Snort 3 User Manual

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents


```
 ,,_     -*> Snort++ <*-
o"  )~   Version 3.0.0 (Build 247) from 2.9.11
 ''''    By Martin Roesch & The Snort Team
         http://snort.org/contact#team
         Copyright (C) 2014-2018 Cisco and/or its affiliates. All rights reserved.
         Copyright (C) 1998-2013 Sourcefire, Inc., et al.
```

1 Overview

Snort 3.0 is an updated version of the Snort Intrusion Prevention System (IPS) which features a new design that provides a superset of Snort 2.X functionality with better throughput, detection, scalability, and usability. Some of the key features of Snort 3.0 are:

- Support multiple packet processing threads
- Use a shared configuration and attribute table
- Autodetect services for portless configuration
- Modular design
- Plugin framework with over 200 plugins
- More scalable memory profile
- LuaJIT configuration, loggers, and rule options
- Hyperscan support
- Rewritten TCP handling
- New rule parser and syntax
- Service rules like alert http
- Rule "sticky" buffers

- Way better SO rules
- New HTTP inspector
- New performance monitor
- New time and space profiling
- New latency monitoring and enforcement
- Piglets to facilitate component testing
- Inspection Events
- Automake and Cmake
- Autogenerate reference documentation

Additional features are on the road map:

- Use a shared network map
- Support hardware offload for fast pattern acceleration
- Provide support for DPDK and ODP
- Support pipelining of packet processing
- Support proxy mode
- Multi-tennant support
- Incremental reload
- New serialization of perf data and events
- Enhanced rule processing
- Windows support
- Anomaly detection
- and more!

The remainder of this section provides a high level survey of the inputs, processing, and outputs available with Snort 3.0.

Snort++ is the project that is creating Snort 3.0. In this manual "Snort" or "Snort 3" refers to the 3.0 version and earlier versions will be referred to as "Snort 2" where the distinction is relevant.

1.1 First Steps

Snort can be configured to perform complex packet processing and deep packet inspection but it is best start simply and work up to more interesting tasks. Snort won't do anything you didn't specifically ask it to do so it is safe to just try things out and see what happens. Let's start by just running Snort with no arguments:

```
$ snort
```

That will output usage information including some basic help commands. You should run all of these commands now to see what is available:

```
$ snort -V
$ snort -?
$ snort --help
```

Note that Snort has extensive command line help available so if anything below isn't clear, there is probably a way to get the exact information you need from the command line.

Now let's examine the packets in a capture file (pcap):

```
$ snort -r a.pcap
```

Snort will decode and count the packets in the file and output some statistics. Note that the output excludes non-zero numbers so it is easy to see what is there.

You may have noticed that there are command line options to limit the number of packets examined or set a filter to select particular packets. Now is a good time to experiment with those options.

If you want to see details on each packet, you can dump the packets to console like this:

```
$ snort -r a.pcap -L dump
```

Add the -d option to see the TCP and UDP payload. Now let's switch to live traffic. Replace eth0 in the below command with an available network interface:

```
$ snort -i eth0 -L dump
```

Unless the interface is taken down, Snort will just keep running, so enter Control-C to terminate or use the -n option to limit the number of packets.

Generally it is better to capture the packets for later analysis like this:

```
$ snort -i eth0 -L pcap -n 10
```

Snort will write 10 packets to log.pcap.# where # is a timestamp value. You can read these back with -r and dump to console or pcap with -L. You get the idea.

Note that you can do similar things with other tools like tcpdump or Wireshark however these commands are very useful when you want to check your Snort setup.

The examples above use the default pcap DAQ. Snort supports non-pcap interfaces as well via the DAQ (data acquisition) library. Other DAQs provide additional functionality such as inline operation and/or higher performance. There are even DAQs that support raw file processing (ie without packets), socket processing, and plain text packets. To load external DAQ libraries and see available DAQs or select a particular DAQ use one of these commands:

```
$ snort --daq-dir <path> --daq-list
$ snort --daq-dir <path> --daq <type>
```

Be sure to put the --daq-dir option ahead of the --daq-list option or the external DAQs won't appear in the list.

To leverage intrusion detection features of Snort you will need to provide some configuration details. The next section breaks down what must be done.

1.2 Configuration

Effective configuration of Snort is done via the environment, command line, a Lua configuration file, and a set of rules.

Note that backwards compatibility with Snort 2 was sacrificed to obtain new and improved functionality. While Snort 3 leverages some of the Snort 2 code base, a lot has changed. The configuration of Snort 3 is done with Lua, so your old conf won't work as is. Rules are still text based but with syntax tweaks, so your 2.X rules must be fixed up. However, snort2lua will help you convert your conf and rules to the new format.

1.2.1 Environment

LUA_PATH must be set based on your install:

```
LUA_PATH=$install_prefix/include/snort/lua/\?.lua\;\;
```

SNORT_LUA_PATH must be set to load auxiliary configuration files if you use the default snort.lua. For example:

```
export SNORT_LUA_PATH=$install_prefix/etc/snort
```

1.2.2 Command Line

A simple command line might look like this:

```
snort -c snort.lua -R cool.rules -r some.pcap -A cmg
```

To understand what that does, you can start by just running snort with no arguments by running snort --help. Help for all configuration and rule options is available via a suitable command line. In this case:

-c snort.lua is the main configuration file. This is a Lua script that is executed when loaded.

-R cool.rules contains some detection rules. You can write your own or obtain them from Talos (native 3.0 rules are not yet available from Talos so you must convert them with snort2lua). You can also put your rules directly in your configuration file.

-r some.pcap tells Snort to read network traffic from the given packet capture file. You could instead use -i eth0 to read from a live interface. There many other options available too depending on the DAQ you use.

-A cmg says to output intrusion events in "cmg" format, which has basic header details followed by the payload in hex and text.

Note that you add to and/or override anything in your configuration file by using the --lua command line option. For example:

```
--lua 'ips = { enable_builtin_rules = true }'
```

will load the built-in decoder and inspector rules. In this case, ips is overwritten with the config you see above. If you just want to change the config given in your configuration file you would do it like this:

```
--lua 'ips.enable_builtin_rules = true'
```

1.2.3 Configuration File

The configuration file gives you complete control over how Snort processes packets. Start with the default snort.lua included in the distribution because that contains some key ingredients. Note that most of the configurations look like:

```
stream = { }
```

This means enable the stream module using internal defaults. To see what those are, you could run:

```
snort --help-config stream
```

Snort is organized into a collection of builtin and plugin modules. If a module has parameters, it is configured by a Lua table of the same name. For example, we can see what the active module has to offer with this command:

```
$ snort --help-module active

What: configure responses

Type: basic
```

```
Configuration:

int active.attempts = 0: number of TCP packets sent per response (with
varying sequence numbers) { 0:20 }

string active.device: use 'ip' for network layer responses or 'eth0' etc
for link layer

string active.dst_mac: use format '01:23:45:67:89:ab'

int active.max_responses = 0: maximum number of responses { 0: }

int active.min_interval = 255: minimum number of seconds between
responses { 1: }
```

This says active is a basic module that has several parameters. For each, you will see:

```
type module.name = default: help { range }
```

For example, the active module has a max_responses parameter that takes non-negative integer values and defaults to zero. We can change that in Lua as follows:

```
active = { max_responses = 1 }
```

or:

```
active = { }
active.max_responses = 1
```

If we also wanted to limit retries to at least 5 seconds, we could do:

```
active = { max_responses = 1, min_interval = 5 }
```

1.2.4 Rules

Rules determine what Snort is looking for. They can be put directly in your Lua configuration file with the ips module, on the command line with --lua, or in external files. Generally you will have many rules obtained from various sources such as Talos and loading external files is the way to go so we will summarize that here. Add this to your Lua configuration:

```
ips = { include = 'rules.txt' }
```

to load the external rules file named rules.txt. You can only specify one file this way but rules files can include other rules files with the include statement. In addition you can load rules like:

```
$ sort -c snort.lua -R rules.txt
```

You can use both approaches together.

1.2.5 Converting Your 2.X Configuration

If you have a working 2.X configuration snort2lua makes it easy to get up and running with Snort 3. This tool will convert your configuration and/or rules files automatically. You will want to clean up the results and double check that it is doing exactly what you need.

```
snort2lua -c snort.conf
```

The above command will generate snort.lua based on your 2.X configuration. For more information and options for more sophisticated use cases, see the Snort2Lua section later in the manual.

1.3 Output

Snort can produce quite a lot of data. In the following we will summarize the key aspects of the core output types. Additional data such as from appid is covered later.

1.3.1 Basic Statistics

At shutdown, Snort will output various counts depending on configuration and the traffic processed. Generally, you may see:

- Packet Statistics - this includes data from the DAQ and decoders such as the number of packets received and number of UDP packets.
- Module Statistics - each module tracks activity via a set of peg counts that indicate how many times something was observed or performed. This might include the number of HTTP GET requests processed and the number of TCP reset packets trimmed.
- File Statistics - look here for a breakdown of file type, bytes, signatures.
- Summary Statistics - this includes total runtime for packet processing and the packets per second. Profiling data will appear here as well if configured.

Note that only the non-zero counts are output. Run this to see the available counts:

```
$ snort --help-counts
```

1.3.2 Alerts

If you configured rules, you will need to configure alerts to see the details of detection events. Use the -A option like this:

```
$ snort -c snort.lua -r a.pcap -A cmg
```

There are many types of alert outputs possible. Here is a brief list:

- -A cmg is the same as -A fast -d -e and will show information about the alert along with packet headers and payload.
- -A u2 is the same as -A unified2 and will log events and triggering packets in a binary file that you can feed to other tools for post processing. Note that Snort 3 does not provide the raw packets for alerts on PDUs; you will get the actual buffer that alerted.
- -A csv will output various fields in comma separated value format. This is entirely customizable and very useful for pcap analysis.

To see the available alert types, you can run this command:

```
$ snort --list-plugins | grep logger
```

1.3.3 Files and Paths

Note that output is specific to each packet thread. If you run 4 packet threads with u2 output, you will get 4 different u2 files. The basic structure is:

```
<logdir>/[<run_prefix>][<id#>][<X>]<name>
```

where:

- logdir is set with -l and defaults to ./

- run_prefix is set with --run-prefix else not used
- id# is the packet thread number that writes the file; with one packet thread, id# (zero) is omitted without --id-zero
- X is / if you use --id-subdir, else _ if id# is used
- name is based on module name that writes the file

Additional considerations:

- There is no way to explicitly configure a full path to avoid issues with multiple packet threads.
- All text mode outputs default to stdout

1.3.4 Performance Statistics

Still more data is available beyond the above.

- By configuring the perf_monitor module you can capture a configurable set of peg counts during runtime. This is useful to feed to an external program so you can see what is happening without stopping Snort.
- The profiler module allows you to track time and space used by module and rules. Use this data to tune your system for best performance. The output will show up under Summary Statistics at shutdown.

2 Concepts

This section provides background on essential aspects of Snort's operation.

2.1 Terminology

- **basic module**: a module integrated into Snort that does not come from a plugin.
- **binder**: inspector that maps configuration to traffic
- **builtin rules**: codec and inspector rules for anomalies detected internally.
- **codec**: short for coder / decoder. These plugins are used for basic protocol decoding, anomaly detection, and construction of active responses.
- **data module**: an adjunct configuration plugin for use with certain inspectors.
- **dynamic rules**: plugin rules loaded at runtime. See SO rules.
- **fast pattern**: the content in an IPS rule that must be found by the search engine in order for a rule to be evaluated.
- **fast pattern matcher**: see search engine.
- **hex**: a type of protocol magic that the wizard uses to identify binary protocols.
- **inspector**: plugin that processes packets (similar to the Snort 2 preprocessor)
- **IPS**: intrusion prevention system, like Snort.
- **IPS action**: plugin that allows you to perform custom actions when events are generated. Unlike loggers, these are invoked before thresholding and can be used to control external agents or send active responses.
- **IPS option**: this plugin is the building blocks of IPS rules.
- **logger**: a plugin that performs output of events and packets. Events are thresholded before reaching loggers.

- **module**: the user facing portion of a Snort component. Modules chiefly provide configuration parameters, but may also provide commands, builtin rules, profiling statistics, peg counts, etc. Note that not all modules are plugins and not all plugins have modules.
- **peg count**: the number of times a given event or condition occurs.
- **plugin**: one of several types of software components that can be loaded from a dynamic library when Snort starts up. Some plugins are coupled with the main engine in such a way that they must be built statically, but a newer version can be loaded dynamically.
- **search engine**: a plugin that performs multipattern searching of packets and payload to find rules that should be evaluated. There are currently no specific modules, although there are several search engine plugins. Related configuration is done with the basic detection module. Aka fast pattern matcher.
- **SO rule**: a IPS rule plugin that performs custom detection that can't be done by a text rule. These rules typically do not have associated modules. SO comes from shared object, meaning dynamic library.
- **spell**: a type of protocol magic that the wizard uses to identify ASCII protocols.
- **text rule**: a rule loaded from the configuration that has a header and body. The header specifies action, protocol, source and destination IP addresses and ports, and direction. The body specifies detection and non-detection options.
- **wizard**: inspector that applies protocol magic to determine which inspectors should be bound to traffic absent a port specific binding. See hex and spell.

2.2 Modules

Modules are the building blocks of Snort. They encapsulate the types of data that many components need including parameters, peg counts, profiling, builtin rules, and commands. This allows Snort to handle them generically and consistently. You can learn quite a lot about any given module from the command line. For example, to see what stream_tcp is all about, do this:

```
$ snort --help-config stream_tcp
```

Modules are configured using Lua tables with the same name. So the stream_tcp module is configured with defaults like this:

```
stream_tcp = { }
```

The earlier help output showed that the default session tracking timeout is 30 seconds. To change that to 60 seconds, you can configure it this way:

```
stream_tcp = { session_timeout = 60 }
```

Or this way:

```
stream_tcp = { }
stream_tcp.session_timeout = 60
```

More on parameters is given in the next section.

Other things to note about modules:

- Shutdown output will show the non-zero peg counts for all modules. For example, if stream_tcp did anything, you would see the number of sessions processed among other things.
- Providing the builtin rules allows the documentation to include them automatically and also allows for autogenerating the rules at startup.
- Only a few module provide commands at this point, most notably the snort module.

2.3 Parameters

Parameters are given with this format:

```
type name = default: help { range }
```

The following types are used:

- **addr**: any valid IP4 or IP6 address or CIDR
- **addr_list**: a space separated list of addr values
- **bit_list**: a list of consecutive integer values from 1 to the range maximum
- **bool**: true or false
- **dynamic**: a select type determined by loaded plugins
- **enum**: a string selected from the given range
- **implied**: an IPS rule option that takes no value but means true
- **int**: a whole number in the given range
- **interval**: a set of ints (see below)
- **ip4**: an IP4 address or CIDR
- **mac**: an ethernet address with the form 01:02:03:04:05:06
- **multi**: one or more space separated strings from the given range
- **port**: an int in the range 0:65535 indicating a TCP or UDP port number
- **real**: a real number in the given range
- **select**: a string selected from the given range
- **string**: any string with no more than the given length, if any

The parameter name may be adorned in various ways to indicate additional information about the type and use of the parameter:

- For Lua configuration (not IPS rules), if the name ends with [] it is a list item and can be repeated.
- For IPS rules only, names starting with ~ indicate positional parameters. The names of such parameters do not appear in the rule.
- IPS rules may also have a wild card parameter, which is indicated by a *. Used for unquoted, comma-separated lists such as service and metadata.
- The snort module has command line options starting with a -.

Some additional details to note:

- Table and variable names are case sensitive; use lower case only.
- String values are case sensitive too; use lower case only.
- Numeric ranges may be of the form low:high where low and high are bounds included in the range. If either is omitted, there is no hard bound. E.g. 0: means any x where x >= 0.
- Strings may have a numeric range indicating a length limit; otherwise there is no hard limit.
- bit_list is typically used to store a set of byte, port, or VLAN ID values.
- interval takes the form [operator]i, j<>k, or j<⇒k where i,j,k are integers and operator is one of =, !, != (same as !), <, ⇐, >, >=. j<>k means j < int < k and j<⇒k means j ⇐ int ⇐ k.

2.4 Plugins

Snort uses a variety of plugins to accomplish much of its processing objectives, including:

- Codec - to decode and encode packets
- Inspector - like Snort 2 preprocessors, for normalization, etc.
- IpsOption - for detection in Snort rules
- IpsAction - for custom actions
- Logger - for handling events
- Mpse - for fast pattern matching
- So - for dynamic rules

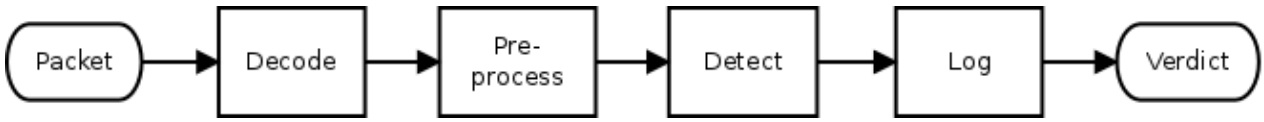
The power of plugins is that they have a very focused purpose and can be created with relative ease. For example, you can extend the rule language by writing your own IpsOption and it will plug in and function just like existing options. The extra directory has examples of each type of plugin.

Most plugins can be built statically or dynamically. By default they are all static. There is no difference in functionality between static or dynamic plugins but the dynamic build generates a slightly lighter weight binary. Either way you can add dynamic plugins with --plugin-path and newer versions will replace older versions, even when built statically.

A single dynamic library may contain more than one plugin. For example, an inspector will typically be packaged together with any associated rule options.

2.5 Operation

Snort is a signature-based IPS, which means that as it receives network packets it reassembles and normalizes the content so that a set of rules can be evaluated to detect the presence of any significant conditions that merit further action. A rough processing flow is as follows:



The steps are:

1. Decode each packet to determine the basic network characteristics such as source and destination addresses and ports. A typical packet might have ethernet containing IP containing TCP containing HTTP (ie eth:ip:tcp:http). The various encapsulating protocols are examined for sanity and anomalies as the packet is decoded. This is essentially a stateless effort.
2. Preprocess each decoded packet using accumulated state to determine the purpose and content of the innermost message. This step may involve reordering and reassembling IP fragments and TCP segments to produce the original application protocol data unit (PDU). Such PDUs are analyzed and normalized as needed to support further processing.
3. Detection is a two step process. For efficiency, most rules contain a specific content pattern that can be searched for such that if no match is found no further processing is necessary. Upon start up, the rules are compiled into pattern groups such that a single, parallel search can be done for all patterns in the group. If any match is found, the full rule is examined according to the specifics of the signature.
4. The logging step is where Snort saves any pertinent information resulting from the earlier steps. More generally, this is where other actions can be taken as well such as blocking the packet.

2.5.1 Snort 2 Processing

The preprocess step in Snort 2 is highly configurable. Arbitrary preprocessors can be loaded dynamically at startup, configured in snort.conf, and then executed at runtime. Basically, the preprocessors are put into a list which is iterated for each packet. Recent versions have tweaked the list handling some, but the same basic architecture has allowed Snort 2 to grow from a sniffer, with no preprocessing, to a full-fledged IPS, with lots of preprocessing.

