port group evaluated separately (low memory, high performance). Note this is shorthand for `search-method ac, split-any-any` * `intel-cpm` - Intel CPM library (must have compiled Snort with location of libraries to enable this) – No queue search methods - The "nq" option specifies that matches should not be queued and evaluated as they are found. * `ac-nq` - Aho-Corasick Full (high memory, best performance). * `ac-bnfa-nq` - Aho-Corasick Binary NFA (low memory, high performance). This is the default search method if none is specified. * `lowmem-nq` - Low Memory Keyword Trie (low memory, moderate performance) – Other search methods (the above are considered superior to these) * `ac-std` - Aho-Corasick Standard (high memory, high performance) * `acs` - Aho-Corasick Sparse (high memory, moderate performance) * `ac-banded` - Aho-Corasick Banded (high memory, moderate performance) * `ac-sparsebands` - Aho-Corasick Sparse-Banded (high memory, moderate performance)



Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 32

03-02-2022-24064301-Hauptpost-0575

`config detection: [split-any-any] [search-optimize] [max-pattern-len <int>]`	Other options that affect fast pattern matching. • `split-any-any` – A memory/performance tradeoff. By default, ANY-ANY port rules are added to every non ANY-ANY port group so that only one port group rule evaluation needs to be done per packet. Not putting the ANY-ANY port rule group into every other port group can significantly reduce the memory footprint of the fast pattern matchers if there are many ANY-ANY port rules. But doing so may require two port group evaluations per packet - one for the specific port group and one for the ANY-ANY port group, thus potentially reducing performance. This option is generic and can be used with any `search-method` but was specifically intended for use with the `ac search-method` where the memory footprint is significantly reduced though overall fast pattern performance is better than `ac-bnfa`. Of note is that the lower memory footprint can also increase performance through fewer cache misses. Default is not to split the ANY-ANY port group. • `search-optimize` – Optimizes fast pattern memory when used with `search-method ac` or `ac-split` by dynamically determining the size of a state based on the total number of states. When used with `ac-bnfa`, some fail-state resolution will be attempted, potentially increasing performance. Default is not to optimize. • `max-pattern-len <integer>` – This is a memory optimization that specifies the maximum length of a pattern that will be put in the fast pattern matcher. Patterns longer than this length will be truncated to this length before inserting into the pattern matcher. Useful when there are very long contents being used and truncating the pattern won't diminish the uniqueness of the patterns. Note that this may cause more false positive rule evaluations, i.e. rules that will be evaluated because a fast pattern was matched, but eventually fail, however CPU cache can play a part in performance so a smaller memory footprint of the fast pattern matcher can potentially increase performance. Default is to not set a maximum pattern length.



`config detection:` `[no_stream_inserts]` `[max_queue_events <int>]` `[enable-single-rule-group]` `[bleedover-port-limit]`	Other detection engine options. • `no_stream_inserts` – Specifies that stream inserted packets should not be evaluated against the detection engine. This is a potential performance improvement with the idea that the stream rebuilt packet will contain the payload in the inserted one so the stream inserted packet doesn't need to be evaluated. Default is to inspect stream inserts. • `max_queue_events <integer>` – Specifies the maximum number of events to queue per packet. Default is 5 events. • `enable-single-rule-group` – Put all rules into one port group. Not recommended. Default is not to do this. • `bleedover-port-limit` – The maximum number of source or destination ports designated in a rule before the rule is considered an ANY-ANY port group rule. Default is 1024.



Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 34

03-02-2022-24064301-HauptPost-0576



`config detection: [debug] [debug-print-nocontent-rule-tests] [debug-print-rule-group-build-details] [debug-print-rule-groups-uncompiled] [debug-print-rule-groups-compiled] [debug-print-fast-pattern] [bleedover-warnings-enabled]`	Options for detection engine debugging. • `debug` – Prints fast pattern information for a particular port group. • `debug-print-nocontent-rule-tests` – Prints port group information during packet evaluation. • `debug-print-rule-group-build-details` – Prints port group information during port group compilation. • `debug-print-rule-groups-uncompiled` – Prints uncompiled port group information. • `debug-print-rule-groups-compiled` – Prints compiled port group information. • `debug-print-fast-pattern` – For each rule with fast pattern content, prints information about the content being used for the fast pattern matcher. • `bleedover-warnings-enabled` – Prints a warning if the number of source or destination ports used in a rule exceed the `bleedover-port-limit` forcing the rule to be moved into the ANY-ANY port group.
`config disable_decode_alerts`	Turns off the alerts generated by the decode phase of Snort.
`config disable_inline_init_failopen`	Disables failopen thread that allows inline traffic to pass while Snort is starting up. Only useful if Snort was configured with –enable-inline-init-failopen. (`snort --disable-inline-init-failopen`)
`config disable_ipopt_alerts`	Disables IP option length validation alerts.
`config disable_tcpopt_alerts`	Disables option length validation alerts.
`config disable_tcpopt_experimental_alerts`	Turns off alerts generated by experimental TCP options.
`config disable_tcpopt_obsolete_alerts`	Turns off alerts generated by obsolete TCP options.
`config disable_tcpopt_ttcp_alerts`	Turns off alerts generated by T/TCP options.
`config disable_ttcp_alerts`	Turns off alerts generated by T/TCP options.
`config dump_chars_only`	Turns on character dumps (`snort -C`).
`config dump_payload`	Dumps application layer (`snort -d`).
`config dump_payload_verbose`	Dumps raw packet starting at link layer (`snort -X`).
`config enable_decode_drops`	Enables the dropping of bad packets identified by decoder (only applicable in inline mode).
`config enable_decode_oversized_alerts`	Enable alerting on packets that have headers containing length fields for which the value is greater than the length of the packet.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 35