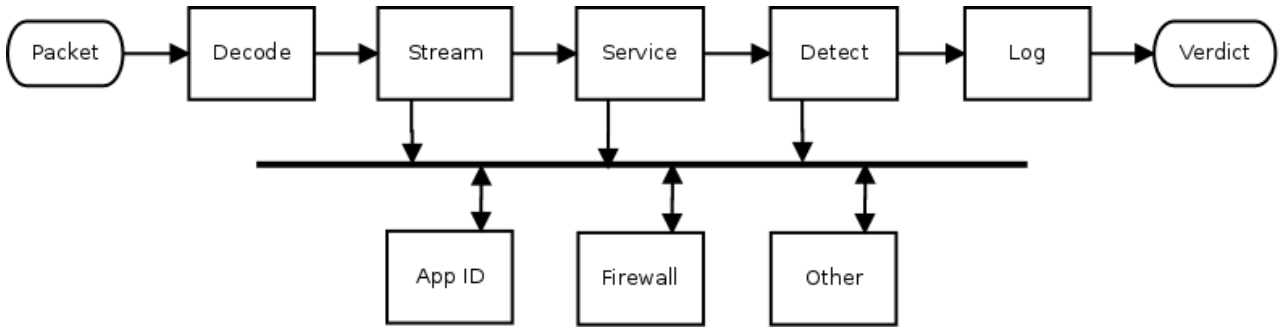
While this "list of plugins" approach has considerable flexibility, it hampers future development when the flow of data from one preprocessor to the next depends on traffic conditions, a common situation with advanced features like application identification. In this case, a preprocessor like HTTP may be extracting and normalizing data that ultimately is not used, or appID may be repeatedly checking for data that is just not available.

Callbacks help break out of the preprocess straitjacket. This is where one preprocessor supplies another with a function to call when certain data is available. Snort has started to take this approach to pass some HTTP and SIP preprocessor data to appID. However, it remains a peripheral feature and still requires the production of data that may not be consumed.

2.5.2 Snort 3 Processing

One of the goals of Snort 3 is to provide a more flexible framework for packet processing by implementing an event-driven approach. Another is to produce data only when needed to minimize expensive normalizations. However, the basic packet processing provides very similar functionality.

The basic processing steps Snort 3 takes are similar to Snort 2 as seen in the following diagram. The preprocess step employs specific inspector types instead of a generalized list, but the basic procedure includes stateless packet decoding, TCP stream reassembly, and service specific analysis in both cases. (Snort 3 provides hooks for arbitrary inspectors, but they are not central to basic flow processing and are not shown.)



However, Snort 3 also provides a more flexible mechanism than callback functions. By using inspection events, it is possible for an inspector to supply data that other inspectors can process. This is known as the observer pattern or publish-subscribe pattern.

Note that the data is not actually published. Instead, access to the data is published, and that means that subscribers can access the raw or normalized version(s) as needed. Normalizations are done only on the first access, and subsequent accesses get the previously normalized data. This results in just in time (JIT) processing.

A basic example of this in action is provided by the extra data_log plugin. It is a passive inspector, ie it does nothing until it receives the data it subscribed for (*other* in the above diagram). By adding the following to your snort.lua configuration, you will get a simple URI logger.

```
data_log = { key = 'http_raw_uri' }
```

Inspection events coupled with pluggable inspectors provide a very flexible framework for implementing new features. And JIT buffer stuffers allow Snort to work smarter, not harder. These capabilities will be leveraged more and more as Snort development continues.

2.6 Rules

Rules tell Snort how to detect interesting conditions, such as an attack, and what to do when the condition is detected. Here is an example rule:

```
alert tcp any any -> 192.168.1.1 80 ( msg:"A ha!"; content:"attack"; sid:1; )
```

The structure is:

```
action proto source dir dest ( body )
```

Where:

action - tells Snort what to do when a rule "fires", ie when the signature matches. In this case Snort will log the event. It can also do thing like block the flow when running inline.

proto - tells Snort what protocol applies. This may be ip, icmp, tcp, udp, http, etc.

source - specifies the sending IP address and port, either of which can be the keyword any, which is a wildcard.

dir - must be either unidirectional as above or bidirectional indicated by <>.

dest - similar to source but indicates the receiving end.

body - detection and other information contained in parenthesis.

There are many rule options available to construct as sophisticated a signature as needed. In this case we are simply looking for the "attack" in any TCP packet. A better rule might look like this:

```
alert http
(
    msg:"Gotcha!";
    flow:established, to_server;
    http_uri:"attack";
    sid:2;
)
```

Note that these examples have a sid option, which indicates the signature ID. In general rules are specified by gid:sid:rev notation, where gid is the generator ID and rev is the revision of the rule. By default, text rules are gid 1 and shared-object (SO) rules are gid 3. The various components within Snort that generate events have 1XX gids, for example the decoder is gid 116. You can list the internal gids and sids with these commands:

```
$ snort --list-gids
$ snort --list-builtin
```

For details on these and other options, see the reference section.

2.7 Pattern Matching

Snort evaluates rules in a two-step process which includes a fast pattern search and full evaluation of the signature. More details on this process follow.

2.7.1 Rule Groups

When Snort starts or reloads configuration, rules are grouped by protocol, port and service. For example, all TCP rules using the HTTP_PORTS variable will go in one group and all service HTTP rules will go in another group. These rule groups are compiled into multipattern search engines (MPSE) which are designed to search for all patterns with just a single pass through a given packet or buffer. You can select the algorithm to use for fast pattern searches with search_engine.search_method which defaults to *ac_bnfa*, which balances speed and memory. For a faster search at the expense of significantly more memory, use *ac_full*. For best performance and reasonable memory, download the hyperscan source from Intel.

2.7.2 Fast Patterns

Fast patterns are content strings that have the fast_pattern option or which have been selected by Snort automatically to be used as a fast pattern. Snort will by default choose the longest pattern in the rule since that is likely to be most unique. That is not always the case so add fast_pattern to the appropriate content option for best performance. The ideal fast pattern is one which, if found, is very likely to result in a rule match. Fast patterns that match frequently for unrelated traffic will cause Snort to work hard with little to show for it.

Certain contents are not eligible to be used as fast patterns. Specifically, if a content is negated, then if it is also relative to another content, case sensitive, or has non-zero offset or depth, then it is not eligible to be used as a fast pattern.

2.7.3 Rule Evaluation

For each fast pattern match, the corresponding rule(s) are evaluated left-to-right. Rule evaluation requires checking each detection option in a rule and is a fairly costly process which is why fast patterns are so important. Rule evaluation aborts on the first non-matching option.

When rule evaluation takes place, the fast pattern match will automatically be skipped if possible. Note that this differs from Snort 2 which provided the fast_pattern:only option to designate such cases. This is one less thing for the rule writer to worry about.

3 Tutorial

The section will walk you through building and running Snort. It is not exhaustive but, once you master this material, you should be able to figure out more advanced usage.

3.1 Dependencies

Required:

- autotools or cmake to build from source
- daq from http://www.snort.org for packet IO
- g++ >= 4.8 or other recent C++11 compiler
- dnet from https://github.com/dugsong/libdnet.git for network utility functions
- hwloc from https://www.open-mpi.org/projects/hwloc/ for CPU affinity management
- LuaJIT from http://luajit.org for configuration and scripting
- OpenSSL from https://www.openssl.org/source/ for SHA and MD5 file signatures, the protected_content rule option, and SSL service detection
- pcap from http://www.tcpdump.org for tcpdump style logging
- pcre from http://www.pcre.org for regular expression pattern matching
- pkgconfig from https://www.freedesktop.org/wiki/Software/pkg-config/ to locate build dependencies
- zlib from http://www.zlib.net for decompression (>= 1.2.8 recommended)

Optional:

- asciidoc from http://www.methods.co.nz/asciidoc/ to build the HTML manual
- cpputest from http://cpputest.github.io to run additional unit tests with make check

- dblatex from http://dblatex.sourceforge.net to build the pdf manual (in addition to asciidoc)
- flatbuffers from https://google.github.io/flatbuffers/ for enabling the flatbuffers serialization format
- hyperscan >= 4.4.0 from https://github.com/01org/hyperscan to build new the regex and sd_pattern rule options and hyperscan search engine. Hyperscan is large so it recommended to follow their instructions for building it as a shared library.
- iconv from https://ftp.gnu.org/pub/gnu/libiconv/ for converting UTF16-LE filenames to UTF8 (usually included in glibc)
- lzma >= 5.1.2 from http://tukaani.org/xz/ for decompression of SWF and PDF files
- safec from https://sourceforge.net/projects/safeclib/ for runtime bounds checks on certain legacy C-library calls
- source-highlight from http://www.gnu.org/software/src-highlite/ to generate the dev guide
- w3m from http://sourceforge.net/projects/w3m/ to build the plain text manual
- uuid from uuid-dev package for unique identifiers

3.2 Building

- Optionally built features are listed in the reference section.
- Create an install path:

```
export my_path=/path/to/snorty
mkdir -p $my_path
```

- If you are using a github clone with autotools, do this:

```
autoreconf -isvf
```

- Now do one of the following:

  a. To build with cmake and make, run configure_cmake.sh. It will automatically create and populate a new subdirectory named *build*.

  ```
  ./configure_cmake.sh --prefix=$my_path
  cd build
  make -j 8
  make install
  ln -s $my_path/conf $my_path/etc
  ```

  b. You can also specify a cmake project generator:

  ```
  ./configure_cmake.sh --generator=Xcode --prefix=$my_path
  ```

  c. Or use ccmake directly to configure and generate from an arbitrary build directory like one of these:

  ```
  ccmake -G Xcode /path/to/Snort++/tree
  open snort.xcodeproj

  ccmake -G "Eclipse CDT4 - Unix Makefiles" /path/to/Snort++/tree
  run eclipse and do File > Import > Existing Eclipse Project
  ```

- To build with g++ on OS X where clang is installed, do this first:

```
export CXX=g++
```

3.3 Running

First set up the environment:

```
export LUA_PATH=$my_path/include/snort/lua/\?.lua\;\;
export SNORT_LUA_PATH=$my_path/etc/snort/
```

Then give it a go:

- Get some help:

```
$my_path/bin/snort --help
$my_path/bin/snort --help-module suppress
$my_path/bin/snort --help-config | grep thread
```

- Examine and dump a pcap:

```
$my_path/bin/snort -r <pcap>
$my_path/bin/snort -L dump -d -e -q -r <pcap>
```

- Verify config, with or w/o rules:

```
$my_path/bin/snort -c $my_path/etc/snort/snort.lua
$my_path/bin/snort -c $my_path/etc/snort/snort.lua -R $my_path/etc/snort/sample. ↩
    rules
```

- Run IDS mode. To keep it brief, look at the first n packets in each file:

```
$my_path/bin/snort -c $my_path/etc/snort/snort.lua -R $my_path/etc/snort/sample. ↩
    rules \
     -r <pcap> -A alert_test -n 100000
```

- Let's suppress 1:2123. We could edit the conf or just do this:

```
$my_path/bin/snort -c $my_path/etc/snort/snort.lua -R $my_path/etc/snort/sample. ↩
    rules \
     -r <pcap> -A alert_test -n 100000 --lua "suppress = { { gid = 1, sid = 2123 } ↩
         }"
```

- Go whole hog on a directory with multiple packet threads:

```
$my_path/bin/snort -c $my_path/etc/snort/snort.lua -R $my_path/etc/snort/sample. ↩
    rules \
     --pcap-filter \*.pcap --pcap-dir <dir> -A alert_fast -n 1000 --max-packet- ↩
        threads 8
```

For more examples, see the usage section.

3.4 Tips

One of the goals of Snort 3 is to make it easier to configure your sensor. Here is a summary of tips and tricks you may find useful.

General Use

- Snort tries hard not to error out too quickly. It will report multiple semantic errors.

- Snort always assumes the simplest mode of operation. Eg, you can omit the -T option to validate the conf if you don't provide a packet source.
- Warnings are not emitted unless --warn-* is specified. --warn-all enables all warnings, and --pedantic makes such warnings fatal.
- You can process multiple sources at one time by using the -z or --max-threads option.
- To make it easy to find the important data, zero counts are not output at shutdown.
- Load plugins from the command line with --plugin-path /path/to/install/lib.
- You can process multiple sources at one time by using the -z or --max-threads option.
- Unit tests are configured with --enable-unit-tests. They can then be run with snort --catch-test [tags]|all.

Lua Configuration

- Configure the wizard and default bindings will be created based on configured inspectors. No need to explicitly bind ports in this case.
- You can override or add to your Lua conf with the --lua command line option.
- The Lua conf is a live script that is executed when loaded. You can add functions, grab environment variables, compute values, etc.
- You can also rename symbols that you want to disable. For example, changing normalizer to Xnormalizer (an unknown symbol) will disable the normalizer. This can be easier than commenting in some cases.
- By default, symbols unknown to Snort are silently ignored. You can generate warnings for them with --warn-unknown. To ignore such symbols, export them in the environment variable SNORT_IGNORE.

Writing and Loading Rules

Snort rules allow arbitrary whitespace. Multi-line rules make it easier to structure your rule for clarity. There are multiple ways to add comments to your rules:

- The # character starts a comment to end of line. In addition, all lines between #begin and #end are comments.
- The rem option allows you to write a comment that is conveyed with the rule.
- C style multi-line comments are allowed, which means you can comment out portions of a rule while testing it out by putting the options between /* and */.

There are multiple ways to load rules too:

- Set ips.rules or ips.include.
- include statements can be used in rules files.
- Use -R to load a rules file.
- Use --stdin-rules with command line redirection.
- Use --lua to specify one or more rules as a command line argument.

Output Files

To make it simple to configure outputs when you run with multiple packet threads, output files are not explicitly configured. Instead, you can use the options below to format the paths:

```
<logdir>/[<run_prefix>][<id#>][<X>]<name>
```

- logdir is set with -l and defaults to ./
- run_prefix is set with --run-prefix else not used
- id# is the packet thread number that writes the file; with one packet thread, id# (zero) is omitted without --id-zero
- X is / if you use --id-subdir, else _ if id# is used
- name is based on module name that writes the file
- all text mode outputs default to stdout

3.5 Help

```
Snort has several options to get more help:

-? list command line options (same as --help)
--help this overview of help
--help-commands [<module prefix>] output matching commands
--help-config [<module prefix>] output matching config options
--help-counts [<module prefix>] output matching peg counts
--help-module <module> output description of given module
--help-modules list all available modules with brief help
--help-plugins list all available plugins with brief help
--help-options [<option prefix>] output matching command line options
--help-signals dump available control signals
--list-buffers output available inspection buffers
--list-builtin [<module prefix>] output matching builtin rules
--list-gids [<module prefix>] output matching generators
--list-modules [<module type>] list all known modules
--list-plugins list all known modules
--show-plugins list module and plugin versions

--help* and --list* options preempt other processing so should be last on the
command line since any following options are ignored.  To ensure options like
--markup and --plugin-path take effect, place them ahead of the help or list
options.

Options that filter output based on a matching prefix, such as --help-config
won't output anything if there is no match.  If no prefix is given, everything
matches.

Report bugs to bugs@snort.org.
```

3.6 Common Errors

FATAL: snort_config is required

- add this line near top of file:

```
require('snort_config')
```

PANIC: unprotected error in call to Lua API (cannot open snort_defaults.lua: No such file or directory)

- export SNORT_LUA_PATH to point to any dofiles

ERROR can't find xyz

- if xyz is the name of a module, make sure you are not assigning a scalar where a table is required (e.g. xyz = 2 should be xyz = { }).

ERROR can't find x.y

- module x does not have a parameter named y. check --help-module x for available parameters.

ERROR invalid x.y = z

- the value z is out of range for x.y. check --help-config x.y for the range allowed.

ERROR: x = { y = z } is in conf but is not being applied

- make sure that x = { } isn't set later because it will override the earlier setting. same for x.y.

FATAL: can't load lua/errors.lua: lua/errors.lua:68: = expected near ';'

- this is a syntax error reported by Lua to Snort on line 68 of errors.lua.

ERROR: rules(2) unknown rule keyword: find.

- this was due to not including the --script-path.

WARNING: unknown symbol x

- if you any variables, you can squelch such warnings by setting them in an environment variable SNORT_IGNORE. to ignore x, y, and z:

```
export SNORT_IGNORE="x y z"
```

3.7 Gotchas

- A nil key in a table will not be caught. Neither will a nil value in a table. Neither of the following will cause errors, nor will they actually set http_inspect.request_depth:

```
http_inspect = { request_depth }
http_inspect = { request_depth = undefined_symbol }
```

- It is not an error to set a value multiple times. The actual value applied may not be the last in the table either. It is best to avoid such cases.

```
http_inspect =
{
    request_depth = 1234,
    request_depth = 4321
}
```

- Snort can't tell you the exact filename or line number of a semantic error but it will tell you the fully qualified name.

3.8 Known Issues

- The dump DAQ will not work with multiple threads unless you use --daq-var output=none. This will be fixed at some point to use the Snort log directory, etc.
- If you build with hyperscan on OS X and see:

```
dyld: Library not loaded: @rpath/libhs.4.0.dylib

when you try to run src/snort, export DYLD_LIBRARY_PATH with the path to
libhs.  You can also do:

install_name_tool -change @rpath/libhs.4.0.dylib \
    /path-to/libhs.4.0.dylib src/snort
```

- Snort built with tcmalloc support (--enable-tcmalloc) on Ubuntu 17.04/18.04 crashes immediately.

```
Workaround:
Uninstall gperftools 2.5 provided by the distribution and install gperftools
2.7 before building Snort.
```

4 Usage

For the following examples "$my_path" is assumed to be the path to the Snort install directory. Additionally, it is assumed that "$my_path/bin" is in your PATH.

4.1 Environment

LUA_PATH is used directly by Lua to load and run required libraries. SNORT_LUA_PATH is used by Snort to load supplemental configuration files.

```
export LUA_PATH=$my_path/include/snort/lua/\?.lua\;\;
export SNORT_LUA_PATH=$my_path/etc/snort
```

4.2 Help

Print the help summary:

```
snort --help
```

Get help on a specific module ("stream", for example):

```
snort --help-module stream
```

Get help on the "-A" command line option:

```
snort --help-options A
```

Grep for help on threads:

```
snort --help-config | grep thread
```

Output help on "rule" options in AsciiDoc format:

```
snort --markup --help-options rule
```

Note
Snort stops reading command-line options after the "--help-**" and "--list-**" options, so any other options should be placed before them.

4.3 Sniffing and Logging

Read a pcap:

```
snort -r /path/to/my.pcap
```

Dump the packets to stdout:

```
snort -r /path/to/my.pcap -L dump
```

Dump packets with application data and layer 2 headers

```
snort -r /path/to/my.pcap -L dump -d -e
```

Note
Command line options must be specified separately. "snort -de" won't work. You can still concatenate options and their arguments, however, so "snort -Ldump" will work.

Dump packets from all pcaps in a directory:

```
snort --pcap-dir /path/to/pcap/dir --pcap-filter '*.pcap' -L dump -d -e
```

Log packets to a directory:

```
snort --pcap-dir /path/to/pcap/dir --pcap-filter '*.pcap' -L dump -l /path/to/log/ ↩
   dir
```

4.4 Configuration

Validate a configuration file:

```
snort -c $my_path/etc/snort/snort.lua
```

Validate a configuration file and a separate rules file:

```
snort -c $my_path/etc/snort/snort.lua -R $my_path/etc/snort/sample.rules
```

Read rules from stdin and validate:

```
snort -c $my_path/etc/snort/snort.lua --stdin-rules < $my_path/etc/snort/sample. ↩
   rules
```

Enable warnings for Lua configurations and make warnings fatal:

```
snort -c $my_path/etc/snort/snort.lua --warn-all --pedantic
```

Tell Snort where to look for additional Lua scripts:

```
snort --script-path /path/to/script/dir
```

4.5 IDS mode

Run Snort in IDS mode, reading packets from a pcap:

```
snort -c $my_path/etc/snort/snort.lua -r /path/to/my.pcap
```

Log any generated alerts to the console using the "-A" option:

```
snort -c $my_path/etc/snort/snort.lua -r /path/to/my.pcap -A alert_full
```

Capture separate stdout, stderr, and stdlog files (out has startup and shutdown output, err has warnings and errors, and log has alerts):

```
snort -c $my_path/etc/snort/snort.lua -r /path/to/my.pcap -A csv \
    1>out 2>err 3>log
```

Add or modify a configuration from the command line using the "--lua" option:

```
snort -c $my_path/etc/snort/snort.lua -r /path/to/my.pcap -A cmg \
    --lua 'ips = { enable_builtin_rules = true }'
```

Note
The "--lua" option can be specified multiple times.

Run Snort in IDS mode on an entire directory of pcaps, processing each input source on a separate thread:

```
snort -c $my_path/etc/snort/snort.lua --pcap-dir /path/to/pcap/dir \
    --pcap-filter '*.pcap' --max-packet-threads 8
```

Run Snort on 2 interfaces, eth0 and eth1:

```
snort -c $my_path/etc/snort/snort.lua -i "eth0 eth1" -z 2 -A cmg
```

Run Snort inline with the afpacket DAQ:

```
snort -c $my_path/etc/snort/snort.lua --daq afpacket -i "eth0:eth1" \
    -A cmg
```

4.6 Plugins

Load external plugins and use the "ex" alert:

```
snort -c $my_path/etc/snort/snort.lua \
    --plugin-path $my_path/lib/snort_extra \
    -A alert_ex -r /path/to/my.pcap
```

Test the LuaJIT rule option *find* loaded from stdin:

```
snort -c $my_path/etc/snort/snort.lua \
    --script-path $my_path/lib/snort_extra \
    --stdin-rules -A cmg -r /path/to/my.pcap << END
alert tcp any any -> any 80 (
    sid:3; msg:"found"; content:"GET";
    find:"pat='HTTP/1%.%d'" ; )
END
```

4.7 Output Files

To make it simple to configure outputs when you run with multiple packet threads, output files are not explicitly configured. Instead, you can use the options below to format the paths:

```
<logdir>/[<run_prefix>][<id#>][<X>]<name>
```

Log to unified in the current directory:

```
snort -c $my_path/etc/snort/snort.lua -r /path/to/my.pcap -A unified2
```

Log to unified in the current directory with a different prefix:

```
snort -c $my_path/etc/snort/snort.lua -r /path/to/my.pcap -A unified2 \
    --run-prefix take2
```

Log to unified in /tmp:

```
snort -c $my_path/etc/snort/snort.lua -r /path/to/my.pcap -A unified2 -l /tmp
```

Run 4 packet threads and log with thread number prefix (0-3):

```
snort -c $my_path/etc/snort/snort.lua --pcap-dir /path/to/pcap/dir \
    --pcap-filter '*.pcap' -z 4 -A unified2
```

Run 4 packet threads and log in thread number subdirs (0-3):

```
snort -c $my_path/etc/snort/snort.lua --pcap-dir /path/to/pcap/dir \
    --pcap-filter '*.pcap' -z 4 -A unified2 --id-subdir
```

Note
subdirectories are created automatically if required. Log filename is based on module name that writes the file. All text mode outputs default to stdout. These options can be combined.

4.8 DAQ Alternatives

Process hext packets from stdin:

```
snort -c $my_path/etc/snort/snort.lua \
    --daq-dir $my_path/lib/snort/daqs --daq hext -i tty << END
$packet 10.1.2.3 48620 -> 10.9.8.7 80
"GET / HTTP/1.1\r\n"
"Host: localhost\r\n"
"\r\n"
END
```

Process raw ethernet from hext file:

```
snort -c $my_path/etc/snort/snort.lua \
    --daq-dir $my_path/lib/snort/daqs --daq hext \
    --daq-var dlt=1 -r <hext-file>
```

Process a directory of plain files (ie non-pcap) with 4 threads with 8K buffers:

```
snort -c $my_path/etc/snort/snort.lua \
    --daq-dir $my_path/lib/snort/daqs --daq file \
    --pcap-dir path/to/files -z 4 -s 8192
```

Bridge two TCP connections on port 8000 and inspect the traffic:

```
snort -c $my_path/etc/snort/snort.lua \
    --daq-dir $my_path/lib/snort/daqs --daq socket
```

4.9 Logger Alternatives

Dump TCP stream payload in hext mode:

```
snort -c $my_path/etc/snort/snort.lua -L hext
```

Output timestamp, pkt_num, proto, pkt_gen, dgm_len, dir, src_ap, dst_ap, rule, action for each alert:

```
snort -c $my_path/etc/snort/snort.lua -A csv
```

Output the old test format alerts:

```
snort -c $my_path/etc/snort/snort.lua \
    --lua "alert_csv = { fields = 'pkt_num gid sid rev', separator = '\t' }"
```

4.10 Shell

You must build with --enable-shell to make the command line shell available.

Enable shell mode:

```
snort --shell <args>
```

You will see the shell mode command prompt, which looks like this:

```
o")~
```

(The prompt can be changed with the SNORT_PROMPT environment variable.)

You can pause immediately after loading the configuration and again before exiting with:

```
snort --shell --pause <args>
```

In that case you must issue the resume() command to continue. Enter quit() to terminate Snort or detach() to exit the shell. You can list the available commands with help().

To enable local telnet access on port 12345:

```
snort --shell -j 12345 <args>
```

The command line interface is still under development. Suggestions are welcome.

4.11 Signals

Note
The following examples assume that Snort is currently running and has a process ID of <pid>.

Modify and Reload Configuration:

```
echo 'suppress = { { gid = 1, sid = 2215 } }' >> $my_path/etc/snort/snort.lua
kill -hup <pid>
```

Dump stats to stdout:

```
kill -usr1 <pid>
```

Shutdown normally:

```
kill -term <pid>
```

Exit without flushing packets:

```
kill -quit <pid>
```

List available signals:

```
snort --help-signals
```

Note
The available signals may vary from platform to platform.

5 Features

This section explains how to use key features of Snort.

5.1 Active Response

Snort can take more active role in securing network by sending active responses to shutdown offending sessions. When active responses is enabled, snort will send TCP RST or ICMP unreachable when dropping a session.

5.1.1 Changes from Snort 2.9

- stream5_global:max_active_responses and min_response_seconds are now active.max_responses and active.min_interval.
- Response actions were removed from IPS rule body to the rule action in the header. This includes react, reject, and rewrite (split out of replace which now just does the detection part). These IPS actions are plugins.
- drop and block are synonymous in Snort 2.9 but in Snort 3.0 drop means don't forward the current packet only whereas block means don't forward this or any following packet on the flow.

5.1.2 Configure Active

Active response is enabled by configuring one of following IPS action plugins:

```
react = { }
reject = { }
rewrite = { }
```

Active responses will be performed for reject, react or rewrite IPS rule actions, and response packets are encoded based on the triggering packet. TTL will be set to the value captured at session pickup.

Configure the number of attempts to land a TCP RST within the session's current window (so that it is accepted by the receiving TCP). This sequence "strafing" is really only useful in passive mode. In inline mode the reset is put straight into the stream in lieu of the triggering packet so strafing is not necessary.

Each attempt (sent in rapid succession) has a different sequence number. Each active response will actually cause this number of TCP resets to be sent. TCP data is multiplied similarly. At most 1 ICMP unreachable is sent, iff attempts > 0.

Device IP will perform network layer injection. It is probably a better choice to specify an interface and avoid kernel routing tables, etc.

dst_mac will change response destination MAC address, if the device is eth0, eth1, eth2 etc. Otherwise, response destination MAC address is derived from packet.

Example:

```
active =
{
    attempts = 2,
    device = "eth0",
    dst_mac = "00:06:76:DD:5F:E3",
}
```

5.1.3 Reject

IPS action reject perform active response to shutdown hostile network session by injecting TCP resets (TCP connections) or ICMP unreachable packets.

Example:

```
reject = { reset = "both", control = "all" }

local_rules =
[[
reject tcp ( msg:"hostile connection"; flow:established, to_server;
content:"HACK!"; sid:1; )
]]

ips =
{
    rules = local_rules,
}
```

5.1.4 React

IPS action react enables sending an HTML page on a session and then resetting it.

The page to be sent can be read from a file:

```
react = { page = "custmized_block_page.html", }
```

or else the default is used:

```
<default_page> ::= \
    "HTTP/1.1 403 Forbidden\r\n"
    "Connection: close\r\n"
    "Content-Type: text/html; charset=utf-8\r\n"
    "\r\n"
    "<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.1//EN\"\r\n" \
    "    \"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd\">\r\n" \
    "<html xmlns=\"http://www.w3.org/1999/xhtml\"
    xml:lang=\"en\">\r\n" \
    "<head>\r\n" \
```

```
    "<meta http-equiv=\"Content-Type\" content=\"text/html;
    charset=UTF-8\" />\r\n" \
    "<title>Access Denied</title>\r\n" \
    "</head>\r\n" \
    "<body>\r\n" \
    "<h1>Access Denied</h1>\r\n" \
    "<p>%s</p>\r\n" \
    "</body>\r\n" \
    "</html>\r\n";
```

Note that the file must contain the entire response, including any HTTP headers. In fact, the response isn't strictly limited to HTTP. You could craft a binary payload of arbitrary content.

When the rule is configured, the page is loaded and the %s is replaced with the selected message, which defaults to:

```
"You are attempting to access a forbidden site.<br />" \
"Consult your system administrator for details."
```

Additional formatting operators beyond a single %s are prohibited, including %d, %x, %s, as well as any URL encodings such as as %20 (space) that may be within a reference URL.

Example:

```
react = { page = "my_block_page.html" }

local_rules =
[[
react http ( msg:"Unauthorized Access Prohibited!"; flow:established,
to_server; http_method; content:"GET"; sid:1; )
]]

ips =
{
    rules = local_rules,
}
```

5.1.5 Rewrite

IPS action rewrite enables overwrite packet contents based on "replace" option in the rules.

For example:

```
rewrite = { }
local_rules =
[[
rewrite tcp 10.1.1.87 any -> 10.1.1.0/24 80
(
    sid:1000002;
    msg:"test replace rule";
    content:"index.php", nocase;
    replace:"indax.php";
)
]]

ips =
{
    rules = local_rules,
}
```

this rule replaces "index.php" with "indax.php", and rewrite action updates that packet.

to enable rewrite action:

```
rewrite = { }
```

the replace operation can be disabled by changing the configuration:

```
rewrite = { disable_replace = true }
```

5.2 AppId

Network administrators need application awareness in order to fine tune their management of the ever-growing number of applications passing traffic over the network. Application awareness allows an administrator to create rules for applications as needed by the business. The rules can be used to take action based on the application, such as block, allow or alert.

5.2.1 Overview

The AppId inspector provides an application level view when managing networks by providing the following features:

- Network control: The inspector works with Snort rules by providing a set of application identifiers (AppIds) to Snort rule writers.
- Application usage awareness: The inspector outputs statistics to show how many times applications are being used on the network.
- Custom applications: Administrators can create their own application detectors to detect new applications. The detectors are written in Lua and interface with Snort using a well-defined C-Lua API.
- Open Detector Package (ODP): A set of pre-defined application detectors are provided by the Snort team and can be downloaded from snort.org.

5.2.2 Dependency Requirements

For proper functioning of the AppId inspector, at a minimum stream flow tracking must be enabled. In addition, to identify TCP-based or UDP-based applications then the appropriate stream inspector must be enabled, e.g. stream_tcp or stream_udp.

In addition, in order to identify HTTP-based applications, the HTTP inspector must be enabled. Otherwise, only non-HTTP applications will be identified.

AppId subscribes to the inspection events published by other inspectors, such as the HTTP and SSL inspectors, to gain access to the data needed. It uses that data to help determine the application ID.

5.2.3 Configuration

The AppId feature can be enabled via configuration. To enable it with the default settings use:

```
appid = { }
```

To use an AppId as a matching parameter in an IPS rule, use the *appids* keyword. For example, to block HTTP traffic that contains a specific header:

```
block tcp any any -> 192.168.0.1 any ( msg:"Block Malicious HTTP header";
  appids:"HTTP"; content:"X-Header: malicious"; sid:18000; )
```

Alternatively, the HTTP application can be specified in place of *tcp* instead of using the *appids* keyword. The AppId inspector will set the service when it is discovered so it can be used in IPS rules like this. Note that this rule also does not specify the IPs or ports which default to *any*.

```
block http ( msg:"Block Malicious HTTP header";
  content:"X-Header: malicious"; sid:18000; )
```

It's possible to specify multiple applications (as many as desired) with the appids keyword. A rule is considered a match if any of the applications on the rule match. Note that this rule does not match specific content which will reduce performance.

```
alert tcp any any -> 192.168.0.1 any ( msg:"Alert ";
  appids:"telnet,ssh,smtp,http";
```

Below is a minimal Snort configuration that is sufficient to block flows based on a specific HTTP header:

```
require("snort_config")

dir = os.getenv('SNORT_LUA_PATH')

if ( not dir ) then
    dir = '.'
end

dofile(dir .. '/snort_defaults.lua')

local_rules =
[[
block http ( msg:"openAppId: test content match for app http";
content:"X-Header: malicious"; sid:18760; rev:4; )
]]

stream = { }

stream_tcp = { }

binder =
{
    {
        when =
        {
            proto = 'tcp',
            ports = [[ 80 8080 ]],
        },
        use =
        {
            type = 'http_inspect',
        },
    },
}

http_inspect = { }

appid = { }

ips =
{
    rules = local_rules,
}
```

5.2.4 Session Application Identifiers

There are up to four AppIds stored in a session as defined below:

- serviceAppId - An appId associated with server side of a session. Example: http server.
- clientAppId - An appId associated with application on client side of a session. Example: Firefox.
- payloadAppId - For services like http this appId is associated with a webserver host. Example: Facebook.
- miscAppId - For some encapsulated protocols, this is the highest encapsulated application.

For packets originating from the client, a payloadAppid in a session is matched with all AppIds listed on a rule. Thereafter miscAppId, clientAppId and serviceAppId are matched. Since Alert Events contain one AppId, only the first match is reported. If a rule without an appids option matches, then the most specific appId (in order of payload, misc, client, server) is reported.

The same logic is followed for packets originating from the server with one exception. The order of matching is changed to make serviceAppId come before clientAppId.

5.2.5 AppId Usage Statistics

The AppId inspector prints application network usage periodically in the snort log directory in unified2 format. File name, time interval for statistic and file rollover are controlled by appId inspection configuration.

5.2.6 Open Detector Package (ODP) Installation

Application detectors from Snort team will be delivered in a separate package called the Open Detector Package (ODP) that can be downloaded from snort.org. ODP is a package that contains the following artifacts:

- Application detectors in the Lua language.
- Port detectors, which are port only application detectors, in meta-data in YAML format.
- appMapping.data file containing application metadata. This file should not be modified. The first column contains application identifier and second column contains application name. Other columns contain internal information.
- Lua library files DetectorCommon.lua, flowTrackerModule.lua and hostServiceTrackerModule.lua

A user can install the ODP package in any directory and configure this directory via the app_detector_dir option in the appid preprocessor configuration. Installing ODP will not modify any subdirectory named custom, where user-created detectors are located.

When installed, ODP will create following sub-directories:

- odp/port //Cisco port-only detectors
- odp/lua //Cisco Lua detectors
- odp/libs //Cisco Lua modules

5.2.7 User Created Application Detectors

Users can detect new applications by adding detectors in the Lua language. A document will be posted on the Snort Website with details on API. Users can also copy over Snort team provided detectors and modify them. Users can also use the detector creation tool described in the next section.

Users must organize their Lua detectors and libraries by creating the following directory structure, under the ODP installation directory.

- custom/port //port-only detectors
- custom/lua //Lua detectors
- custom/libs //Lua modules

The root path is specified by the "app_detector_dir" parameter of the appid section of snort.conf:

```
appid  =
{
    app_detector_dir = '/usr/local/lib/openappid',
}
```

So the path to the user-created lua files would be /usr/local/lib/openappid/custom/lua/

None of the directories below /usr/local/lib/openappid/ would be added for you.

5.2.8 Application Detector Creation Tool

For rudimentary Lua detectors, there is a tool provided called appid_detector_builder.sh. This is a simple, menu-driven bash script which creates .lua files in your current directory, based on your choices and on patterns you supply.

When you launch the script, it will prompt for the Application Id that you are giving for your detector. This is free-form ASCII with minor restrictions. The Lua detector file will be named based on your Application Id. If the file name already exists you will be prompted to overwrite it.

You will also be prompted for a description of your detector to be placed in the comments of the Lua source code. This is optional.

You will then be asked a series of questions designed to construct Lua code based on the kind of pattern data, protocol, port(s), etc.

When complete, the Protocol menu will be changed to include the option, "Save Detector". Instead of saving the file and exiting the script, you are allowed to give additional criteria for another pattern which may also be incorporated in the detection scheme. Then either pattern, when matched, will be considered a valid detection.

For example, your first choices might create an HTTP detection pattern of "example.com", and the next set of choices would add the HTTP detection pattern of "example.uk.co" (an equally fictional British counterpart). They would then co-exist in the Lua detector, and either would cause a detection with the name you give for your Application Id.

The resulting .lua file will need to be placed in the directory, "custom/lua", described in the previous section of the README above called "User Created Application Detectors"

5.3 Binder

One of the fundamental differences between Snort 2 and Snort 3 concerns configuration related to networks and ports. Here is a brief review of Snort 2 configuration for network and service related components:

- Snort's configuration has a default policy and optional policies selected by VLAN or network (with config binding).
- Each policy contains a user defined set of preprocessor configurations.
- Each preprocessor has a default configuration and some support non-default configurations selected by network.
- Most preprocessors have port configurations.
- The default policy may also contain a list of ports to ignore.

In Snort 3, the above configurations are done in a single module called the binder. Here is an example:

```
binder =
{
    -- allow all tcp port 22:
    -- (similar to Snort 2 config ignore_ports)
    { when = { proto = 'tcp', ports = '22' }, use = { action = 'allow' } },

-- select a config file by vlan
-- (similar to Snort 2 config binding by vlan)
{ when = { vlans = '1024' }, use = { file = 'vlan.lua' } },

-- use a non-default HTTP inspector for port 8080:
-- (similar to a Snort 2 targeted preprocessor config)
{ when = { nets = '192.168.0.0/16', proto = 'tcp', ports = '8080' },
  use = { name = 'alt_http', type = 'http_inspect' } },

-- use the default inspectors:
-- (similar to a Snort 2 default preprocessor config)
{ when = { proto = 'tcp' }, use = { type = 'stream_tcp' } },
{ when = { service = 'http' }, use = { type = 'http_inspect' } },

    -- figure out which inspector to run automatically:
    { use = { type = 'wizard' } }
}
```

Bindings are evaluated when a session starts and again if and when service is identified on the session. Essentially, the bindings are a list of when-use rules evaluated from top to bottom. The first matching network and service configurations are applied. binder.when can contain any combination of criteria and binder.use can specify an action, config file, or inspector configuration.

5.4 Byte rule options

5.4.1 byte_test

This rule option tests a byte field against a specific value (with operator). Capable of testing binary values or converting representative byte strings to their binary equivalent and testing them.

Snort uses the C operators for each of these operators. If the & operator is used, then it would be the same as using

```
if (data & value) { do_something(); }
```

! operator negates the results from the base check. *!<oper>* is considered as

```
!(data <oper> value)
```

Note: The bitmask option applies bitwise AND operator on the bytes converted. The result will be right-shifted by the number of bits equal to the number of trailing zeros in the mask. This applies for the other rule options as well.