`config enable_decode_oversized_drops`	Enable dropping packets that have headers containing length fields for which the value is greater than the length of the packet. `enable_decode_oversized_alerts` must also be enabled for this to be effective (only applicable in inline mode).
`config enable_deep_teredo_inspection`	Snort's packet decoder only decodes Teredo (IPv6 over UDP over IPv4) traffic on UDP port 3544. This option makes Snort decode Teredo traffic on all UDP ports.
`config enable_ipopt_drops`	Enables the dropping of bad packets with bad/truncated IP options (only applicable in inline mode).
`config enable_mpls_multicast`	Enables support for MPLS multicast. This option is needed when the network allows MPLS multicast traffic. When this option is off and MPLS multicast traffic is detected, Snort will generate an alert. By default, it is off.
`config enable_mpls_overlapping_ip`	Enables support for overlapping IP addresses in an MPLS network. In a normal situation, where there are no overlapping IP addresses, this configuration option should not be turned on. However, there could be situations where two private networks share the same IP space and different MPLS labels are used to differentiate traffic from the two VPNs. In such a situation, this configuration option should be turned on. By default, it is off.
`config enable_tcpopt_drops`	Enables the dropping of bad packets with bad/truncated TCP option (only applicable in inline mode).
`config enable_tcpopt_experimental_drops`	Enables the dropping of bad packets with experimental TCP option. (only applicable in inline mode).
`config enable_tcpopt_obsolete_drops`	Enables the dropping of bad packets with obsolete TCP option. (only applicable in inline mode).
`config enable_tcpopt_ttcp_drops`	Enables the dropping of bad packets with T/TCP option. (only applicable in inline mode).
`config enable_ttcp_drops`	Enables the dropping of bad packets with T/TCP option. (only applicable in inline mode).
`config event_filter: memcap <bytes>`	Set global memcap in bytes for thresholding. Default is 1048576 bytes (1 megabyte).
`config event_queue: [max_queue <num>] [log <num>] [order_events <order>]`	Specifies conditions about Snort's event queue. You can use the following options: • `max_queue <integer>` (max events supported) • `log <integer>` (number of events to log) • `order_events [priority|content_length]` (how to order events within the queue) See Section 2.4.4 for more information and examples.
`config flowbits_size: <num-bits>`	Specifies the maximum number of flowbit tags that can be used within a rule set. The default is 1024 bits and maximum is 2096.
`config ignore_ports: <proto> <port-list>`	Specifies ports to ignore (useful for ignoring noisy NFS traffic). Specify the protocol (TCP, UDP, IP, or ICMP), followed by a list of ports. Port ranges are supported.
`config interface: <iface>`	Sets the network interface (`snort -i`).

164447095646012585319

03-02-2022-24064301-Hauptpost-0577

config ipv6_frag: [bsd_icmp_frag_alert on|off] [, bad_ipv6_frag_alert on|off] [, frag_timeout <secs>] [, max_frag_sessions <max-track>]	The following options can be used: • bsd_icmp_frag_alert on|off (Specify whether or not to alert. Default is on) • bad_ipv6_frag_alert on|off (Specify whether or not to alert. Default is on) • frag_timeout <integer> (Specify amount of time in seconds to timeout first frag in hash table) • max_frag_sessions <integer> (Specify the number of fragments to track in the hash table)
config logdir: <dir>	Sets the logdir (snort -l).
config max_attribute_hosts: <hosts>	Sets a limit on the maximum number of hosts to read from the attribute table. Minimum value is 32 and the maximum is 524288 (512k). The default is 10000. If the number of hosts in the attribute table exceeds this value, an error is logged and the remainder of the hosts are ignored. This option is only supported with a Host Attribute Table (see section 2.7).
config max_mpls_labelchain_len: <num-hdrs>	Sets a Snort-wide limit on the number of MPLS headers a packet can have. Its default value is -1, which means that there is no limit on label chain length.
config min_ttl: <ttl>	Sets a Snort-wide minimum ttl to ignore all traffic.
config mpls_payload_type: ipv4|ipv6|ethernet	Sets a Snort-wide MPLS payload type. In addition to ipv4, ipv6 and ethernet are also valid options. The default MPLS payload type is ipv4
config no_promisc	Disables promiscuous mode (snort -p).
config nolog	Disables logging. Note: Alerts will still occur. (snort -N).
config nopcre	Disables pcre pattern matching.
config obfuscate	Obfuscates IP Addresses (snort -O).
config order: <order>	Changes the order that rules are evaluated, eg: pass alert log activation.
config pcre_match_limit: <integer>	Restricts the amount of backtracking a given PCRE option. For example, it will limit the number of nested repeats within a pattern. A value of -1 allows for unlimited PCRE, up to the PCRE library compiled limit (around 10 million). A value of 0 results in no PCRE evaluation. The snort default value is 1500.
config pcre_match_limit_recursion: <integer>	Restricts the amount of stack used by a given PCRE option. A value of -1 allows for unlimited PCRE, up to the PCRE library compiled limit (around 10 million). A value of 0 results in no PCRE evaluation. The snort default value is 1500. This option is only useful if the value is less than the pcre_match_limit
config pkt_count: <N>	Exits after N packets (snort -n).
config policy_version: <base-version-string> [<binding-version-string>]	Supply versioning information to configuration files. Base version should be a string in all configuration files including included ones. In addition, binding version must be in any file configured with config binding. This option is used to avoid race conditions when modifying and loading a configuration within a short time span - before Snort has had a chance to load a previous configuration.
config profile_preprocs	Print statistics on preprocessor performance. See Section 2.5.2 for more details.
config profile_rules	Print statistics on rule performance. See Section 2.5.1 for more details.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 37