Examples

```
alert tcp (byte_test:2, =, 568, 0, bitmask 0x3FF0;)
```

This example extracts 2 bytes at offset 0, performs bitwise and with bitmask 0x3FF0, shifts the result by 4 bits and compares to 568.

```
alert udp (byte_test:4, =, 1234, 0, string, dec;
    msg:"got 1234!";)
```

```
alert udp (byte_test:8, =, 0xdeadbeef, 0, string, hex;
    msg:"got DEADBEEF!";)
```

5.4.2 byte_jump

The byte_jump rule option allows rules to be written for length encoded protocols trivially. By having an option that reads the length of a portion of data, then skips that far forward in the packet, rules can be written that skip over specific portions of length-encoded protocols and perform detection in very specific locations.

Examples

```
alert tcp (content:"Begin";
    byte_jump:0, 0, from_end, post_offset -6;
    content:"end..", distance 0, within 5;
    msg:"Content match from end of the payload";)

alert tcp (content:"catalog";
    byte_jump:2, 1, relative, post_offset 2, bitmask 0x03f0;
    byte_test:2, =, 968, 0, relative;
    msg:"Bitmask applied on the 2 bytes extracted for byte_jump";)
```

5.4.3 byte_extract

The byte_extract keyword is another useful option for writing rules against length-encoded protocols. It reads in some number of bytes from the packet payload and saves it to a variable. These variables can be referenced later in the rule, instead of using hard-coded values.

Other options which use byte_extract variables

A byte_extract rule option detects nothing by itself. Its use is in extracting packet data for use in other rule options.

Here is a list of places where byte_extract variables can be used:

- content/uricontent: offset, depth, distance, within
- byte_test: offset, value
- byte_jump: offset, post_offset
- isdataat: offset

Examples

```
alert tcp (byte_extract:1, 0, str_offset;
    byte_extract:1, 1, str_depth;
    content:"bad stuff", offset str_offset, depth str_depth;
    msg:"Bad Stuff detected within field";)

alert tcp (content:"START"; byte_extract:1, 0, myvar, relative;
    byte_jump:1, 3, relative, post_offset myvar;
    content:"END", distance 6, within 3;
    msg: "byte_jump - pass variable to post_offset";)
```

This example uses two variables.

The first variable keeps the offset of a string, read from a byte at offset 0. The second variable keeps the depth of a string, read from a byte at offset 1. These values are used to constrain a pattern match to a smaller area.

```
alert tcp (content:"|04 63 34 35|", offset 4, depth 4;
    byte_extract: 2, 0, var_match, relative, bitmask 0x03ff;
    byte_test: 2, =, var_match, 2, relative;
    msg:"Test value match, after applying bitmask on bytes extracted";)
```

5.4.4 byte_math

Perform a mathematical operation on an extracted value and a specified value or existing variable, and store the outcome in a new resulting variable. These resulting variables can be referenced later in the rule, at the same places as byte_extract variables.

The syntax for this rule option is different. The order of the options is critical for the other rule options and can't be changed. For example, the first option is the number of bytes to extract. Here the name of the option is explicitly written, for example : bytes 2. The order is not important.

Note
Byte_math operations are performed on unsigned 32-bit values. When writing a rule it should be taken into consideration to avoid wrap around.

Examples

```
alert tcp ( byte_math: bytes 2, offset 0, oper *, rvalue 10, result area;
   byte_test:2,>,area,16;)
```

At the zero offset of the payload, extract 2 bytes and apply multiplication operation with value 10. Store result in variable area. The area variable is given as input to byte_test value option.

Let's consider 2 bytes of extracted data is 5. The rvalue is 10. Result variable area is 50 (5 * 10). Area variable can be used in either byte_test offset/value options.

5.4.5 Testing Numerical Values

The rule options byte_test and byte_jump were written to support writing rules for protocols that have length encoded data. RPC was the protocol that spawned the requirement for these two rule options, as RPC uses simple length based encoding for passing data.

In order to understand why byte test and byte jump are useful, let's go through an exploit attempt against the sadmind service.

This is the payload of the exploit:

```
89 09 9c e2 00 00 00 00 00 00 00 02 00 01 87 88  ................
00 00 00 0a 00 00 00 01 00 00 00 01 00 00 00 20  ................
40 28 3a 10 00 00 00 0a 4d 45 54 41 53 50 4c 4f  @(:.....metasplo
49 54 00 00 00 00 00 00 00 00 00 00 00 00 00 00  it..............
00 00 00 00 00 00 00 00 40 28 3a 14 00 07 45 df  ........@(:...e.
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 06 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 04 00 00 00 00 00 00 00 04  ................
7f 00 00 01 00 01 87 88 00 00 00 0a 00 00 00 04  ................
7f 00 00 01 00 01 87 88 00 00 00 0a 00 00 00 11  ................
00 00 00 1e 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 3b 4d 45 54 41 53 50 4c 4f  .......;metasplo
```

```
49 54 00 00 00 00 00 00 00 00 00 00 00 00 00 00  it..............
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 06 73 79 73 74 65 6d 00 00  ........system..
00 00 00 15 2e 2e 2f 2e 2e 2f 2e 2e 2f 2e 2e 2f  ....../../../../
2e 2e 2f 62 69 6e 2f 73 68 00 00 00 00 00 04 1e  ../bin/sh.......
```

Let's break this up, describe each of the fields, and figure out how to write a rule to catch this exploit.

There are a few things to note with RPC:

Numbers are written as uint32s, taking four bytes. The number 26 would show up as 0x0000001a.

Strings are written as a uint32 specifying the length of the string, the string, and then null bytes to pad the length of the string to end on a 4-byte boundary. The string *bob* would show up as 0x00000003626f6200.

```
89 09 9c e2     - the request id, a random uint32, unique to each request
00 00 00 00     - rpc type (call = 0, response = 1)
00 00 00 02     - rpc version (2)
00 01 87 88     - rpc program (0x00018788 = 100232 = sadmind)
00 00 00 0a     - rpc program version (0x0000000a = 10)
00 00 00 01     - rpc procedure (0x00000001 = 1)
00 00 00 01     - credential flavor (1 = auth_unix)
00 00 00 20     - length of auth_unix data (0x20 = 32)

## the next 32 bytes are the auth_unix data
40 28 3a 10 - unix timestamp (0x40283a10 = 1076378128 = feb 10 01:55:28 2004 gmt)
00 00 00 0a - length of the client machine name (0x0a = 10)
4d 45 54 41 53 50 4c 4f 49 54 00 00  - metasploit

00 00 00 00 - uid of requesting user (0)
00 00 00 00 - gid of requesting user (0)
00 00 00 00 - extra group ids (0)

00 00 00 00 - verifier flavor (0 = auth_null, aka none)
00 00 00 00 - length of verifier (0, aka none)
```

The rest of the packet is the request that gets passed to procedure 1 of sadmind.

However, we know the vulnerability is that sadmind trusts the uid coming from the client. sadmind runs any request where the client's uid is 0 as root. As such, we have decoded enough of the request to write our rule.

First, we need to make sure that our packet is an RPC call.

```
content:"|00 00 00 00|", offset 4, depth 4;
```

Then, we need to make sure that our packet is a call to sadmind.

```
content:"|00 01 87 88|", offset 12, depth 4;
```

Then, we need to make sure that our packet is a call to the procedure 1, the vulnerable procedure.

```
content:"|00 00 00 01|", offset 20, depth 4;
```

Then, we need to make sure that our packet has auth_unix credentials.

```
content:"|00 00 00 01|", offset 24, depth 4;
```

We don't care about the hostname, but we want to skip over it and check a number value after the hostname. This is where byte_test is useful. Starting at the length of the hostname, the data we have is:

```
00 00 00 0a 4d 45 54 41 53 50 4c 4f 49 54 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00
```

We want to read 4 bytes, turn it into a number, and jump that many bytes forward, making sure to account for the padding that RPC requires on strings. If we do that, we are now at:

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00
```

which happens to be the exact location of the uid, the value we want to check.

In English, we want to read 4 bytes, 36 bytes from the beginning of the packet, and turn those 4 bytes into an integer and jump that many bytes forward, aligning on the 4-byte boundary. To do that in a Snort rule, we use:

```
byte_jump:4,36,align;
```

then we want to look for the uid of 0.

```
content:"|00 00 00 00|", within 4;
```

Now that we have all the detection capabilities for our rule, let's put them all together.

```
content:"|00 00 00 00|", offset 4, depth 4;
content:"|00 01 87 88|", offset 12, depth 4;
content:"|00 00 00 01|", offset 20, depth 4;
content:"|00 00 00 01|", offset 24, depth 4;
byte_jump:4,36,align;
content:"|00 00 00 00|", within 4;
```

The 3rd and fourth string match are right next to each other, so we should combine those patterns. We end up with:

```
content:"|00 00 00 00|", offset 4, depth 4;
content:"|00 01 87 88|", offset 12, depth 4;
content:"|00 00 00 01 00 00 00 01|", offset 20, depth 8;
byte_jump:4,36,align;
content:"|00 00 00 00|", within 4;
```

If the sadmind service was vulnerable to a buffer overflow when reading the client's hostname, instead of reading the length of the hostname and jumping that many bytes forward, we would check the length of the hostname to make sure it is not too large.

To do that, we would read 4 bytes, starting 36 bytes into the packet, turn it into a number, and then make sure it is not too large (let's say bigger than 200 bytes). In Snort, we do:

```
byte_test:4,>,200,36;
```

Our full rule would be:

```
content:"|00 00 00 00|", offset 4, depth 4;
content:"|00 01 87 88|", offset 12, depth 4;
content:"|00 00 00 01 00 00 00 01|", offset 20, depth 8;
byte_test:4,>,200,36;
```

5.5 DCE Inspectors

The main purpose of these inspector are to perform SMB desegmentation and DCE/RPC defragmentation to avoid rule evasion using these techniques.

5.5.1 Overview

The following transports are supported for DCE/RPC: SMB, TCP, and UDP. New rule options have been implemented to improve performance, reduce false positives and reduce the count and complexity of DCE/RPC based rules.

Different from Snort 2, the DCE-RPC preprocessor is split into three inspectors - one for each transport: dce_smb, dce_tcp, dce_udp. This includes the configuration as well as the inspector modules. The Snort 2 server configuration is now split between the inspectors. Options that are meaningful to all inspectors, such as policy and defragmentation, are copied into each inspector configuration. The address/port mapping is handled by the binder. Autodetect functionality is replaced by wizard curses.

5.5.2 Quick Guide

A typical dcerpce configuration looks like this:

```
binder =
{
   {
       when =
       {
           proto = 'tcp',
           ports = '139 445 1025',
        },
       use =
       {
           type = 'dce_smb',
       },
    },
    {
       when =
       {
           proto = 'tcp',
           ports = '135 2103',
       },
       use =
       {
           type = 'dce_tcp',
       },
    },
    {
       when =
       {
           proto = 'udp',
           ports = '1030',
       },
       use =
       {
           type = 'dce_udp',
       },
    }
 }

dce_smb = { }

dce_tcp = { }

dce_udp = { }
```

In this example, it defines smb, tcp and udp inspectors based on port. All the configurations are default.

5.5.3 Target Based

There are enough important differences between Windows and Samba versions that a target based approach has been implemented. Some important differences:

- Named pipe instance tracking
- Accepted SMB commands
- AndX command chaining
- Transaction tracking
- Multiple Bind requests
- DCE/RPC Fragmented requests - Context ID
- DCE/RPC Fragmented requests - Operation number
- DCE/RPC Stub data byte order

Because of those differences, each inspector can be configured to different policy. Here are the list of policies supported:

- WinXP (default)
- Win2000
- WinVista
- Win2003
- Win2008
- Win7
- Samba
- Samba-3.0.37
- Samba-3.0.22
- Samba-3.0.20

5.5.4 Reassembling

Both SMB inspector and TCP inspector support reassemble. Reassemble threshold specifies a minimum number of bytes in the DCE/RPC desegmentation and defragmentation buffers before creating a reassembly packet to send to the detection engine. This option is useful in inline mode so as to potentially catch an exploit early before full defragmentation is done. A value of 0 s supplied as an argument to this option will, in effect, disable this option. Default is disabled.

5.5.5 SMB

SMB inspector is one of the most complex inspectors. In addition to supporting rule options and lots of inspector rule events, it also supports file processing for both SMB version 1, 2, and 3.

Finger Print Policy

In the initial phase of an SMB session, the client needs to authenticate with a SessionSetupAndX. Both the request and response to this command contain OS and version information that can allow the inspector to dynamically set the policy for a session which allows for better protection against Windows and Samba specific evasions.

File Inspection

SMB inspector supports file inspection. A typical configuration looks like this:

```
binder =
{
   {
       when =
       {
           proto = 'tcp',
           ports = '139 445',
       },
       use =
       {
           type = 'dce_smb',
       },
   },
}

dce_smb =
{
    smb_file_inspection = 'on',
    smb_file_depth = 0,
 }

file_id =
{
    enable_type = true,
    enable_signature = true,
    enable_capture = true,
    file_rules = magics,
}
```

First, define a binder to map tcp port 139 and 445 to smb. Then, enable file inspection in smb inspection and set the file depth as unlimited. Lastly, enable file inspector to inspect file type, calculate file signature, and capture file. The details of file inspector are explained in file processing section.

SMB inspector does inspection of normal SMB file transfers. This includes doing file type and signature through the file processing as well as setting a pointer for the "file_data" rule option. Note that the "file_depth" option only applies to the maximum amount of file data for which it will set the pointer for the "file_data" rule option. For file type and signature it will use the value configured for the file API. If "only" is specified, the inspector will only do SMB file inspection, i.e. it will not do any DCE/RPC tracking or inspection. If "on" is specified with no arguments, the default file depth is 16384 bytes. An argument of -1 to "file-depth" disables setting the pointer for "file_data", effectively disabling SMB file inspection in rules. An argument of 0 to "file_depth" means unlimited. Default is "off", i.e. no SMB file inspection is done in the inspector.

5.5.6 TCP

dce_tcp inspector supports defragmentation, reassembling, and policy that is similar to SMB.

5.5.7 UDP

dce_udp is a very simple inspector that only supports defragmentation

5.5.8 Rule Options

New rule options are supported by enabling the dcerpc2 inspectors:

- dce_iface
- dce_opnum
- dce_stub_data

New modifiers to existing byte_test and byte_jump rule options:

- byte_test: dce
- byte_jump: dce

dce_iface

For DCE/RPC based rules it has been necessary to set flow-bits based on a client bind to a service to avoid false positives. It is necessary for a client to bind to a service before being able to make a call to it. When a client sends a bind request to the server, it can, however, specify one or more service interfaces to bind to. Each interface is represented by a UUID. Each interface UUID is paired with a unique index (or context id) that future requests can use to reference the service that the client is making a call to. The server will respond with the interface UUIDs it accepts as valid and will allow the client to make requests to those services. When a client makes a request, it will specify the context id so the server knows what service the client is making a request to. Instead of using flow-bits, a rule can simply ask the inspector, using this rule option, whether or not the client has bound to a specific interface UUID and whether or not this client request is making a request to it. This can eliminate false positives where more than one service is bound to successfully since the inspector can correlate the bind UUID to the context id used in the request. A DCE/RPC request can specify whether numbers are represented as big endian or little endian. The representation of the interface UUID is different depending on the endianness specified in the DCE/RPC previously requiring two rules - one for big endian and one for little endian. The inspector eliminates the need for two rules by normalizing the UUID. An interface contains a version. Some versions of an interface may not be vulnerable to a certain exploit. Also, a DCE/RPC request can be broken up into 1 or more fragments. Flags (and a field in the connectionless header) are set in the DCE/RPC header to indicate whether the fragment is the first, a middle or the last fragment. Many checks for data in the DCE/RPC request are only relevant if the DCE/RPC request is a first fragment (or full request), since subsequent fragments will contain data deeper into the DCE/RPC request. A rule which is looking for data, say 5 bytes into the request (maybe it's a length field), will be looking at the wrong data on a fragment other than the first, since the beginning of subsequent fragments are already offset some length from the beginning of the request. This can be a source of false positives in fragmented DCE/RPC traffic. By default it is reasonable to only evaluate if the request is a first fragment (or full request). However, if the "any_frag" option is used to specify evaluating on all fragments.

Examples:

```
dce_iface: 4b324fc8-1670-01d3-1278-5a47bf6ee188;
dce_iface: 4b324fc8-1670-01d3-1278-5a47bf6ee188,<2;
dce_iface: 4b324fc8-1670-01d3-1278-5a47bf6ee188,any_frag;
dce_iface: 4b324fc8-1670-01d3-1278-5a47bf6ee188,=1,any_frag;
```

This option is used to specify an interface UUID. Optional arguments are an interface version and operator to specify that the version be less than (<), greater than (>), equal to (=) or not equal to (!) the version specified. Also, by default the rule will only be evaluated for a first fragment (or full request, i.e. not a fragment) since most rules are written to start at the beginning of a request. The "any_frag" argument says to evaluate for middle and last fragments as well. This option requires tracking client Bind and Alter Context requests as well as server Bind Ack and Alter Context responses for connection-oriented DCE/RPC in the inspector. For each Bind and Alter Context request, the client specifies a list of interface UUIDs along with a handle (or context id) for each interface UUID that will be used during the DCE/RPC session to reference the interface. The server response indicates which interfaces it will allow the client to make requests to - it either accepts or rejects the client's wish to bind to a certain interface. This tracking is required so that when a request is processed, the context id used in the request can be correlated with the interface UUID it is a handle for.

hexlong and hexshort will be specified and interpreted to be in big endian order (this is usually the default way an interface UUID will be seen and represented). As an example, the following Messenger interface UUID as taken off the wire from a little endian Bind request:

```
|f8 91 7b 5a 00 ff d0 11 a9 b2 00 c0 4f b6 e6 fc|
```

must be written as:

```
5a7b91f8-ff00-11d0-a9b2-00c04fb6e6fc
```

The same UUID taken off the wire from a big endian Bind request:

```
|5a 7b 91 f8 ff 00 11 d0 a9 b2 00 c0 4f b6 e6 fc|
```

must be written the same way:

```
5a7b91f8-ff00-11d0-a9b2-00c04fb6e6fc
```

This option matches if the specified interface UUID matches the interface UUID (as referred to by the context id) of the DCE/RPC request and if supplied, the version operation is true. This option will not match if the fragment is not a first fragment (or full request) unless the "any_frag" option is supplied in which case only the interface UUID and version need match. Note that a defragmented DCE/RPC request will be considered a full request.

Using this rule option will automatically insert fast pattern contents into the fast pattern matcher. For UDP rules, the interface UUID, in both big and little endian format will be inserted into the fast pattern matcher. For TCP rules, (1) if the rule option "flow:to_server|from_client" is used, |05 00 00| will be inserted into the fast pattern matcher, (2) if the rule option "flow:from_server|to_client" is used, |05 00 02| will be inserted into the fast pattern matcher and (3) if the flow isn't known, |05 00| will be inserted into the fast pattern matcher. Note that if the rule already has content rule options in it, the best (meaning longest) pattern will be used. If a content in the rule uses the fast_pattern rule option, it will unequivocally be used over the above mentioned patterns.

dce_opnum

The opnum represents a specific function call to an interface. After is has been determined that a client has bound to a specific interface and is making a request to it (see above - dce_iface) usually we want to know what function call it is making to that service. It is likely that an exploit lies in the particular DCE/RPC function call.

Examples:

```
dce_opnum: 15;
dce_opnum: 15-18;
dce_opnum: 15,18-20;
dce_opnum: 15,17,20-22;
```

This option is used to specify an opnum (or operation number), opnum range or list containing either or both opnum and/or opnum-range. The opnum of a DCE/RPC request will be matched against the opnums specified with this option. This option matches if any one of the opnums specified match the opnum of the DCE/RPC request.

dce_stub_data

Since most DCE/RPC based rules had to do protocol decoding only to get to the DCE/RPC stub data, i.e. the remote procedure call or function call data, this option will alleviate this need and place the cursor at the beginning of the DCE/RPC stub data. This reduces the number of rule option checks and the complexity of the rule.

This option takes no arguments.

Example:

```
dce_stub_data;
```

This option is used to place the cursor (used to walk the packet payload in rules processing) at the beginning of the DCE/RPC stub data, regardless of preceding rule options. There are no arguments to this option. This option matches if there is DCE/RPC stub data.

The cursor is moved to the beginning of the stub data. All ensuing rule options will be considered "sticky" to this buffer. The first rule option following dce_stub_data should use absolute location modifiers if it is position-dependent. Subsequent rule options should use a relative modifier if they are meant to be relative to a previous rule option match in the stub data buffer. Any rule option that does not specify a relative modifier will be evaluated from the start of the stub data buffer. To leave the stub data buffer and return to the main payload buffer, use the "pkt_data" rule option.

byte_test and byte_jump

A DCE/RPC request can specify whether numbers are represented in big or little endian. These rule options will take as a new argument "dce" and will work basically the same as the normal byte_test/byte_jump, but since the DCE/RPC inspector will know the endianness of the request, it will be able to do the correct conversion.

Examples:

```
byte_test: 4,>,35000,0,relative,dce;
byte_test: 2,!=,2280,-10,relative,dce;
```

When using the "dce" argument to a byte_test, the following normal byte_test arguments will not be allowed: "big", "little", "string", "hex", "dec" and "oct".

Examples:

```
byte_jump:4,-4,relative,align,multiplier 2,post_offset -4,dce;
```

When using the dce argument to a byte_jump, the following normal byte_jump arguments will not be allowed: "big", "little", "string", "hex", "dec", "oct" and "from_beginning"

5.6 File Processing

With the volume of malware transferred through network increasing, network file inspection becomes more and more important. This feature will provide file type identification, file signature creation, and file capture capabilities to help users deal with those challenges.

5.6.1 Overview

There are two parts of file services: file APIs and file policy. File APIs provides all the file inspection functionalities, such as file type identification, file signature calculation, and file capture. File policy provides users ability to control file services, such as enable/disable/configure file type identification, file signature, or file capture.

In addition to all capabilities from Snort 2, we support customized file policy along with file event log.

- Supported protocols: HTTP, SMTP, IMAP, POP3, FTP, and SMB.
- Supported file signature calculation: SHA256

5.6.2 Quick Guide

A very simple configuration has been included in lua/snort.lua file. A typical file configuration looks like this:

```
dofile('magic.lua')
```

```
my_file_policy =
{
    {  when = { file_type_id = 0 }, use = { verdict = 'log', enable_file_signature ↩
        = true, enable_file_capture = true } }
    {  when = { file_type_id = 22 }, use = { verdict = 'log',  ↩
       enable_file_signature = true } },
    {  when = { sha256 = " ↩
       F74DC976BC8387E7D4FC0716A069017A0C7ED13F309A523CC41A8739CCB7D4B6" }, use =  ↩
       { verdict = 'block'} },
}

file_id =
{
    enable_type = true,
    enable_signature = true,
    enable_capture = true,
    file_rules = magics,
    trace_type = true,
    trace_signature = true,
    trace_stream = true,
    file_policy = my_file_policy,
 }

file_log =
{
    log_pkt_time = true,
    log_sys_time = false,
}
```

There are 3 steps to enable file processing:

- First, you need to include the file magic rules.
- Then, define the file policy and configure the inspector
- At last, enable file_log to get detailed information about file event

5.6.3 Pre-packaged File Magic Rules

A set of file magic rules is packaged with Snort. They can be located at "lua/file_magic.lua". To use this feature, it is recommended that these pre-packaged rules are used; doing so requires that you include the file in your Snort configuration as such (already in snort.lua):

```
dofile('magic.lua')
```

Example:

```
{ type = "GIF", id = 62, category = "Graphics", rev = 1,
  magic = { { content = "| 47 49 46 38 37 61 |",offset = 0 } } },

{ type = "GIF", id = 63, category = "Graphics", rev = 1,
  magic = { { content = "| 47 49 46 38 39 61 |",offset = 0 } } },
```

The previous two rules define GIF format, because two file magics are different. File magics are specified by content and offset, which look at content at particular file offset to identify the file type. In this case, two magics look at the beginning of the file. You can use character if it is printable or hex value in between "|".

5.6.4 File Policy

You can enabled file type, file signature, or file capture by configuring file_id. In addition, you can enable trace to see file stream data, file type, and file signature information.

Most importantly, you can configure a file policy that can block/alert some file type or an individual file based on SHA. This allows you build a file blacklist or whitelist.

Example:

```
file_policy =
{
    {  when = { file_type_id = 22 }, use = { verdict = 'log',  ↩
       enable_file_signature = true } },
    {  when = { sha256 = " ↩
       F74DC976BC8387E7D4FC0716A069017A0C7ED13F309A523CC41A8739CCB7D4B6" }, use =  ↩
       { verdict = 'block'} },
    {  when = { file_type_id = 0 }, use = { verdict = 'log', enable_file_signature ↩
        = true, enable_file_capture = true } }
}
```

In this example, it enables this policy:

- For PDF files, they will be logged with signatures.
- For the file matching this SHA, it will be blocked
- For all file types identified, they will be logged with signature, and also captured onto log folder.

5.6.5 File Capture

File can be captured and stored to log folder. We use SHA as file name instead of actual file name to avoid conflicts. You can capture either all files, some file type, or a particular file based on SHA.

You can enable file capture through this config:

```
enable_capture = true,
```

or enable it for some file or file type in your file policy:

```
{  when = { file_type_id = 22 }, use = { verdict = 'log', enable_file_capture =  ↩
   true } },
```

The above rule will enable PDF file capture.

5.6.6 File Events

File inspect preprocessor also works as a dynamic output plugin for file events. It logs basic information about file. The log file is in the same folder as other log files with name starting with "file.log".

Example:

```
file_log = { log_pkt_time = true, log_sys_time = false }
```

All file events will be logged in packet time, system time is not logged.

File event example:

```
08/14-19:14:19.100891  10.22.75.72:33734 -> 10.22.75.36:80,
[Name: "malware.exe"] [Verdict: Block] [Type: MSEXE]
[SHA: 6F26E721FDB1AAFD29B41BCF90196DEE3A5412550615A856DAE8E3634BCE9F7A]
[Size: 1039328]
```

5.7 High Availability

High Availability includes the HA flow synchronization and the SideChannel messaging subsystems.

5.7.1 HA

HighAvailability (or HA) is a Snort module that provides state coherency between two partner snort instances. It uses SideChannel for messaging.

There can be multiple types of HA within Snort and Snort plugins. HA implements an extensible architecture to enable plugins to subscribe to the base flow HA messaging. These plugins can then include their own messages along with the flow cache HA messages.

HA produces and consumes two type of messages:

- Update - Update flow status. Plugins may add their own data to the messages
- Delete - A flow has been removed from the cache

The HA module is configured with these items:

```
high_availability =
{
    ports = "1",
    enable = true,
    min_age = 0.0,
    min_sync = 0.0
}
```

The *ports* item maps to the SideChannel port to use for the HA messaging.

The *enabled* item controls the overall HA operation.

The items min_age and min_sync are used in the stream HA logic. min_age is the number of seconds that a flow must exist in the flow cache before sending HA messages to the partner. min_sync is the minimum time between HA status updates. HA messages for a particular flow will not be sent faster than min_sync. Both are expressed as a floating point number of seconds.

HA messages are composed of the base *stream* information plus any content from additional modules. Modules subscribe HA in order to add message content. The *stream* HA content is always present in the messages while the ancillary module content is only present when requested via a status change request.

5.7.2 Connector

Connectors are a set of modules that are used to exchange message-oriented data among Snort threads and the external world. A typical use-case is HA (High Availability) message exchange. Connectors serve to decouple the message transport from the message creation/consumption. Connectors expose a common API for several forms of message transport.

Connectors are a Snort plugin type.

Connector (parent plugin class)

Connectors may either be a simplex channel and perform unidirectional communications. Or may be duplex and perform bidirectional communications. The TcpConnector is duplex while the FileConnector is simplex.

All subtypes of Connector have a *direction* configuration element and a *connector* element. The *connector* string is the key used to identify the element for sidechannel configuration. The *direction* element may have a default value, for instance TcpConnector's are *duplex*.

There are currently two implementations of Connectors:

- TcpConnector - Exchange messages over a tcp channel.
- FileConnector - Write messages to files and read messages from files.

TcpConnector

TcpConnector is a subclass of Connector and implements a DUPLEX type Connector, able to send and receive messages over a tcp session.

TcpConnector adds a few session setup configuration elements:

- setup = *call* or *answer* - *call* is used to have TcpConnector initiate the connection. *answer* is used to have TcpConnector accept incoming connections.
- address = <*addr*> - used for *call* setup to specify the partner
- base_port = port - used to contruct the actual port number for *call* and *answer* modes. Actual port used is (base_port + instance_id).

An example segment of TcpConnector configuration:

```
tcp_connector =
{
    {
        connector = 'tcp_1',
        address = '127.0.0.1',
        setup = 'call',
        base_port = 11000
    },
}
```

FileConnector

FileConnector implements a Connector that can either read from files or write to files. FileConnector's are simplex and must be configured to be CONN_TRANSMIT or CONN_RECEIVE.

FileConnector configuration adds two additional element:

- name = string - used as part of the message file name
- format = *text* or *binary* - FileConnector supports two file types

The configured *name* string is used to construct the actual names as in:

- file_connector_NAME_transmit and file_connector_NAME_receive

All messages for one Snort invocation are read and written to one file.

In the case of a receive FileConnector, all messages are read from the file prior to the start of packet processing. This allows the messages to establish state information for all processed packets.

Connectors are used solely by SideChannel

An example segment of FileConnector configuration:

```
file_connector =
{
    {
        connector = 'file_tx_1',
        direction = 'transmit',
        format = 'text',
        name = 'HA'
    },
    {
```

```
        connector = 'file_rx_1',
        direction = 'receive',
        format = 'text',
        name = 'HA'
    },
}
```

5.7.3 Side Channel

SideChannel is a Snort module that uses Connectors to implement a messaging infrastructure that is used to communicate between Snort threads and the outside world.

SideChannel adds functionality onto the Connector as:

- message multiplexing/demultiplexing - An additional protocol layer is added to the messages. This port number is used to direct message to/from various SideClass instancs.
- application receive processing - handler for received messages on a specific port.

SideChannel's are always implement a duplex (bidirectional) messaging model and can map to separate transmit and receive Connectors.

The message handling model leverages the underlying Connector handling. So please refer to the Connector documentation.

SideChannel's are instantiated by various applications. The SideChannel port numbers are the configuration element used to map SideChannel's to applications.

The SideChannel configuration mostly serves to map a port number to a Connector or set of connectors. Each port mapping can have at most one transmit plus one receive connector or one duplex connector. Multiple SideChannel's may be configured and instantiated to support multiple applications.

An example SideChannel configuration along with the corresponding Connector configuration:

```
side_channel =
{
    {
        ports = '1',
        connectors =
        {
            {
                connector = 'file_rx_1',
            },
            {
                connector = 'file_tx_1',
            }
        },
    },
}

file_connector =
{
    {
        connector = 'file_tx_1',
        direction = 'transmit',
        format = 'text',
        name = 'HA'
    },
    {
```

```
        connector = 'file_rx_1',
        direction = 'receive',
        format = 'text',
        name = 'HA'
    },
}
```

5.8 FTP

Given an FTP command channel buffer, FTP will interpret the data, identifying FTP commands and parameters, as well as FTP response codes and messages. It will enforce correctness of the parameters, determine when an FTP command connection is encrypted, and determine when an FTP data channel is opened.

5.8.1 Configuring the inspector to block exploits and attacks

ftp_server configuration

- ftp_cmds

This specifies additional FTP commands outside of those checked by default within the inspector. The inspector may be configured to generate an alert when it sees a command it does not recognize.

Aside from the default commands recognized, it may be necessary to allow the use of the "X" commands, specified in RFC 775. To do so, use the following ftp_cmds option. Since these are rarely used by FTP client implementations, they are not included in the defaults.

```
ftp_cmds = [[ XPWD XCWD XCUP XMKD XRMD ]]
```

- def_max_param_len

This specifies the default maximum parameter length for all commands in bytes. If the parameter for an FTP command exceeds that length, and the inspector is configured to do so, an alert will be generated. This is used to check for buffer overflow exploits within FTP servers.

- cmd_validity

This specifies the valid format and length for parameters of a given command.

- cmd_validity[].len

This specifies the maximum parameter length for the specified command in bytes, overriding the default. If the parameter for that FTP command exceeds that length, and the inspector is configured to do so, an alert will be generated. It can be used to restrict specific commands to small parameter values. For example the USER command — usernames may be no longer than 16 bytes, so the appropriate configuration would be:

```
cmd_validity =
{
    {
        command = 'USER',
        length = 16,
    }
}
```

- cmd_validity[].format

format is as follows:

```
int                 Param must be an integer
number              Param must be an integer between 1 and 255
char <chars>        Param must be a single char, and one of <chars>
date <datefmt>      Param follows format specified where
                    # = Number, C=Char, []=optional, |=OR, {}=choice,
                    anything else=literal (i.e., .+- )
string              Param is string (effectively unrestricted)
host_port           Param must a host port specifier, per RFC 959.
long_host_port      Parameter must be a long host port specified, per RFC 1639
extended_host_port  Parameter must be an extended host port specified, per RFC  ↩
    2428
```

Examples of the cmd_validity option are shown below. These examples are the default checks (per RFC 959 and others) performed by the inspector.

```
cmd_validity =
{
    {
        command = 'CWD',
        length = 200,
    },
    {
        command = 'MODE',
        format = '< char SBC >',
    },
    {
        command = 'STRU',
        format = '< char FRP >',
    },
    {
        command = 'ALLO',
        format = '< int [ char R int ] >',
    },
    {
        command = 'TYPE',
        format = [[ < { char AE [ char NTC ] | char I | char L [ number ]
            } > ]],
    },
    {
        command = 'PORT',
        format = '< host_port >',
    },
}
```

A cmd_validity entry in the configuration can be used to override these defaults and/or add a check for other commands. A few examples follow.

This allows additional modes, including mode Z which allows for zip-style compression:

```
cmd_validity =
{
    {
        command = 'MODE',
        format = '< char ASBCZ >',
```

```
    },
}
```

Allow for a date in the MDTM command:

```
cmd_validity =
{
    {
        command = 'MDTM',
        format = '< [ date nnnnnnnnnnnnnn[.n[n[n]]] ] string >',
    },
}
```

MDTM is an odd case that is worth discussing...

While not part of an established standard, certain FTP servers accept MDTM commands that set the modification time on a file. The most common among servers that do, accept a format using YYYYMMDDHHmmss[.uuu]. Some others accept a format using YYYYMMDDHHmmss[+|-]TZ format. The example above is for the first case.

To check validity for a server that uses the TZ format, use the following:

```
cmd_validity =
{
    {
        command = 'MDTM',
        format = '< [ date nnnnnnnnnnnnnn[{+|-}n[n]] ] string >',
    },
}
```

- chk_str_fmt

This causes the inspector to check for string format attacks on the specified commands.

- telnet_cmds

Detect and alert when telnet cmds are seen on the FTP command channel.

- ignore_telnet_erase_cmds

This option allows Snort to ignore telnet escape sequences for erase character (TNC EAC) and erase line (TNC EAL) when normalizing FTP command channel. Some FTP servers do not process those telnet escape sequences.

- ignore_data_chan

When set to true, causes the FTP inspector to force the rest of snort to ignore the FTP data channel connections. NO INSPECTION other than state (inspector AND rules) will be performed on that data channel. It can be turned on to improve performance — especially with respect to large file transfers from a trusted source — by ignoring traffic. If your rule set includes virus-type rules, it is recommended that this option not be used.

ftp_client configuration

- max_resp_len

This specifies the maximum length for all response messages in bytes. If the message for an FTP response (everything after the 3 digit code) exceeds that length, and the inspector is configured to do so, an alert will be generated. This is used to check for buffer overflow exploits within FTP clients.

- telnet_cmds

Detect and alert when telnet cmds are seen on the FTP command channel.

- ignore_telnet_erase_cmds

This option allows Snort to ignore telnet escape sequences for erase character (TNC EAC) and erase line (TNC EAL) when normalizing FTP command channel. Some FTP clients do not process those telnet escape sequences.

ftp_data

In order to enable file inspection for ftp, the following should be added to the configuration:

```
ftp_data = {}
```

5.9 HTTP Inspector

One of the major undertakings for Snort 3 is developing a completely new HTTP inspector.

5.9.1 Overview

You can configure it by adding:

```
http_inspect = {}
```

to your snort.lua configuration file. Or you can read about it in the source code under src/service_inspectors/http_inspect.

So why a new HTTP inspector?

For starters it is object-oriented. That's good for us because we maintain this software. But it should also be really nice for open-source developers. You can make meaningful changes and additions to HTTP processing without having to understand the whole thing. In fact much of the new HTTP inspector's knowledge of HTTP is centralized in a series of tables where it can be easily reviewed and modified. Many significant changes can be made just by updating these tables.

http_inspect is the first inspector written specifically for the new Snort 3 architecture. This provides access to one of the very best features of Snort 3: purely PDU-based inspection. The classic preprocessor processes HTTP messages, but even while doing so it is constantly aware of IP packets and how they divide up the TCP data stream. The same HTTP message might be processed differently depending on how the sender (bad guy) divided it up into IP packets.

http_inspect is free of this burden and can focus exclusively on HTTP. This makes it much simpler, easier to test, and less prone to false positives. It also greatly reduces the opportunity for adversaries to probe the inspector for weak spots by adjusting packet boundaries to disguise bad behavior.

Dealing solely with HTTP messages also opens the door for developing major new features. The http_inspect design supports true stateful processing. Want to ask questions that involve both the client request and the server response? Or different requests in the same session? These things are possible.

Another new feature on the horizon is HTTP/2 analysis. HTTP/2 derives from Google's SPDY project and is in the process of being standardized. Despite the name, it is better to think of HTTP/2 not as a newer version of HTTP/1.1, but rather a separate protocol layer that runs under HTTP/1.1 and on top of TLS or TCP. It's a perfect fit for the new Snort 3 architecture because a new HTTP/2 inspector would naturally output HTTP/1.1 messages but not any underlying packets. Exactly what http_inspect wants to input.

http_inspect is taking a very different approach to HTTP header fields. The classic preprocessor divides all the HTTP headers following the start line into cookies and everything else. It normalizes the two pieces using a generic process and puts them in buffers that one can write rules against. There is some limited support for examining individual headers within the inspector but it is very specific.

The new concept is that every header should be normalized in an appropriate and specific way and individually made available for the user to write rules against it. If for example a header is supposed to be a date then normalization means put that date in a standard format.

5.9.2 Configuration

Configuration can be as simple as adding:

```
http_inspect = {}
```

to your snort.lua file. The default configuration provides a thorough inspection and may be all that you need. But there are some options that provide extra features, tweak how things are done, or conserve resources by doing less.

request_depth and response_depth

These replace the flow depth parameters used by the old HTTP inspector but they work differently.

The default is to inspect the entire HTTP message body. That's a very sound approach but if your HTTP traffic includes many very large files such as videos the load on Snort can become burdensome. Setting the request_depth and response_depth parameters will limit the amount of body data that is sent to the rule engine. For example:

```
request_depth = 10000,
response_depth = 80000,
```

would examine only the first 10000 bytes of POST, PUT, and other message bodies sent by the client. Responses from the server would be limited to 80000 bytes.

These limits apply only to the message bodies. HTTP headers are always completely inspected.

If you want to only inspect headers and no body, set the depth to 0. If you want to inspect the entire body set the depth to -1 or simply omit the depth parameter entirely because that is the default.

These limits have no effect on how much data is forwarded to file processing.

gzip

http_inspect by default decompresses deflate and gzip message bodies before inspecting them. This feature can be turned off by unzip = false. Turning off decompression provides a substantial performance improvement but at a very high price. It is unlikely that any meaningful inspection of message bodies will be possible. Effectively HTTP processing would be limited to the headers.

normalize_utf

http_inspect will decode utf-8, utf-7, utf-16le, utf-16be, utf-32le, and utf-32be in response message bodies based on the Content-Type header. This feature is on by default: normalize_utf = false will deactivate it.

decompress_pdf

decompress_pdf = true will enable decompression of compressed portions of PDF files encountered in a response body. http_inspect will examine the response body for PDF files that are then parsed to locate PDF streams with a single /FlateDecode filter. The compressed content is decompressed and made available through the file data rule option.

decompress_swf

decompress_swf = true will enable decompression of compressed SWF (Adobe Flash content) files encountered in a response body. The available decompression modes are 'deflate' and 'lzma'. http_inspect will search for the file signatures CWS for Deflate/ZLIB and ZWS for LZMA. The compressed content is decompressed and made available through the file data rule option. The compressed SWF file signature is converted to FWS to indicate an uncompressed file.

normalize_javascript

normalize_javascript = true will enable normalization of JavaScript within the HTTP response body. http_inspect looks for JavaScript by searching for the <script> tag without a type. Obfuscated data within the JavaScript functions such as unescape, String.fromCharCode, decodeURI, and decodeURIComponent are normalized. The different encodings handled within the unescape, decodeURI, or decodeURIComponent are %XX, %uXXXX, XX and uXXXXi. http_inspect also replaces consecutive whitespaces with a single space and normalizes the plus by concatenating the strings.

URI processing

Normalization and inspection of the URI in the HTTP request message is a key aspect of what http_inspect does. The best way to normalize a URI is very dependent on the idiosyncrasies of the HTTP server being accessed. The goal is to interpret the URI the same way as the server will so that nothing the server will see can be hidden from the rule engine.

The default URI inspection parameters are oriented toward following the HTTP RFCs—reading the URI the way the standards say it should be read. Most servers deviate from this ideal in various ways that can be exploited by an attacker. The options provide tools for the user to cope with that.