16447095646012585319

config quiet	Disables banner and status reports (snort -q). NOTE: The command line switch -q takes effect immediately after processing the command line parameters, whereas using config quiet in snort.conf takes effect when the configuration line in snort.conf is parsed. That may occur after other configuration settings that result in output to console or syslog.
config read_bin_file: <pcap>	Specifies a pcap file to use (instead of reading from network), same effect as -r <tf> option.
config reference: <ref>	Adds a new reference system to Snort, eg: myref http://myurl.com/?id=
config reference_net <cidr>	For IP obfuscation, the obfuscated net will be used if the packet contains an IP address in the reference net. Also used to determine how to set up the logging directory structure for the session post detection rule option and ASCII output plugin - an attempt is made to name the log directories after the IP address that is not in the reference net.
config response: [attempts <count>] [, device <dev>]	Set the number of strafing attempts per injected response and/or the device, such as eth0, from which to send responses. These options may appear in any order but must be comma separated. The are intended for passive mode.
config set_gid: <gid>	Changes GID to specified GID (snort -g).
config set_uid: <uid>	Sets UID to <id> (snort -u).
config show_year	Shows year in timestamps (snort -y).
config snaplen: <bytes>	Set the snaplength of packet, same effect as -P <snaplen> or --snaplen <snaplen> options.
config so_rule_memcap: <bytes>	Set global memcap in bytes for so rules that dynamically allocate memory for storing session data in the stream preprocessor. A value of 0 disables the memcap. Default is 0. Maximum value is the maximum value an unsigned 32 bit integer can hold which is 4294967295 or 4GB.
config stateful	Sets assurance mode for stream (stream is established).
config tagged_packet_limit: <max-tag>	When a metric other than packets is used in a tag option in a rule, this option sets the maximum number of packets to be tagged regardless of the amount defined by the other metric. See Section 3.7.5 on using the tag option when writing rules for more details. The default value when this option is not configured is 256 packets. Setting this option to a value of 0 will disable the packet limit.
config threshold: memcap <bytes>	Set global memcap in bytes for thresholding. Default is 1048576 bytes (1 megabyte). (This is deprecated. Use config event_filter instead.)
config timestats_interval: <secs>	Set the amount of time in seconds between logging time stats. Default is 3600 (1 hour). Note this option is only available if Snort was built to use time stats with --enable-timestats.
config umask: <umask>	Sets umask when running (snort -m).
config utc	Uses UTC instead of local time for timestamps (snort -U).
config verbose	Uses verbose logging to STDOUT (snort -v).
config vlan_agnostic	Causes Snort to ignore vlan headers for the purposes of connection tracking. This option is only valid in the base configuration when using multiple configurations, and the default is off.
config policy_mode: tap|inline|inline_test	Sets the policy mode to either passive, inline or inline_test.



03-02-2022-24064301-HauptPost-0578

2.2 Preprocessors

Preprocessors were introduced in version 1.5 of Snort. They allow the functionality of Snort to be extended by allowing users and programmers to drop modular plugins into Snort fairly easily. Preprocessor code is run before the detection engine is called, but after the packet has been decoded. The packet can be modified or analyzed in an out-of-band manner using this mechanism.

Preprocessors are loaded and configured using the `preprocessor` keyword. The format of the preprocessor directive in the Snort config file is:

```
preprocessor <name>: <options>
```

2.2.1 Frag3

The frag3 preprocessor is a target-based IP defragmentation module for Snort. Frag3 is intended as a replacement for the frag2 defragmentation module and was designed with the following goals:

1. Faster execution than frag2 with less complex data management.
2. Target-based host modeling anti-evasion techniques.



The frag2 preprocessor used splay trees extensively for managing the data structures associated with defragmenting packets. Splay trees are excellent data structures to use when you have some assurance of locality of reference for the data that you are handling but in high speed, heavily fragmented environments the nature of the splay trees worked against the system and actually hindered performance. Frag3 uses the sfxhash data structure and linked lists for data handling internally which allows it to have much more predictable and deterministic performance in any environment which should aid us in managing heavily fragmented environments.

Target-based analysis is a relatively new concept in network-based intrusion detection. The idea of a target-based system is to model the actual targets on the network instead of merely modeling the protocols and looking for attacks within them. When IP stacks are written for different operating systems, they are usually implemented by people who read the RFCs and then write their interpretation of what the RFC outlines into code. Unfortunately, there are ambiguities in the way that the RFCs define some of the edge conditions that may occur and when this happens different people implement certain aspects of their IP stacks differently. For an IDS this is a big problem.