```
utf8 = true
plus_to_space = true
percent_u = false
utf8_bare_byte = false
iis_unicode = false
iis_double_decode = false
```

The HTTP inspector normalizes percent encodings found in URIs. For instance it will convert "%48%69%64%64%65%6e" to "Hidden". All the options listed above control how this is done. The options listed as true are fairly standard features that are decoded by default. You don't need to list them in snort.lua unless you want to turn them off by setting them to false. But that is not recommended unless you know what you are doing and have a definite reason.

The other options are primarily for the protection of servers that support irregular forms of decoding. These features are off by default but you can activate them if you need to by setting them to true in snort.lua.

```
bad_characters = "0x25 0x7e 0x6b 0x80 0x81 0x82 0x83 0x84"
```

That's a list of 8-bit Ascii characters that you don't want present in any normalized URI after the percent decoding is done. For example 0x25 is a hexadecimal number (37 in decimal) which stands for the % character. The % character is legitimately used for encoding special characters in a URI. But if there is still a percent after normalization one might conclude that something is wrong. If you choose to configure 0x25 as a bad character there will be an alert whenever this happens.

Another example is 0x00 which signifies the null character zero. Null characters in a URI are generally wrong and very suspicious.

The default is not to alert on any of the 256 8-bit Ascii characters. Add this option to your configuration if you want to define some bad characters.

```
ignore_unreserved = "abc123"
```

Percent encoding common characters such as letters and numbers that have no special meaning in HTTP is suspicious. It's legal but why would you do it unless you have something to hide? http_inspect will alert whenever an upper-case or lower-case letter, a digit, period, underscore, tilde, or minus is percent-encoded. But if a legitimate application in your environment encodes some of these characters for some reason this allows you to create exemptions for those characters.

In the example, the lower-case letters a, b, and c and the digits 1, 2, and 3 are exempted. These may be percent-encoded without generating an alert.

```
simplify_path = true
backslash_to_slash = false
```

HTTP inspector simplifies directory paths in URIs by eliminating extra traversals using ., .., and /.

For example I can take a simple URI such as

```
/very/easy/example
```

and complicate it like this:

```
/very/../very/./././easy/////detour/to/nowhere/../.././../example
```

which may be very difficult to match with a detection rule. simplify_path is on by default and you should not turn it off unless you have no interest in URI paths.

backslash_to_slash is a tweak to path simplification for servers that allow directories to be separated by backslashes:

```
/this/is/the/normal/way/to/write/a/path

\this\is\the\other\way\to\write\a\path
```

backslash_to_slash is turned off by default. If you are protecting such a server then set backslash_to_slash = true and all the backslashes will be replaced with slashes during normalization.

5.9.3 Detection rules

http_inspect parses HTTP messages into their components and makes them available to the detection engine through rule options. Let's start with an example:

```
alert tcp any any -> any any ( msg:"URI example"; flow:established,
to_server; http_uri; content:"chocolate"; sid:1; rev:1; )
```

This rule looks for chocolate in the URI portion of the request message. Specifically, the http_uri rule option is the normalized URI with all the percent encodings removed. It will find chocolate in both:

```
GET /chocolate/cake HTTP/1.1
```

and

```
GET /%63%68$6F%63%6F%6C%61%74%65/%63%61%6B%65 HTTP/1.1
```

It is also possible to search the unnormalized URI

```
alert tcp any any -> any any ( msg:"Raw URI example"; flow:established,
to_server; http_raw_uri; content:"chocolate"; sid:2; rev:1; )
```

will match the first message but not the second. If you want to detect someone who is trying to hide his request for chocolate then

```
alert tcp any any -> any any ( msg:"Raw URI example"; flow:established,
to_server; http_raw_uri; content:"%63%68$6F%63%6F%6C%61%74%65";
sid:3; rev:1; )
```

will do the trick.

Let's look at possible ways of writing a rule to match HTTP response messages with the Content-Language header set to "da" (Danish). You could write:

```
alert tcp any any -> any any ( msg:"whole header search";
flow:established, to_client; http_header; content:
"Content-Language: da", nocase; sid:4; rev:1; )
```

This rule leaves much to be desired. Modern headers are often thousands of bytes and seem to get longer every year. Searching all of the headers consumes a lot of resources. Furthermore this rule is easily evaded:

```
HTTP/1.1 ... Content-Language:  da ...
```

the extra space before the "da" throws the rule off. Or how about:

```
HTTP/1.1 ... Content-Language: xx,da ...
```

By adding a made up second language the attacker has once again thwarted the match.

A better way to write this rule is:

```
alert tcp any any -> any any ( msg:"individual header search";
flow:established, to_client; http_header: field content-language;
content:"da", nocase; sid:4; rev:2; )
```

The field option improves performance by narrowing the search to the Content-Language field of the header. Because it uses the header parsing abilities of http_inspect to find the field of interest it will not be thrown off by extra spaces or other languages in the list.

In addition to the headers there are rule options for virtually every part of the HTTP message.

http_uri and http_raw_uri

These provide the URI of the request message. The raw form is exactly as it appeared in the message and the normalized form is determined by the URI normalization options you selected. In addition to searching the entire URI there are six components that can be searched individually:

```
alert tcp any any -> any any ( msg:"URI path"; flow:established,
to_server; http_uri: path; content:"chocolate"; sid:1; rev:2; )
```

By specifying "path" the search is limited to the path portion of the URI. Informally this is the part consisting of the directory path and file name. Thus it will match:

```
GET /chocolate/cake HTTP/1.1
```

but not:

```
GET /book/recipes?chocolate+cake HTTP/1.1
```

The question mark ends the path and begins the query portion of the URI. Informally the query is where parameter values are set and often contains a search to be performed.

The six components are:

1. path: directory and file
2. query: user parameters
3. fragment: part of the file requested, normally found only inside a browser and not transmitted over the network
4. host: domain name of the server being addressed
5. port: TCP port number being addressed
6. scheme: normally "http" or "https" but others are possible such as "ftp"

Here is an example with all six:

```
GET https://www.samplehost.com:287/basic/example/of/path?with-query
#and-fragment HTTP/1.1\r\n
```

The URI is everything between the first space and the last space. "https" is the scheme, "www.samplehost.com" is the host, "287" is the port, "/basic/example/of/path" is the path, "with-query" is the query, and "and-fragment" is the fragment.

Note: this section uses informal language to explain some things. Nothing here is intended to conflict with the technical language of the HTTP RFCs and the implementation follows the RFCs.

http_header and http_raw_header

These cover all the header lines except the first one. You may specify an individual header by name using the field option as shown in this earlier example:

```
alert tcp any any -> any any ( msg:"individual header search";
flow:established, to_client; http_header: field content-language;
content:"da", nocase; sid:4; rev:2; )
```

This rule searches the value of the Content-Language header. Header names are not case sensitive and may be written in the rule in any mixture of upper and lower case.

With http_header the individual header value is normalized in a way that is appropriate for that header.

Specifying an individual header is not available for http_raw_header.

If you don't specify a header you get all of the headers except for the cookie headers Cookie and Set-Cookie. http_raw_header includes the unmodified header names and values as they appeared in the original message. http_header is the same except percent encodings are removed and paths are simplified exactly as if the headers were a URI.

In most cases specifying individual headers creates a more efficient and accurate rule. It is recommended that new rules be written using individual headers whenever possible.

http_trailer and http_raw_trailer

HTTP permits header lines to appear after a chunked body ends. Typically they contain information about the message content that was not available when the headers were created. For convenience we call them trailers.

http_trailer and http_raw_trailer are identical to their header counterparts except they apply to these end headers. If you want a rule to inspect both kinds of headers you need to write two rules, one using header and one using trailer.

http_cookie and http_raw_cookie

These provide the value of the Cookie header for a request message and the Set-Cookie for a response message. If multiple cookies are present they will be concatenated into a comma-separated list.

Normalization for http_cookie is the same URI-style normalization applied to http_header when no specific header is specified.

http_true_ip

This provides the original IP address of the client sending the request as it was stored by a proxy in the request message headers. Specifically it is the last IP address listed in the X-Forwarded-For or True-Client-IP header. If both headers are present the former is used.

http_client_body

This is the body of a request message such as POST or PUT. Normalization for http_client_body is the same URI-like normalization applied to http_header when no specific header is specified.

http_raw_body

This is the body of a request or response message. It will be dechunked and unzipped if applicable but will not be normalized in any other way. The difference between http_raw_body and packet data is a rule that uses packet data will search and may match an HTTP header, but http_raw_body is limited to the message body. Thus the latter is more efficient and more accurate for most uses.

http_method

The method field of a request message. Common values are "GET", "POST", "OPTIONS", "HEAD", "DELETE", "PUT", "TRACE", and "CONNECT".

http_stat_code

The status code field of a response message. This is normally a 3-digit number between 100 and 599. In this example it is 200.

```
HTTP/1.1 200 OK
```

http_stat_msg

The reason phrase field of a response message. This is the human-readable text following the status code. "OK" in the previous example.

http_version

The protocol version information that appears on the first line of an HTTP message. This is usually "HTTP/1.0" or "HTTP/1.1".

http_raw_request and http_raw_status

These are the unmodified first header line of the HTTP request and response messages respectively. These rule options are a safety valve in case you need to do something you cannot otherwise do. In most cases it is better to use a rule option for a specific part of the first header line. For a request message those are http_method, http_raw_uri, and http_version. For a response message those are http_version, http_stat_code, and http_stat_msg.

file_data and packet data

file_data contains the normalized message body. This is the normalization described above under gzip, normalize_utf, decompress_pdf, decompress_swf, and normalize_javascript.

The unnormalized message content is available in the packet data. If gzip is configured the packet data will be unzipped.

5.9.4 Timing issues and combining rule options

HTTP inspector is stateful. That means it is aware of a bigger picture than the packet in front of it. It knows what all the pieces of a message are, the dividing lines between one message and the next, which request message triggered which response message, pipelines, and how many messages have been sent over the current connection.

Some rules use a single rule option:

```
alert tcp any any -> any any ( msg:"URI example"; flow:established,
to_server; http_uri; content:"chocolate"; sid:1; rev:1; )
```

Whenever a new URI is available this rule will be evaluated. Nothing complicated about that, but suppose we use more than one rule option:

```
alert tcp any any -> any any ( msg:"combined example"; flow:established,
to_server; http_uri; content:"chocolate"; file_data;
content:"sinister POST data"; sid:5; rev:1; )
```

This rule requires both the URI and the request message body. That sounds simple until one considers that the message body may be millions of bytes long. The headers with the URI may be long gone by that time.

Is this rule going to work or do we need to do something different?

It is helpful to understand when things happen. All the message headers and the first few thousand bytes of the body go through detection at the same time. Commonly this is about 16K bytes but there are several exceptions and there is no guaranteed minimum amount.

That may be all you need. In many cases that will be the entire message. Or it may be more than your request_depth/response_depth. Or this rule may simply not care what happens after that in a very long message body.

Beyond that the message body will continue to be subdivided into roughly 16K-byte sections and inspected. But the previous rule will not be able to see the URI and hence will not work unless we rewrite it:

```
alert tcp any any -> any any ( msg:"URI with_body"; flow:established,
to_server; http_uri: with_body; content:"chocolate"; file_data;
content:"sinister POST data"; sid:5; rev:2; )
```

The with_body option to http_uri causes the URI to be made available with every body section, not just the first one. These extra inspections have a performance cost which is why they are not done automatically. with_body is an option to be used when you actually need it.

The with_trailer option is analogous and causes an earlier message element to be made available at the end of the message when the trailers following a chunked body arrive.

```
alert tcp any any -> any any ( msg:"double content-language";
flow:established, to_client; http_header: with_trailer, field
content-language; content:"da", nocase; http_trailer: field
content-language; content:"en", nocase; sid:6; rev:1; )
```

This rule will alert if the Content-Language changes from Danish in the headers to English in the trailers. The with_trailer option is essential to make this rule work.

It is also possible to write rules that examine both the client request and the server response to it.

```
alert tcp any any -> any any ( msg:"request and response example";
flow:established, to_client; http_uri: with_body; content:"chocolate";
file_data; content:"white chocolate"; sid:7; rev:1; )
```

This rule looks for white chocolate in a response message body where the URI of the request contained chocolate. Note that this is a "to_client" rule that will alert on and potentially block a server response containing white chocolate, but only if the client URI requested chocolate. If the rule were rewritten "to_server" it would be nonsense and not work. Snort cannot block a client request based on what the server response will be because that has not happened yet.

Another point is "with_body" for http_uri. This ensures the rule works on the entire response body. If we were looking for white chocolate in the response headers this would not be necessary.

Response messages do not have a URI so there was only one thing http_uri could have meant in the previous rule. It had to be referring to the request message. Sometimes that is not so clear.

```
alert tcp any any -> any any ( msg:"header ambiguity example 1";
flow:established, to_client; http_header: with_body; content:
"chocolate"; file_data; content:"white chocolate"; sid:8; rev:1; )


alert tcp any any -> any any ( msg:"header ambiguity example 2";
flow:established, to_client; http_header: with_body, request; content:
"chocolate"; file_data; content:"white chocolate"; sid:8; rev:2; )
```

Our search for chocolate has moved from the URI to the message headers. Both the request and response messages have headers—which one are we asking about? Ambiguity is always resolved in favor of looking in the current message which is the response. The first rule is looking for a server response containing chocolate in the headers and white chocolate in the body.

The second rule uses the "request" option to explicitly say that the http_header to be searched is the request header.

Let's put all of this together. There are six opportunities to do detection:

1. When the first part of the request message body arrives. The request line, all of the headers, and the first part of the body all go through detection at the same time. Of course most requests don't have a body. In that case the request line and the headers are the whole message and get done at the same time.
2. When subsequent sections of the request message body arrive. If you want to combine this with something from the request line or headers you must use the with_body option.
3. When the request trailers arrive. If you want to combine this with something from the request line or headers you must use the with_trailer option.
4. When the first part of the response message body arrives. The status line, all of the headers, and the first part of the body all go through detection at the same time. These may be combined with elements from the request line, request headers, or request trailers. Where ambiguity arises use the request option.
5. When subsequent sections of the response message body arrive. These may be combined with the status line, response headers, request line, request headers, or request trailers as described above.
6. When the response trailers arrive. Again these may be combined as described above.

Message body data can only go through detection at the time it is received. Headers may be combined with later items but the body cannot.

5.10 HTTP/2 Inspector

Snort 3 is developing an inspector for HTTP/2.

You can configure it by adding:

```
http2_inspect = {}
```

to your snort.lua configuration file.

Everything has a beginning and for http2_inspect this is the beginning of the beginning. Most of the protocol including HPACK decompression is not implemented yet.

Currently http2_inspect will divide an HTTP/2 connection into individual frames and make them available for detection. Two new rule options are available for looking at HTTP/2 frames: http2_frame_header provides the 9-octet frame header and http2_frame_data provides the frame content.

```
alert tcp any any -> any any (msg:"Frame type"; flow:established,
to_client; http2_frame_header; content:"|06|", offset 3, depth 1;
sid:1; rev:1; )
```

This will match if the Type byte of the frame header is 6 (PING).

```
alert tcp any any -> any any ( msg:"Content of HTTP/2 frame";
flow:established, to_client; http2_frame_data; content:"peppermint";
sid:2; rev:1; )
```

This will look for peppermint in the frame data but not the frame header.

These can be combined:

```
alert tcp any any -> any any ( msg:"Search in message bodies";
flow:established, to_client;
http2_frame_header; content:"|00|", offset 3, depth 1;
http2_frame_data; content:"MaLwArE"; sid:3; rev:1; )
```

Frame type 0 is DATA which carries the HTTP message body. This rule will search for MaLwArE inside an HTTP message body.

In the future, http2_inspect will support HPACK header decompression and be fully integrated with http_inspect to provide full inspection of the individual HTTP/1.1 streams.

5.11 Module Trace

Snort 3 retired the different flavors of debug macros that used to be set through environment variable SNORT_DEBUG. It was replaced by a module specific trace. Trace is turned on by setting the module-specific trace bitmask in snort.lua. As before, in order to enable it, snort has to be configured and built with --enable-debug-msgs.

5.11.1 Debugging rules using detection trace

Detection engine is responsible for rule evaluation. Turning on the trace for it can help with debugging new rules.

The relevant options for detection are as follow (represented as hex):

```
0x2 - follow rule evaluation
0x4 - print evaluated buffer if it changed
0x8 - print evaluated buffer at every step
0x10 - print value of ips rule options vars
0x20 - print information on fast pattern search
```

Buffer print is useful, but in case the buffer is very big can be too verbose. Choose between 0x4, 0x8 or no buffer trace accordingly.

0x10 is useful when the rule is using ips rule options vars.

5.11.2 Example - rule evaluation traces:

In snort.lua, the following line was added:

detection = {trace = 0x20 + 0x10 + 0x2 + 0x4}

The pcap has a single packet with payload: 10.AAAAAAAfoobar

Evaluated on rules:

```
# byte_math + oper with byte extract and content
# VAL = 1, byte_math = 0 + 10
alert tcp ( byte_extract: 1, 0, VAL, string, dec;
byte_math:bytes 1,offset VAL,oper +, rvalue 10, result var1, string dec;
content:"foo", offset var1; sid:3)


#This rule should not trigger
alert tcp (content:"AAAAA"; byte_jump:2,0,relative;
content:"foo", within 3; sid:2)
```

The output:

```
detection: packet 1 C2S 127.0.0.1:1234 127.0.0.1:5678
detection: Fast pattern search
detection: 1 fp packet[16]
```

```
snort.raw[16]:
- - - - - - - - - - - - - - - -  - - - - - - - - - - - - - - - -  - - - - -  - - - - -
31 30 00 41 41 41 41 41 41 41  66 6F 6F 62 61 72              10.AAAAAAAfoobar
- - - - - - - - - - - - - - - -  - - - - - - - - - - - - - - - -  - - - - -  - - - - -
detection: Processing pattern match #1
detection: Fast pattern packet[5] = 'AAAAA' |41 41 41 41 41 | ( )
detection: Starting tree eval
detection: Evaluating option content, cursor name pkt_data, cursor position 0

snort.raw[16]:
- - - - - - - - - - - - - - - -  - - - - - - - - - - - - - - - -  - - - - -  - - - - -
31 30 00 41 41 41 41 41 41 41  66 6F 6F 62 61 72              10.AAAAAAAfoobar
- - - - - - - - - - - - - - - -  - - - - - - - - - - - - - - - -  - - - - -  - - - - -
detection: Rule options variables:
var[0]=0 var[1]=0 var[2]=0
detection: Evaluating option byte_jump, cursor name pkt_data, cursor position 8

snort.raw[8]:
- - - - - - - - - - - - - - - -  - - - - - - - - - - - - - - - -  - - - - -  - - - - -
41 41 66 6F 6F 62 61 72                                        AAfoobar
- - - - - - - - - - - - - - - -  - - - - - - - - - - - - - - - -  - - - - -  - - - - -
detection: no match
detection: Rule options variables:
var[0]=0 var[1]=0 var[2]=0
detection: Evaluating option byte_jump, cursor name pkt_data, cursor position 9

snort.raw[7]:
- - - - - - - - - - - - - - - -  - - - - - - - - - - - - - - - -  - - - - -  - - - - -
41 66 6F 6F 62 61 72                                           Afoobar
- - - - - - - - - - - - - - - -  - - - - - - - - - - - - - - - -  - - - - -  - - - - -
detection: no match
detection: Rule options variables:
var[0]=0 var[1]=0 var[2]=0
detection: Evaluating option byte_jump, cursor name pkt_data, cursor position 10

snort.raw[6]:
- - - - - - - - - - - - - - - -  - - - - - - - - - - - - - - - -  - - - - -  - - - - -
66 6F 6F 62 61 72                                              foobar
- - - - - - - - - - - - - - - -  - - - - - - - - - - - - - - - -  - - - - -  - - - - -
detection: no match
detection: no match
detection: Processing pattern match #2
detection: Fast pattern packet[3] = 'foo' |66 6F 6F | ( )
detection: Starting tree eval
detection: Evaluating option byte_extract, cursor name pkt_data, cursor position 0

snort.raw[16]:
- - - - - - - - - - - - - - - -  - - - - - - - - - - - - - - - -  - - - - -  - - - - -
31 30 00 41 41 41 41 41 41 41  66 6F 6F 62 61 72              10.AAAAAAAfoobar
- - - - - - - - - - - - - - - -  - - - - - - - - - - - - - - - -  - - - - -  - - - - -
detection: Rule options variables:
var[0]=1 var[1]=0 var[2]=0
detection: Evaluating option byte_math, cursor name pkt_data, cursor position 1
```

```
snort.raw[15]:
- - - - - - - - - - - - - - -  - - - - - - - - - - - - - - - -  - - - - -  - - - - -
30 00 41 41 41 41 41 41 66  6F 6F 62 61 72                 0.AAAAAAAfoobar
- - - - - - - - - - - - - - -  - - - - - - - - - - - - - - - -  - - - - -  - - - - -
detection: Rule options variables:
var[0]=1 var[1]=10 var[2]=0
detection: Evaluating option content, cursor name pkt_data, cursor position 2

snort.raw[14]:
- - - - - - - - - - - - - - -  - - - - - - - - - - - - - - - -  - - - - -  - - - - -
00 41 41 41 41 41 41 41 66 6F  6F 62 61 72                   .AAAAAAAfoobar
- - - - - - - - - - - - - - -  - - - - - - - - - - - - - - - -  - - - - -  - - - - -
detection: Rule options variables:
var[0]=1 var[1]=10 var[2]=0
detection: Reached leaf, cursor name pkt_data, cursor position 13

snort.raw[3]:
- - - - - - - - - - - - - - -  - - - - - - - - - - - - - - - -  - - - - -  - - - - -
62 61 72                                                    bar
- - - - - - - - - - - - - - -  - - - - - - - - - - - - - - - -  - - - - -  - - - - -
detection: Matched rule gid:sid:rev 1:3:0
detection: Rule options variables:
var[0]=1 var[1]=10 var[2]=0
04/22-20:21:40.905630, 1, TCP, raw, 56, C2S, 127.0.0.1:1234, 127.0.0.1:5678,  ↩
    1:3:0, allow
```

5.11.3 Protocols decoding trace

Turning on decode trace will print out information about the packets decoded protocols. Can be useful in case of tunneling.

Example for a icmpv4-in-ipv6 packet:

In snort.lua, the following line was added:

decode = { trace = 1 }

The output:

```
decode: Codec eth (protocol_id: 34525) ip header starts at: 0x7f70800110f0, length ↩
     is 14
decode: Codec ipv6 (protocol_id: 1) ip header starts at: 0x7f70800110f0, length is ↩
     40
decode: Codec icmp4 (protocol_id: 256) ip header starts at: 0x7f70800110f0, length ↩
     is 8
decode: Codec unknown (protocol_id: 256) ip header starts at: 0x7f70800110f0,  ↩
    length is 0
```

5.11.4 Other available traces

There are more trace options supported by detection:

```
0x1 - prints statistics about the engine
0x40 - prints a message when disabling content detect for packet
0x80 - prints option tree data structure
0x100 - prints a message when a new tag is added
```

Detection is the only module that support multiple options for trace.

The rest support only 1 option, and can be turned on by adding trace = 1 to their lua config.

- stream module trace:

When turned on prints a message in case inspection is stopped on a flow. Example for output:

stream: stop inspection on flow, dir BOTH

- stream_ip, stream_user: trace will output general processing messages

Other modules that support trace have messages as seemed fit to the developer. Some are for corner cases, other for complex data structures prints. Current list of additional modules supporting trace: appid, dce_smb, gtp_inspect and dce_udp.

5.12 Performance Monitor

The new and improved performance monitor! Is your sensor being bogged down by too many flows? perf_monitor! Why are certain TCP segments being dropped without hitting a rule? perf_monitor! Why is a sensor leaking water? Not perf_monitor, check with stream...

5.12.1 Overview

The Snort performance monitor is the built-in utility for monitoring system and traffic statistics. All statistics are separated by processing thread. perf_monitor supports several trackers for monitoring such data:

5.12.2 Base Tracker

The base tracker is used to gather running statistics about Snort and its running modules. All Snort modules gather, at the very least, counters for the number of packets reaching it. Most supplement these counts with those for domain specific functions, such as http_inspect's number of GET requests seen.

Statistics are gathered live and can be reported at regular intervals. The stats reported correspond only to the interval in question and are reset at the beginning of each interval.

These are the same counts displayed when Snort shuts down, only sorted amongst the discrete intervals in which they occurred.

Base differs from prior implementations in Snort in that all stats gathered are only raw counts, allowing the data to be evaluated as needed. Additionally, base is entirely pluggable. Data from new Snort plugins can be added to the existing stats either automatically or, if specified, by name and function.

All plugins and counters can be enabled or disabled individually, allowing for only the data that is actually desired instead of overly verbose performance logs.

To enable everything:

```
perf_monitor = { modules = {} }
```

To enable everything within a module:

```
perf_monitor =
{
    modules =
    {
        {
            name = 'stream_tcp',
            pegs = [[ ]]
        },
    }
}
```

To enable specific counts within modules:

```
perf_monitor =
{
    modules =
    {
        {
            name = 'stream_tcp',
            pegs = [[ overlaps gaps ]]
        },
    }
```

Note: Event stats from prior Snorts are now located within base statistics.

5.12.3 Flow Tracker

Flow tracks statistics regarding traffic and L3/L4 protocol distributions. This data can be used to build a profile of traffic for inspector tuning and for identifying where Snort may be stressed.

To enable:

```
perf_monitor = { flow = true }
```

5.12.4 FlowIP Tracker

FlowIP provides statistics for individual hosts within a network. This data can be used for identifying communication habits, such as generating large or small amounts of data, opening a small or large number of sessions, and tendency to send smaller or larger IP packets.

To enable:

```
perf_monitor = { flow_ip = true }
```

5.12.5 CPU Tracker

This tracker monitors the CPU and wall time spent by a given processing thread.

To enable:

```
perf_monitor = { cpu = true }
```

5.12.6 Formatters

Performance monitor allows statistics to be output in a few formats. Along with human readable text (as seen at shutdown) and csv formats, a Flatbuffers binary format is also available if Flatbuffers is present at build. A utility for accessing the statistics generated in this format has been included for convenience (see fbstreamer in tools). This tool generates a YAML array of records found, allowing the data to be read by humans or passed into other analysis tools. For information on working directly with the Flatbuffers file format used by Performance monitor, see the developer notes for Performance monitor or the code provided for fbstreamer.

5.13 POP and IMAP

POP inspector is a service inspector for POP3 protocol and IMAP inspector is for IMAP4 protocol.

5.13.1 Overview

POP and IMAP inspectors examine data traffic and find POP and IMAP commands and responses. The inspectors also identify the command, header, body sections and extract the MIME attachments and decode it appropriately. The pop and imap also identify and whitelist the pop and imap traffic.

5.13.2 Configuration

POP inspector and IMAP inspector offer same set of configuration options for MIME decoding depth. These depths range from 0 to 65535 bytes. Setting the value to 0 ("do none") turns the feature off. Alternatively the value -1 means an unlimited amount of data should be decoded. If you do not specify the default value is 1460 bytes.

The depth limits apply per attachment. They are:

b64_decode_depth

Set the base64 decoding depth used to decode the base64-encoded MIME attachments.

qp_decode_depth

Set the Quoted-Printable (QP) decoding depth used to decode QP-encoded MIME attachments.

bitenc_decode_depth

Set the non-encoded MIME extraction depth used for non-encoded MIME attachments.

uu_decode_depth

Set the Unix-to-Unix (UU) decoding depth used to decode UU-encoded attachments.

Examples

```
stream = { }

stream_tcp = { }

stream_ip = { }

binder =
{
    {
        {
            when = { proto = 'tcp', ports = '110', },
            use = { type = 'pop', },
        },
        {
            when = { proto = 'tcp', ports = '143', },
            use =  { type = 'imap', },
        },
    },
}
```

```
imap =
{
    qp_decode_depth = 500,
}

pop =
{
    qp_decode_depth = -1,
    b64_decode_depth = 3000,
}
```

5.14 Port Scan

A module to detect port scanning

5.14.1 Overview

This module 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 services 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. Portscan was designed to be able to detect the different types of scans Nmap can produce.

The following are a list of the types of Nmap scans Portscan will currently alert for.

- TCP Portscan
- UDP Portscan
- IP Portscan

These alerts are for one to 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.

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

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

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

These are many to 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.

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

These alerts are for one to 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.

- 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 on 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.

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

5.14.2 Scan levels

There are 3 default scan levels that can be set.

```
1) default_hi_port_scan
2) default_med_port_scan
3) default_low_port_scan
```

Each of these default levels have separate options that can be edited to alter the scan sensitivity levels (scans, rejects, nets or ports)

Example:

```
port_scan = default_low_port_scan

port_scan.tcp_decoy.ports = 1
port_scan.tcp_decoy.scans = 1
port_scan.tcp_decoy.rejects = 1
port_scan.tcp_ports.nets = 1
```

The example above would change each of the individual settings to 1.

NOTE:The default levels for scans, rejects, nets and ports can be seen in the snort_defaults.lua file.

The counts can be seen in the alert outputs (-Acmg shown below):

```
50 72 69 6F 72 69 74 79  20 43 6F 75 6E 74 3A 20  Priority  Count:
30 0A 43 6F 6E 6E 65 63  74 69 6F 6E 20 43 6F 75  0.Connec tion Cou
6E 74 3A 20 34 35 0A 49  50 20 43 6F 75 6E 74 3A  nt: 45.I P Count:
20 31 0A 53 63 61 6E 6E  65 72 20 49 50 20 52 61  1.Scann er IP Ra
6E 67 65 3A 20 31 2E 32  2E 33 2E 34 3A 31 2E 32  nge: 1.2 .3.4:1.2
2E 33 2E 34 0A 50 6F 72  74 2F 50 72 6F 74 6F 20  .3.4.Por t/Proto
43 6F 75 6E 74 3A 20 33  37 0A 50 6F 72 74 2F 50  Count: 3 7.Port/P
72 6F 74 6F 20 52 61 6E  67 65 3A 20 31 3A 39 0A  roto Ran ge: 1:9.
```

"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" 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" 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 Portscan.

5.14.3 Tuning Portscan

The most important aspect in detecting portscans is tuning the detection engine for your network(s). Here are some tuning tips:

Use the watch_ip, ignore_scanners, and ignore_scanned options. It's important to correctly set these options. The watch_ip option is easy to understand. The analyst should set this option to the list of CIDR blocks and IPs that they want to watch. If no watch_ip is defined, Portscan will watch all network traffic. The ignore_scanners and ignore_scanned options come into play in weeding out legitimate hosts that are very active on your network. Some of the most common examples are NAT IPs, DNS cache servers, syslog servers, and nfs servers. Portscan may not generate false positives for these types of hosts, but be aware when first tuning Portscan for these IPs. Depending on the type of alert that the host generates, the analyst will know which to ignore it as. If the host is generating portsweep events, then add it to the ignore_scanners option. If the host is generating portscan alerts (and is the host that is being scanned), add it to the ignore_scanned option.

Filtered scan alerts are much more prone to false positives. When determining false positives, the alert type is very important. Most of the false positives that Portscan may generate are of the filtered scan alert type. So be much more suspicious of filtered

portscans. Many times this just indicates that a host was very active during the time period in question. If the host continually generates these types of alerts, add it to the ignore_scanners list or use a lower sensitivity level.

Make use of the Priority Count, Connection Count, IP Count, Port Count, IP range, and Port range to determine false positives. The portscan alert details are vital in determining the scope of a portscan and also the confidence of the portscan. In the future, we hope to automate much of this analysis in assigning a scope level and confidence level, but for now the user must manually do this. The easiest way to determine false positives is through simple ratio estimations. The following is a list of ratios to estimate and the associated values that indicate a legitimate scan and not a false positive.

Connection Count / IP Count: This ratio indicates an estimated average of connections per IP. For portscans, this ratio should be high, the higher the better. For portsweeps, this ratio should be low.

Port Count / IP Count: This ratio indicates an estimated average of ports connected to per IP. For portscans, this ratio should be high and indicates that the scanned host's ports were connected to by fewer IPs. For portsweeps, this ratio should be low, indicating that the scanning host connected to few ports but on many hosts.

Connection Count / Port Count: This ratio indicates an estimated average of connections per port. For portscans, this ratio should be low. This indicates that each connection was to a different port. For portsweeps, this ratio should be high. This indicates that there were many connections to the same port.

The reason that Priority Count is not included, is because the priority count is included in the connection count and the above comparisons take that into consideration. The Priority Count play an important role in tuning because the higher the priority count the more likely it is a real portscan or portsweep (unless the host is firewalled).

If all else fails, lower the sensitivity level. If none of these other tuning techniques work or the analyst doesn't have the time for tuning, lower the sensitivity level. You get the best protection the higher the sensitivity level, but it's also important that the portscan detection engine generates alerts that the analyst will find informative. The low sensitivity level only generates alerts based on error responses. These responses indicate a portscan and the alerts generated by the low sensitivity level are highly accurate and require the least tuning. The low sensitivity level does not catch filtered scans, since these are more prone to false positives.

5.15 Sensitive Data Filtering

The `sd_pattern` IPS option provides detection and filtering of Personally Identifiable Information (PII). This information includes credit card numbers, U.S. Social Security numbers, and email addresses. A rich regular expression syntax is available for defining your own PII.

5.15.1 Hyperscan

The `sd_pattern` rule option is powered by the open source Hyperscan library from Intel. It provides a regex grammar which is mostly PCRE compatible. To learn more about Hyperscan see https://intel.github.io/hyperscan/dev-reference/

5.15.2 Syntax

Snort provides `sd_pattern` as IPS rule option with no additional inspector overhead. The Rule option takes the following syntax.

```
sd_pattern: "<pattern>"[, threshold <count>];
```

Pattern

Pattern is the most important and is the only required parameter to `sd_pattern`. It supports 3 built in patterns which are configured by name: "credit_card", "us_social" and "us_social_nodashes", as well as user defined regular expressions of the Hyperscan dialect (see https://intel.github.io/hyperscan/dev-reference/compilation.html#pattern-support).

```
sd_pattern:"credit_card";
```

When configured, Snort will replace the pattern *credit_card* with the built in pattern. In addition to pattern matching, Snort will validate that the matched digits will pass the Luhn-check algorithm. Currently the only pattern that performs extra verification.

```
sd_pattern:"us_social";
sd_pattern:"us_social_nodashes";
```

These special patterns will also be replaced with a built in pattern. Naturally, "us_social" is a pattern of 9 digits separated by −'s in the canonical form.

```
sd_pattern:"\b\w+@ourdomain\.com\b"
```

This is a user defined pattern which matches what is most likely email addresses for the site "ourdomain.com". The pattern is a PCRE compatible regex, \b matches a word boundary (whitespace, end of line, non-word characters) and \w+ matches one or more word characters. \. matches a literal ..

The above pattern would match "a@ourdomain.com", "aa@ourdomain.com" but would not match `1@ourdomain.com` `ab12@ourdomain.com` or `@ourdomain.com`.

Note: This is just an example, this pattern is not suitable to detect many correctly formatted emails.

Threshold

Threshold is an optional parameter allowing you to change built in default value (default value is *1*). The following two instances are identical. The first will assume the default value of *1* the second declaration explicitly sets the threshold to *1*.

```
sd_pattern:"This rule requires 1 match";
sd_pattern:"This rule requires 1 match", threshold 1;
```

That's pretty easy, but here is one more example anyway.

```
sd_pattern:"This is a string literal", threshold 300;
```

This example requires 300 matches of the pattern "This is a string literal" to qualify as a positive match. That is, if the string only occurred 299 times in a packet, you will not see an event.

Obfuscating Credit Cards and Social Security Numbers

Snort provides discreet logging for the built in patterns "credit_card", "us_social" and "us_social_nodashes". Enabling `output.obfuscate_pii` makes Snort obfuscate the suspect packet payload which was matched by the patterns. This configuration is disabled by default.

```
output =
{
    obfuscate_pii = true
}
```

5.15.3 Example

A complete Snort IPS rule

```
alert tcp ( sid:1; msg:"Credit Card"; sd_pattern:"credit_card"; )
```

Logged output when running Snort in "cmg" alert format.