In an environment where the attacker can determine what style of IP defragmentation is being used on a particular target, the attacker can try to fragment packets such that the target will put them back together in a specific manner while any passive systems trying to model the host traffic have to guess which way the target OS is going to handle the overlaps and retransmits. As I like to say, if the attacker has more information about the targets on a network than the IDS does, it is possible to evade the IDS. This is where the idea for "target-based IDS" came from. For more detail on this issue and how it affects IDS, check out the famous Ptacek & Newsham paper at `http://www.snort.org/docs/idspaper/`.

The basic idea behind target-based IDS is that we tell the IDS information about hosts on the network so that it can avoid Ptacek & Newsham style evasion attacks based on information about how an individual target IP stack operates. Vern Paxson and Umesh Shankar did a great paper on this very topic in 2003 that detailed mapping the hosts on a network and determining how their various IP stack implementations handled the types of problems seen in IP defragmentation and TCP stream reassembly. Check it out at `http://www.icir.org/vern/papers/activemap-oak03.pdf`.

We can also present the IDS with topology information to avoid TTL-based evasions and a variety of other issues, but that's a topic for another day. Once we have this information we can start to really change the game for these complex modeling problems.

Frag3 was implemented to showcase and prototype a target-based module within Snort to test this idea.

Frag 3 Configuration

Frag3 configuration is somewhat more complex than frag2. There are at least two preprocessor directives required to activate frag3, a global configuration directive and an engine instantiation. There can be an arbitrary number of

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 39

engines defined at startup with their own configuration, but only one global configuration.

Global Configuration

- Preprocessor name: `frag3_global`
- Available options: NOTE: Global configuration options are comma separated.
  - `max_frags <number>` - Maximum simultaneous fragments to track. Default is 8192.
  - `memcap <bytes>` - Memory cap for self preservation. Default is 4MB.
  - `prealloc_frags <number>` - Alternate memory management mode. Use preallocated fragment nodes (faster in some situations).
  - `disabled` - Option to turn off the preprocessor. By default this option is turned off. When the preprocessor is disabled only the options memcap, prealloc_memcap, and prealloc_frags are applied when specified with the configuration.

Engine Configuration



- Preprocessor name: `frag3_engine`
- Available options: NOTE: Engine configuration options are space separated.
  - `timeout <seconds>` - Timeout for fragments. Fragments in the engine for longer than this period will be automatically dropped. Default is 60 seconds.
  - `min_ttl <value>` - Minimum acceptable TTL value for a fragment packet. Default is 1. The accepted range for this option is 1 - 255.
  - `detect_anomalies` - Detect fragment anomalies.
  - `bind_to <ip_list>` - IP List to bind this engine to. This engine will only run for packets with destination addresses contained within the IP List. Default value is `all`.
  - `overlap_limit <number>` - Limits the number of overlapping fragments per packet. The default is "0" (unlimited). This config option takes values equal to or greater than zero. This is an optional parameter. detect_anomalies option must be configured for this option to take effect.
  - `min_fragment_length <number>` - Defines smallest fragment size (payload size) that should be considered valid. Fragments smaller than or equal to this limit are considered malicious and an event is raised, if detect_anomalies is also configured. The default is "0" (unlimited), the minimum is "0". This is an optional parameter. detect_anomalies option must be configured for this option to take effect.
  - `policy <type>` - Select a target-based defragmentation mode. Available types are first, last, bsd, bsd-right, linux. Default type is bsd.

    The Paxson Active Mapping paper introduced the terminology frag3 is using to describe policy types. The known mappings are as follows. Anyone who develops more mappings and would like to add to this list please feel free to send us an email!

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 40

03-02-2022-24064301-HauPtPost-0579

Platform	Type
AIX 2	BSD
AIX 4.3 8.9.3	BSD
Cisco IOS	Last
FreeBSD	BSD
HP JetDirect (printer)	BSD-right
HP-UX B.10.20	BSD
HP-UX 11.00	First
IRIX 4.0.5F	BSD
IRIX 6.2	BSD
IRIX 6.3	BSD
IRIX64 6.4	BSD
Linux 2.2.10	linux
Linux 2.2.14-5.0	linux
Linux 2.2.16-3	linux
Linux 2.2.19-6.2.10smp	linux
Linux 2.4.7-10	linux
Linux 2.4.9-31SGI 1.0.2smp	linux
Linux 2.4 (RedHat 7.1-7.3)	linux
MacOS (version unknown)	First
NCD Thin Clients	BSD
OpenBSD (version unknown)	linux
OpenBSD (version unknown)	linux
OpenVMS 7.1	BSD
OS/2 (version unknown)	BSD
OSF1 V3.0	BSD
OSF1 V3.2	BSD
OSF1 V4.0,5.0,5.1	BSD
SunOS 4.1.4	BSD
SunOS 5.5.1,5.6,5.7,5.8	First
Tru64 Unix V5.0A,V5.1	BSD
Vax/VMS	BSD
Windows (95/98/NT4/W2K/XP)	First



Format

Note in the advanced configuration below that there are three engines specified running with *Linux*, `first` and `last` policies assigned. The first two engines are bound to specific IP address ranges and the last one applies to all other traffic. Packets that don't fall within the address requirements of the first two engines automatically fall through to the third one.

Basic Configuration

```
preprocessor frag3_global
preprocessor frag3_engine
```

Advanced Configuration

```
preprocessor frag3_global: prealloc_nodes 8192
preprocessor frag3_engine: policy linux, bind_to 192.168.1.0/24
preprocessor frag3_engine: policy first, bind_to [10.1.47.0/24,172.16.8.0/24]
preprocessor frag3_engine: policy last, detect_anomalies
```

Frag 3 Alert Output

Frag3 is capable of detecting eight different types of anomalies. Its event output is packet-based so it will work with all output modes of Snort. Read the documentation in the `doc/signatures` directory with filenames that begin with "123-" for information on the different event types.

2.2.2 Stream5

The Stream5 preprocessor is a target-based TCP reassembly module for Snort. It is capable of tracking sessions for both TCP and UDP. With Stream5, the rule 'flow' and 'flowbits' keywords are usable with TCP as well as UDP traffic.

Transport Protocols

TCP sessions are identified via the classic TCP "connection". UDP sessions are established as the result of a series of UDP packets from two end points via the same set of ports. ICMP messages are tracked for the purposes of checking for unreachable and service unavailable messages, which effectively terminate a TCP or UDP session.



Target-Based

Stream5, like Frag3, introduces target-based actions for handling of overlapping data and other TCP anomalies. The methods for handling overlapping data, TCP Timestamps, Data on SYN, FIN and Reset sequence numbers, etc. and the policies supported by Stream5 are the results of extensive research with many target operating systems.

Stream API

Stream5 fully supports the Stream API, other protocol normalizers/preprocessors to dynamically configure reassembly behavior as required by the application layer protocol, identify sessions that may be ignored (large data transfers, etc), and update the identifying information about the session (application protocol, direction, etc) that can later be used by rules.

Anomaly Detection

TCP protocol anomalies, such as data on SYN packets, data received outside the TCP window, etc are configured via the `detect_anomalies` option to the TCP configuration. Some of these anomalies are detected on a per-target basis. For example, a few operating systems allow data in TCP SYN packets, while others do not.

Stream5 Global Configuration

Global settings for the Stream5 preprocessor.

```
preprocessor stream5_global: \
    [track_tcp <yes|no>], [max_tcp <number>], \
    [memcap <number bytes>], \
    [track_udp <yes|no>], [max_udp <number>], \
    [track_icmp <yes|no>], [max_icmp <number>], \
    [flush_on_alert], [show_rebuilt_packets], \
    [prune_log_max <bytes>], [disabled]
```


Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 42

03-02-2022-24064301-HauptPost-0580

Option	Description
track_tcp <yes|no>	Track sessions for TCP. The default is "yes".
max_tcp <num sessions>	Maximum simultaneous TCP sessions tracked. The default is "262144", maximum is "1048576", minimum is "1".
memcap <num bytes>	Memcap for TCP packet storage. The default is "8388608" (8MB), maximum is "1073741824" (1GB), minimum is "32768" (32KB).
track_udp <yes|no>	Track sessions for UDP. The default is "yes".
max_udp <num sessions>	Maximum simultaneous UDP sessions tracked. The default is "131072", maximum is "1048576", minimum is "1".
track_icmp <yes|no>	Track sessions for ICMP. The default is "no".
max_icmp <num sessions>	Maximum simultaneous ICMP sessions tracked. The default is "65536", maximum is "1048576", minimum is "1".
disabled	Option to disable the stream5 tracking. By default this option is turned off. When the preprocessor is disabled only the options memcap, max_tcp, max_udp and max_icmp are applied when specified with the configuration.
flush_on_alert	Backwards compatibility. Flush a TCP stream when an alert is generated on that stream. The default is set to off.
show_rebuilt_packets	Print/display packet after rebuilt (for debugging). The default is set to off.
prune_log_max <num bytes>	Print a message when a session terminates that was consuming more than the specified number of bytes. The default is "1048576" (1MB), minimum can be either "0" (disabled) or if not disabled the minimum is "1024" and maximum is "1073741824".

16444709564601258531 9

Stream5 TCP Configuration

Provides a means on a per IP address target to configure TCP policy. This can have multiple occurrences, per policy that is bound to an IP address or network. One default policy must be specified, and that policy is not bound to an IP address or network.

```
preprocessor stream5_tcp: \
    [bind_to <ip_addr>], \
    [timeout <number secs>], [policy <policy_id>], \
    [overlap_limit <number>], [max_window <number>], \
    [require_3whs [<number secs>]], [detect_anomalies], \
    [check_session_hijacking], [use_static_footprint_sizes], \
    [dont_store_large_packets], [dont_reassemble_async], \
    [max_queued_bytes <bytes>], [max_queued_segs <number segs>], \
    [small_segments <number> bytes <number> [ignore_ports number [number]*]],  \
    [ports <client|server|both> <all|number [number]*>], \
    [protocol <client|server|both> <all|service name [service name]*>], \
    [ignore_any_rules], [flush_factor <number segs>]
```

Option	Description
bind_to <ip_addr>	IP address or network for this policy. The default is set to any.
timeout <num seconds>	Session timeout. The default is "30", the minimum is "1", and the maximum is "86400" (approximately 1 day).

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 43



<table>
<tr><td>policy <policy_id></td><td>The Operating System policy for the target OS. The policy_id can be one of the following:
<table>
<tr><th>Policy Name</th><th>Operating Systems.</th></tr>
<tr><td>first</td><td>Favor first overlapped segment.</td></tr>
<tr><td>last</td><td>Favor first overlapped segment.</td></tr>
<tr><td>bsd</td><td>FresBSD 4.x and newer, NetBSD 2.x and newer, OpenBSD 3.x and newer</td></tr>
<tr><td>linux</td><td>Linux 2.4 and newer</td></tr>
<tr><td>old-linux</td><td>Linux 2.2 and earlier</td></tr>
<tr><td>windows</td><td>Windows 2000, Windows XP, Windows 95/98/ME</td></tr>
<tr><td>win2003</td><td>Windows 2003 Server</td></tr>
<tr><td>vista</td><td>Windows Vista</td></tr>
<tr><td>solaris</td><td>Solaris 9.x and newer</td></tr>
<tr><td>hpux</td><td>HPUX 11 and newer</td></tr>
<tr><td>hpux10</td><td>HPUX 10</td></tr>
<tr><td>irix</td><td>IRIX 6 and newer</td></tr>
<tr><td>macos</td><td>MacOS 10.3 and newer</td></tr>
</table></td></tr>
<tr><td>overlap_limit <number></td><td>Limits the number of overlapping packets per session. The default is "0" (unlimited), the minimum is "0", and the maximum is "255".</td></tr>
<tr><td>max_window <number></td><td>Maximum TCP window allowed. The default is "0" (unlimited), the minimum is "0", and the maximum is "1073725440" (65535 left shift 14). That is the highest possible TCP window per RFCs. This option is intended to prevent a DoS against Stream5 by an attacker using an abnormally large window, so using a value near the maximum is discouraged.</td></tr>
<tr><td>require_3whs [<number seconds>]</td><td>Establish sessions only on completion of a SYN/SYN-ACK/ACK handshake. The default is set to off. The optional number of seconds specifies a startup timeout. This allows a grace period for existing sessions to be considered established during that interval immediately after Snort is started. The default is "0" (don't consider existing sessions established), the minimum is "0", and the maximum is "86400" (approximately 1 day).</td></tr>
<tr><td>detect_anomalies</td><td>Detect and alert on TCP protocol anomalies. The default is set to off.</td></tr>
<tr><td>check_session_hijacking</td><td>Check for TCP session hijacking. This check validates the hardware (MAC) address from both sides of the connect – as established on the 3-way handshake against subsequent packets received on the session. If an ethernet layer is not part of the protocol stack received by Snort, there are no checks performed. Alerts are generated (per 'detect_anomalies' option) for either the client or server when the MAC address for one side or the other does not match. The default is set to off.</td></tr>
<tr><td>use_static_footprint_sizes</td><td>Use static values for determining when to build a reassembled packet to allow for repeatable tests. This option should not be used production environments. The default is set to off.</td></tr>
<tr><td>dont_store_large_packets</td><td>Performance improvement to not queue large packets in reassembly buffer. The default is set to off. Using this option may result in missed attacks.</td></tr>
<tr><td>dont_reassemble_async</td><td>Don't queue packets for reassembly if traffic has not been seen in both directions. The default is set to queue packets.</td></tr>
<tr><td>max_queued_bytes <bytes></td><td>Limit the number of bytes queued for reassembly on a given TCP session to bytes. Default is "1048576" (1MB). A value of "0" means unlimited, with a non-zero minimum of "1024", and a maximum of "1073741824" (1GB). A message is written to console/syslog when this limit is enforced.</td></tr>
</table>

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 44

03-02-2022-24064301-Hauptpost-0581

max_queued_segs <num>	Limit the number of segments queued for reassembly on a given TCP session. The default is "2621", derived based on an average size of 400 bytes. A value of "0" means unlimited, with a non-zero minimum of "2", and a maximum of "1073741824" (1GB). A message is written to console/syslog when this limit is enforced.
small_segments <number> bytes <number> [ignore_ports <number(s)>]	Configure the maximum small segments queued. This feature requires that detect_anomalies be enabled. The first number is the number of consecutive segments that will trigger the detection rule. The default value is "0" (disabled), with a maximum of "2048". The second number is the minimum bytes for a segment to be considered "small". The default value is "0" (disabled), with a maximum of "2048". ignore_ports is optional, defines the list of ports in which will be ignored for this rule. The number of ports can be up to "65535". A message is written to console/syslog when this limit is enforced.
ports <client|server|both> <all|number(s)>	Specify the client, server, or both and list of ports in which to perform reassembly. This can appear more than once in a given config. The default settings are ports client 21 23 25 42 53 80 110 111 135 136 137 139 143 445 513 514 1433 1521 2401 3306. The minimum port allowed is "1" and the maximum allowed is "65535".
protocol <client|server|both> <all|service name(s)>	Specify the client, server, or both and list of services in which to perform reassembly. This can appear more than once in a given config. The default settings are ports client ftp telnet smtp nameserver dns http pop3 sunrpc dcerpc netbios-ssn imap login shell mssql oracle cvs mysql. The service names can be any of those used in the host attribute table (see 2.7), including any of the internal defaults (see 2.7.3) or others specific to the network.
ignore_any_rules	Don't process any -> any (ports) rules for TCP that attempt to match payload if there are no port specific rules for the src or destination port. Rules that have flow or flowbits will never be ignored. This is a performance improvement and may result in missed attacks. Using this does not affect rules that look at protocol headers, only those with content, PCRE, or byte test options. The default is "off". This option can be used only in default policy.
flush_factor	Useful in ips mode to flush upon seeing a drop in segment size after N segments of non-decreasing size. The drop in size often indicates an end of request or response.



NOTE

If no options are specified for a given TCP policy, that is the default TCP policy. If only a bind_to option is used with no other options that TCP policy uses all of the default values.

Stream5 UDP Configuration

Configuration for UDP session tracking. Since there is no target based binding, there should be only one occurrence of the UDP configuration.

```
preprocessor stream5_udp: [timeout <number secs>], [ignore_any_rules]
```

Option	Description
timeout <num seconds>	Session timeout. The default is "30", the minimum is "1", and the maximum is "86400" (approximately 1 day).
ignore_any_rules	Don't process any -> any (ports) rules for UDP that attempt to match payload if there are no port specific rules for the src or destination port. Rules that have flow or flowbits will never be ignored. This is a performance improvement and may result in missed attacks. Using this does not affect rules that look at protocol headers, only those with content, PCRE, or byte test options. The default is "off".

NOTE

With the ignore_any_rules option, a UDP rule will be ignored except when there is another port specific rule that may be applied to the traffic. For example, if a UDP rule specifies destination port 53, the 'ignored' any -> any rule will be applied to traffic to/from port 53, but NOT to any other source or destination port. A list of rule SIDs affected by this option are printed at Snort's startup.

NOTE

With the ignore_any_rules option, if a UDP rule that uses any -> any ports includes either flow or flowbits, the ignore_any_rules option is effectively pointless. Because of the potential impact of disabling a flowbits rule, the ignore_any_rules option will be disabled in this case.

1644470956460125853 19

Stream5 ICMP Configuration

Configuration for ICMP session tracking. Since there is no target based binding, there should be only one occurrence of the ICMP configuration.

NOTE

ICMP is currently untested, in minimal code form and is NOT ready for use in production networks. It is not turned on by default.

```
preprocessor stream5_icmp: [timeout <number secs>]
```

Option	Description
timeout <num seconds>	Session timeout. The default is "30", the minimum is "1", and the maximum is "86400" (approximately 1 day).

Example Configurations

1. This example configuration is the default configuration in snort.conf and can be used for repeatable tests of stream reassembly in readback mode.

```
preprocessor stream5_global: \
    max_tcp 8192, track_tcp yes, track_udp yes, track_icmp no

preprocessor stream5_tcp: \
    policy first, use_static_footprint_sizes

preprocessor stream5_udp: \
    ignore_any_rules
```

2. This configuration maps two network segments to different OS policies, one for Windows and one for Linux, with all other traffic going to the default policy of Solaris.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 46

```
preprocessor stream5_global: track_tcp yes
preprocessor stream5_tcp: bind_to 192.168.1.0/24, policy windows
preprocessor stream5_tcp: bind_to 10.1.1.0/24, policy linux
preprocessor stream5_tcp: policy solaris
```

2.2.3 sfPortscan

The sfPortscan module, developed by Sourcefire, is designed to detect the first phase in a network attack: Reconnaissance. In the Reconnaissance phase, an attacker determines what types of network protocols or services a host supports. This is the traditional place where a portscan takes place. This phase assumes the attacking host has no prior knowledge of what protocols or services are supported by the target; otherwise, this phase would not be necessary.

As the attacker has no beforehand knowledge of its intended target, most queries sent by the attacker will be negative (meaning that the service ports are closed). In the nature of legitimate network communications, negative responses from hosts are rare, and rarer still are multiple negative responses within a given amount of time. Our primary objective in detecting portscans is to detect and track these negative responses.

One of the most common portscanning tools in use today is Nmap. Nmap encompasses many, if not all, of the current portscanning techniques. sfPortscan was designed to be able to detect the different types of scans Nmap can produce.



sfPortscan will currently alert for the following types of Nmap scans:

- TCP Portscan
- UDP Portscan
- IP Portscan

These alerts are for one→one portscans, which are the traditional types of scans; one host scans multiple ports on another host. Most of the port queries will be negative, since most hosts have relatively few services available.

sfPortscan also alerts for the following types of decoy portscans:

- TCP Decoy Portscan
- UDP Decoy Portscan
- IP Decoy Portscan

Decoy portscans are much like the Nmap portscans described above, only the attacker has a spoofed source address inter-mixed with the real scanning address. This tactic helps hide the true identity of the attacker.

sfPortscan alerts for the following types of distributed portscans:

- TCP Distributed Portscan
- UDP Distributed Portscan
- IP Distributed Portscan

These are many→one portscans. Distributed portscans occur when multiple hosts query one host for open services. This is used to evade an IDS and obfuscate command and control hosts.

⚠NOTE

Negative queries will be distributed among scanning hosts, so we track this type of scan through the scanned host.

sfPortscan alerts for the following types of portsweeps:

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 47

- TCP Portsweep
- UDP Portsweep
- IP Portsweep
- ICMP Portsweep

These alerts are for one→many portsweeps. One host scans a single port on multiple hosts. This usually occurs when a new exploit comes out and the attacker is looking for a specific service.

NOTE

The characteristics of a portsweep scan may not result in many negative responses. For example, if an attacker portsweeps a web farm for port 80, we will most likely not see many negative responses.

sfPortscan alerts on the following filtered portscans and portsweeps:

- TCP Filtered Portscan
- UDP Filtered Portscan
- IP Filtered Portscan
- TCP Filtered Decoy Portscan
- UDP Filtered Decoy Portscan
- IP Filtered Decoy Portscan
- TCP Filtered Portsweep
- UDP Filtered Portsweep
- IP Filtered Portsweep
- ICMP Filtered Portsweep
- TCP Filtered Distributed Portscan
- UDP Filtered Distributed Portscan
- IP Filtered Distributed Portscan

"Filtered" alerts indicate that there were no network errors (ICMP unreachables or TCP RSTs) or responses on closed ports have been suppressed. It's also a good indicator of whether the alert is just a very active legitimate host. Active hosts, such as NATs, can trigger these alerts because they can send out many connection attempts within a very small amount of time. A filtered alert may go off before responses from the remote hosts are received.

sfPortscan only generates one alert for each host pair in question during the time window (more on windows below). On TCP scan alerts, sfPortscan will also display any open ports that were scanned. On TCP sweep alerts however, sfPortscan will only track open ports after the alert has been triggered. Open port events are not individual alerts, but tags based on the original scan alert.

1644470956460125853 19

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 48

sfPortscan Configuration

Use of the Stream5 preprocessor is required for sfPortscan. Stream gives portscan direction in the case of connectionless protocols like ICMP and UDP. You should enable the Stream preprocessor in your `snort.conf`, as described in Section 2.2.2.

The parameters you can use to configure the portscan module are:

1. **proto <protocol>**
   Available options:
   - `TCP`
   - `UDP`
   - `IGMP`
   - `ip_proto`
   - `all`
2. **scan_type <scan_type>**
   Available options:
   - `portscan`
   - `portsweep`
   - `decoy_portscan`
   - `distributed_portscan`
   - `all`
3. **sense_level <level>**
   Available options:
   - `low` - "Low" alerts are only generated on error packets sent from the target host, and because of the nature of error responses, this setting should see very few false positives. However, this setting will never trigger a Filtered Scan alert because of a lack of error responses. This setting is based on a static time window of 60 seconds, after which this window is reset.
   - `medium` - "Medium" alerts track connection counts, and so will generate filtered scan alerts. This setting may false positive on active hosts (NATs, proxies, DNS caches, etc), so the user may need to deploy the use of Ignore directives to properly tune this directive.
   - `high` - "High" alerts continuously track hosts on a network using a time window to evaluate portscan statistics for that host. A "High" setting will catch some slow scans because of the continuous monitoring, but is very sensitive to active hosts. This most definitely will require the user to tune sfPortscan.
4. **watch_ip <ip1|ip2/cidr[[port|port2-port3]]>**
   Defines which IPs, networks, and specific ports on those hosts to watch. The list is a comma separated list of IP addresses, IP address using CIDR notation. Optionally, ports are specified after the IP address/CIDR using a space and can be either a single port or a range denoted by a dash. IPs or networks not falling into this range are ignored if this option is used.
5. **ignore_scanners <ip1|ip2/cidr[[port|port2-port3]]>**
   Ignores the source of scan alerts. The parameter is the same format as that of `watch_ip`.
6. **ignore_scanned <ip1|ip2/cidr[[port|port2-port3]]>**
   Ignores the destination of scan alerts. The parameter is the same format as that of `watch_ip`.
7. **logfile <file>**
   This option will output portscan events to the file specified. If `file` does not contain a leading slash, this file will be placed in the Snort config dir.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 49

16444709564601258531 9

8. **include_midstream**

   This option will include sessions picked up in midstream by Stream5. This can lead to false alerts, especially under heavy load with dropped packets; which is why the option is off by default.

9. **detect_ack_scans**

   This option will include sessions picked up in midstream by the stream module, which is necessary to detect ACK scans. However, this can lead to false alerts, especially under heavy load with dropped packets; which is why the option is off by default.

10. **disabled**

    This optional keyword is allowed with any policy to avoid packet processing. This option disables the preprocessor. When the preprocessor is disabled only the memcap option is applied when specified with the configuration. The other options are parsed but not used. Any valid configuration may have "disabled" added to it.

Format

```
preprocessor sfportscan: proto <protocols> \
    scan_type <portscan|portsweep|decoy_portscan|distributed_portscan|all> \
    sense_level <low|medium|high> \
    watch_ip <IP or IP/CIDR> \
    ignore_scanners <IP list> \
    ignore_scanned <IP list> \
    logfile <path and filename> \
    disabled
```



Example

```
preprocessor flow: stats_interval 0 hash 2
preprocessor sfportscan:\
    proto { all } \
    scan_type { all } \
    sense_level { low }
```

sfPortscan Alert Output

Unified Output In order to get all the portscan information logged with the alert, snort generates a pseudo-packet and uses the payload portion to store the additional portscan information of priority count, connection count, IP count, port count, IP range, and port range. The characteristics of the packet are:

```
Src/Dst MAC Addr == MACDAD
IP Protocol == 255
IP TTL == 0
```

Other than that, the packet looks like the IP portion of the packet that caused the portscan alert to be generated. This includes any IP options, etc. The payload and payload size of the packet are equal to the length of the additional portscan information that is logged. The size tends to be around 100 - 200 bytes.

Open port alerts differ from the other portscan alerts, because open port alerts utilize the tagged packet output system. This means that if an output system that doesn't print tagged packets is used, then the user won't see open port alerts. The open port information is stored in the IP payload and contains the port that is open.

The sfPortscan alert output was designed to work with unified packet logging, so it is possible to extend favorite Snort GUIs to display portscan alerts and the additional information in the IP payload using the above packet characteristics.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 50