```
02/25-21:19:05.125553 [**] [1:1:0] "Credit Card" [**] [Priority: 0] {TCP}  ↩
   10.1.2.3:48620 -> 10.9.8.7:8
02:01:02:03:04:05 -> 02:09:08:07:06:05 type:0x800 len:0x46
10.1.2.3:48620 -> 10.9.8.7:8 TCP TTL:64 TOS:0x0 ID:14 IpLen:20 DgmLen:56
***A**** Seq: 0xB2  Ack: 0x2  Win: 0x2000  TcpLen: 20
- - - raw[16] - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
58 58 58 58 58 58 58 58 58 58 58 39 32 39 34              XXXXXXXXXXXX9294
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

5.15.4 Caveats

1. Snort currently requires setting the fast pattern engine to use "hyperscan" in order for `sd_pattern` ips option to function correctly.

   ```
   search_engine = { search_method = 'hyperscan' }
   ```

2. Log obfuscation is only applicable to CMG and Unified2 logging formats.
3. Log obfuscation doesn't support user defined PII patterns. It is currently only supported for the built in patterns for Credit Cards and US Social Security numbers.
4. Log obfuscation doesn't work with stream rebuilt packet payloads. (This is a known bug).

5.16 SMTP

SMTP inspector is a service inspector for SMTP protocol.

5.16.1 Overview

The SMTP inspector examines SMTP connections looking for commands and responses. It also identifies the command, header and body sections, TLS data and extracts the MIME attachments. This inspector also identifies and whitelists the SMTP traffic.

SMTP inspector logs the filename, email addresses, attachment names when configured.

5.16.2 Configuration

SMTP command lines can be normalized to remove extraneous spaces. TLS-encrypted traffic can be ignored, which improves performance. In addition, plain-text mail data can be ignored for an additional performance boost.

The configuration options are described below:

normalize and normalize_cmds

Normalization checks for more than one space character after a command. Space characters are defined as space (ASCII 0x20) or tab (ASCII 0x09). "normalize" provides options all|none|cmds, *all* checks all commands, *none* turns off normalization for all commands. *cmds* just checks commands listed with the "normalize_cmds" parameter. For example:

```
smtp = { normalize = 'cmds', normalize_cmds = 'RCPT VRFY EXPN' }
```

ignore_data

Set it to true to ignore data section of mail (except for mail headers) when processing rules.

ignore_tls_data

Set it to true to ignore TLS-encrypted data when processing rules.

max_command_line_len

Alert if an SMTP command line is longer than this value. Absence of this option or a "0" means never alert on command line length. RFC 2821 recommends 512 as a maximum command line length.

max_header_line_len

Alert if an SMTP DATA header line is longer than this value. Absence of this option or a "0" means never alert on data header line length. RFC 2821 recommends 1024 as a maximum data header line length.

max_response_line_len

Alert if an SMTP response line is longer than this value. Absence of this option or a "0" means never alert on response line length. RFC 2821 recommends 512 as a maximum response line length.

alt_max_command_line_len

Overrides max_command_line_len for specific commands For example:

```
alt_max_command_line_len =
{
    {
        command = 'MAIL',
        length = 260,
    },
    {
        command = 'RCPT',
        length = 300,
    },
}
```

invalid_cmds

Alert if this command is sent from client side.

valid_cmds

List of valid commands. We do not alert on commands in this list.

DEFAULT empty list, but SMTP inspector has this list hard-coded: [[ATRN AUTH BDAT DATA DEBUG EHLO EMAL ESAM ESND ESOM ETRN EVFY EXPN HELO HELP IDENT MAIL NOOP ONEX QUEU QUIT RCPT RSET SAML SEND SIZE STARTTLS SOML TICK TIME TURN TURNME VERB VRFY X-EXPS X-LINK2STATE XADR XAUTH XCIR XEXCH50 XGEN XLICENSE XQUE XSTA XTRN XUSR]]

data_cmds

List of commands that initiate sending of data with an end of data delimiter the same as that of the DATA command per RFC 5321 - "<CRLF>.<CRLF>".

binary_data_cmds

List of commands that initiate sending of data and use a length value after the command to indicate the amount of data to be sent, similar to that of the BDAT command per RFC 3030.

auth_cmds

List of commands that initiate an authentication exchange between client and server.

xlink2state

Enable/disable xlink2state alert, options are {disable | alert | drop}. See CVE-2005-0560 for a description of the vulnerability.

MIME processing depth parameters

These four MIME processing depth parameters are identical to their POP and IMAP counterparts. See that section for further details.

b64_decode_depth qp_decode_depth bitenc_decode_depth uu_decode_depth

Log Options

Following log options allow SMTP inspector to log email addresses and filenames. Please note, this is logged only with the unified2 output and is not logged with the console output (-A cmg). u2spewfoo can be used to read this data from the unified2.

log_mailfrom

This option enables SMTP inspector to parse and log the sender's email address extracted from the "MAIL FROM" command along with all the generated events for that session. The maximum number of bytes logged for this option is 1024.

log_rcptto

This option enables SMTP inspector to parse and log the recipient email addresses extracted from the "RCPT TO" command along with all the generated events for that session. Multiple recipients are appended with commas. The maximum number of bytes logged for this option is 1024.

log_filename

This option enables SMTP inspector to parse and log the MIME attachment filenames extracted from the Content-Disposition header within the MIME body along with all the generated events for that session. Multiple filenames are appended with commas. The maximum number of bytes logged for this option is 1024.

log_email_hdrs

This option enables SMTP inspector to parse and log the SMTP email headers extracted from SMTP data along with all generated events for that session. The number of bytes extracted and logged depends upon the email_hdrs_log_depth.

email_hdrs_log_depth

This option specifies the depth for logging email headers. The allowed range for this option is 0 - 20480. A value of 0 will disable email headers logging. The default value for this option is 1464.

5.16.3 Example

```
smtp =
{
    normalize = 'cmds',
    normalize_cmds = 'EXPN VRFY RCPT',
    b64_decode_depth = 0,
    qp_decode_depth = 0,
```

```
    bitenc_decode_depth = 0,
    uu_decode_depth = 0,
    log_mailfrom = true,
    log_rcptto = true,
    log_filename = true,
    log_email_hdrs = true,
    max_command_line_len = 512,
    max_header_line_len = 1000,
    max_response_line_len = 512,
    max_auth_command_line_len = 50,
    xlink2state = 'alert',
    alt_max_command_line_len =
    {
        {
            command = 'MAIL',
            length = 260,
        },
        {
            command = 'RCPT',
            length = 300,
        },
        {
            command = 'HELP',
            length = 500,
        },
        {
            command = 'HELO',
            length = 500,
        },
        {
            command = 'ETRN',
            length = 500,
        },
        {
            command = 'EXPN',
            length = 255,
        },
        {
            command = 'VRFY',
            length = 255,
        },
    },
}
```

5.17 Telnet

Given a telnet data buffer, Telnet will normalize the buffer with respect to telnet commands and option negotiation, eliminating telnet command sequences per RFC 854. It will also determine when a telnet connection is encrypted, per the use of the telnet encryption option per RFC 2946.

5.17.1 Configuring the inspector to block exploits and attacks

ayt_attack_thresh number

Detect and alert on consecutive are you there [AYT] commands beyond the threshold number specified. This addresses a few specific vulnerabilities relating to bsd-based implementations of telnet.

5.18 Wizard

Using the wizard enables port-independent configuration and the detection of malware command and control channels. If the wizard is bound to a session, it peeks at the initial payload to determine the service. For example, *GET* would indicate HTTP and *HELO* would indicate SMTP. Upon finding a match, the service bindings are reevaluated so the session can be handed off to the appropriate inspector. The wizard is still under development; if you find you need to tweak the defaults please let us know.

Additional Details:

- If the wizard and one or more service inspectors are configured w/o explicitly configuring the binder, default bindings will be generated which should work for most common cases.
- Also note that while Snort 2 bindings can only be configured in the default policy, each Snort 3 policy can contain a binder leading to an arbitrary hierarchy.
- The entire configuration can be reloaded and hot-swapped during run-time via signal or command in both Snort 2 and Snort 3. Ultimately, Snort 3 will support commands to update the binder on the fly, thus enabling incremental reloads of individual inspectors.
- Both Snort 2 and Snort 3 support server specific configurations via a hosts table (XML in Snort 2 and Lua in Snort 3). The table allows you to map network, protocol, and port to a service and policy. This table can be reloaded and hot-swapped separately from the config file.
- You can find the specifics on the binder, wizard, and hosts tables in the manual or command line like this: snort --help-module binder, etc.

6 Basic Modules

Internal modules which are not plugins are termed "basic". These include configuration for core processing.

6.1 active

What: configure responses

Type: basic

Usage: global

Configuration:

- int **active.attempts** = 0: number of TCP packets sent per response (with varying sequence numbers) { 0:20 }
- string **active.device**: use *ip* for network layer responses or *eth0* etc for link layer
- string **active.dst_mac**: use format *01:23:45:67:89:ab*
- int **active.max_responses** = 0: maximum number of responses { 0: }
- int **active.min_interval** = 255: minimum number of seconds between responses { 1:255 }

6.2 alerts

What: configure alerts

Type: basic

Usage: global

Configuration:

- bool **alerts.alert_with_interface_name** = false: include interface in alert info (fast, full, or syslog only)
- bool **alerts.default_rule_state** = true: enable or disable ips rules
- int **alerts.detection_filter_memcap** = 1048576: set available bytes of memory for detection_filters { 0: }
- int **alerts.event_filter_memcap** = 1048576: set available bytes of memory for event_filters { 0: }
- bool **alerts.log_references** = false: include rule references in alert info (full only)
- string **alerts.order** = pass drop alert log: change the order of rule action application
- int **alerts.rate_filter_memcap** = 1048576: set available bytes of memory for rate_filters { 0: }
- string **alerts.reference_net**: set the CIDR for homenet (for use with -l or -B, does NOT change $HOME_NET in IDS mode)
- bool **alerts.stateful** = false: don't alert w/o established session (note: rule action still taken)
- string **alerts.tunnel_verdicts**: let DAQ handle non-allow verdicts for gtp|teredo|6in4|4in6|4in4|6in6|gre|mpls traffic

6.3 attribute_table

What: configure hosts loading

Type: basic

Usage: global

Configuration:

- int **attribute_table.max_hosts** = 1024: maximum number of hosts in attribute table { 32:207551 }
- int **attribute_table.max_services_per_host** = 8: maximum number of services per host entry in attribute table { 1:65535 }
- int **attribute_table.max_metadata_services** = 8: maximum number of services in rule metadata { 1:256 }

6.4 classifications

What: define rule categories with priority

Type: basic

Usage: global

Configuration:

- string **classifications[].name**: name used with classtype rule option
- int **classifications[].priority** = 1: default priority for class { 0: }
- string **classifications[].text**: description of class

6.5 daq

What: configure packet acquisition interface

Type: basic

Usage: global

Configuration:

- string **daq.module_dirs[].str**: string parameter

- string **daq.input_spec**: input specification
- string **daq.module**: DAQ module to use
- string **daq.variables[].str**: string parameter
- int **daq.instances[].id**: instance ID (required) { 0: }
- string **daq.instances[].input_spec**: input specification
- string **daq.instances[].variables[].str**: string parameter
- int **daq.snaplen**: set snap length (same as -s) { 0:65535 }
- bool **daq.no_promisc** = false: whether to put DAQ device into promiscuous mode

Peg counts:

- **daq.pcaps**: total files and interfaces processed (max)
- **daq.received**: total packets received from DAQ (sum)
- **daq.analyzed**: total packets analyzed from DAQ (sum)
- **daq.dropped**: packets dropped (sum)
- **daq.filtered**: packets filtered out (sum)
- **daq.outstanding**: packets unprocessed (sum)
- **daq.injected**: active responses or replacements (sum)
- **daq.allow**: total allow verdicts (sum)
- **daq.block**: total block verdicts (sum)
- **daq.replace**: total replace verdicts (sum)
- **daq.whitelist**: total whitelist verdicts (sum)
- **daq.blacklist**: total blacklist verdicts (sum)
- **daq.ignore**: total ignore verdicts (sum)
- **daq.retry**: total retry verdicts (sum)
- **daq.internal_blacklist**: packets blacklisted internally due to lack of DAQ support (sum)
- **daq.internal_whitelist**: packets whitelisted internally due to lack of DAQ support (sum)
- **daq.skipped**: packets skipped at startup (sum)
- **daq.idle**: attempts to acquire from DAQ without available packets (sum)
- **daq.rx_bytes**: total bytes received (sum)

6.6 decode

What: general decoder rules

Type: basic

Usage: context

Rules:

- **116:450** (decode) bad IP protocol
- **116:293** (decode) two or more IP (v4 and/or v6) encapsulation layers present
- **116:459** (decode) fragment with zero length
- **116:150** (decode) loopback IP
- **116:151** (decode) same src/dst IP
- **116:449** (decode) unassigned/reserved IP protocol
- **116:472** (decode) too many protocols present
- **116:473** (decode) ether type out of range

6.7 detection

What: configure general IPS rule processing parameters

Type: basic

Usage: global

Configuration:

- int **detection.asn1** = 256: maximum decode nodes { 1: }
- int **detection.offload_limit** = 99999: minimum sizeof PDU to offload fast pattern search (defaults to disabled) { 0: }
- int **detection.offload_threads** = 0: maximum number of simultaneous offloads (defaults to disabled) { 0: }
- bool **detection.pcre_enable** = true: disable pcre pattern matching
- int **detection.pcre_match_limit** = 1500: limit pcre backtracking, -1 = max, 0 = off { -1:1000000 }
- int **detection.pcre_match_limit_recursion** = 1500: limit pcre stack consumption, -1 = max, 0 = off { -1:10000 }
- bool **detection.enable_address_anomaly_checks** = false: enable check and alerting of address anomalies
- int **detection.trace**: mask for enabling debug traces in module

Peg counts:

- **detection.analyzed**: packets sent to detection (sum)
- **detection.hard_evals**: non-fast pattern rule evaluations (sum)
- **detection.raw_searches**: fast pattern searches in raw packet data (sum)
- **detection.cooked_searches**: fast pattern searches in cooked packet data (sum)
- **detection.pkt_searches**: fast pattern searches in packet data (sum)
- **detection.alt_searches**: alt fast pattern searches in packet data (sum)

- **detection.key_searches**: fast pattern searches in key buffer (sum)
- **detection.header_searches**: fast pattern searches in header buffer (sum)
- **detection.body_searches**: fast pattern searches in body buffer (sum)
- **detection.file_searches**: fast pattern searches in file buffer (sum)
- **detection.offloads**: fast pattern searches that were offloaded (sum)
- **detection.alerts**: alerts not including IP reputation (sum)
- **detection.total_alerts**: alerts including IP reputation (sum)
- **detection.logged**: logged packets (sum)
- **detection.passed**: passed packets (sum)
- **detection.match_limit**: fast pattern matches not processed (sum)
- **detection.queue_limit**: events not queued because queue full (sum)
- **detection.log_limit**: events queued but not logged (sum)
- **detection.event_limit**: events filtered (sum)
- **detection.alert_limit**: events previously triggered on same PDU (sum)

6.8 event_filter

What: configure thresholding of events

Type: basic

Usage: context

Configuration:

- int **event_filter[].gid** = 1: rule generator ID { 0: }
- int **event_filter[].sid** = 1: rule signature ID { 0: }
- enum **event_filter[].type**: 1st count events | every count events | once after count events { limit | threshold | both }
- enum **event_filter[].track**: filter only matching source or destination addresses { by_src | by_dst }
- int **event_filter[].count** = 0: number of events in interval before tripping; -1 to disable { -1: }
- int **event_filter[].seconds** = 0: count interval { 0: }
- string **event_filter[].ip**: restrict filter to these addresses according to track

6.9 event_queue

What: configure event queue parameters

Type: basic

Usage: context

Configuration:

- int **event_queue.max_queue** = 8: maximum events to queue { 1: }
- int **event_queue.log** = 3: maximum events to log { 1: }
- enum **event_queue.order_events** = content_length: criteria for ordering incoming events { priority|content_length }
- bool **event_queue.process_all_events** = false: process just first action group or all action groups

6.10 high_availability

What: implement flow tracking high availability

Type: basic

Usage: global

Configuration:

- bool **high_availability.enable** = false: enable high availability
- bool **high_availability.daq_channel** = false: enable use of daq data plane channel
- bit_list **high_availability.ports**: side channel message port list { 65535 }
- real **high_availability.min_age** = 1.0: minimum session life before HA updates { 0.0:100.0 }
- real **high_availability.min_sync** = 1.0: minimum interval between HA updates { 0.0:100.0 }

Peg counts:

- **high_availability.packets**: total packets (sum)

6.11 host_cache

What: configure hosts

Type: basic

Usage: global

Configuration:

- int **host_cache[].size**: size of host cache

Peg counts:

- **host_cache.lru_cache_adds**: lru cache added new entry (sum)
- **host_cache.lru_cache_replaces**: lru cache replaced existing entry (sum)
- **host_cache.lru_cache_prunes**: lru cache pruned entry to make space for new entry (sum)
- **host_cache.lru_cache_find_hits**: lru cache found entry in cache (sum)
- **host_cache.lru_cache_find_misses**: lru cache did not find entry in cache (sum)
- **host_cache.lru_cache_removes**: lru cache found entry and removed it (sum)
- **host_cache.lru_cache_clears**: lru cache clear API calls (sum)

6.12 host_tracker

What: configure hosts

Type: basic

Usage: global

Configuration:

- addr **host_tracker[].IP** = 0.0.0.0/32: hosts address / cidr

- enum **host_tracker[].frag_policy**: defragmentation policy { first | linux | bsd | bsd_right | last | windows | solaris }
- enum **host_tracker[].tcp_policy**: TCP reassembly policy { first | last | linux | old_linux | bsd | macos | solaris | irix | hpux11 | hpux10 | windows | win_2003 | vista | proxy }
- string **host_tracker[].services[].name**: service identifier
- enum **host_tracker[].services[].proto** = tcp: IP protocol { tcp | udp }
- port **host_tracker[].services[].port**: port number

Peg counts:

- **host_tracker.service_adds**: host service adds (sum)
- **host_tracker.service_finds**: host service finds (sum)
- **host_tracker.service_removes**: host service removes (sum)

6.13 hosts

What: configure hosts

Type: basic

Usage: global

Configuration:

- addr **hosts[].ip** = 0.0.0.0/32: hosts address / CIDR
- enum **hosts[].frag_policy**: defragmentation policy { first | linux | bsd | bsd_right | last | windows | solaris }
- enum **hosts[].tcp_policy**: TCP reassembly policy { first | last | linux | old_linux | bsd | macos | solaris | irix | hpux11 | hpux10 | windows | win_2003 | vista | proxy }
- string **hosts[].services[].name**: service identifier
- enum **hosts[].services[].proto** = tcp: IP protocol { tcp | udp }
- port **hosts[].services[].port**: port number

6.14 inspection

What: configure basic inspection policy parameters

Type: basic

Usage: inspect

Configuration:

- int **inspection.id** = 0: correlate policy and events with other items in configuration { 0:65535 }
- string **inspection.uuid**: correlate events by uuid
- enum **inspection.mode** = inline-test: set policy mode { inline | inline-test }

6.15 ips

What: configure IPS rule processing

Type: basic

Usage: detect

Configuration:

- bool **ips.enable_builtin_rules** = false: enable events from builtin rules w/o stubs
- int **ips.id** = 0: correlate unified2 events with configuration { 0:65535 }
- string **ips.include**: legacy snort rules and includes
- enum **ips.mode**: set policy mode { tap | inline | inline-test }
- string **ips.rules**: snort rules and includes
- string **ips.uuid** = 00000000-0000-0000-0000-000000000000: IPS policy uuid

6.16 latency

What: packet and rule latency monitoring and control

Type: basic

Usage: context

Configuration:

- int **latency.packet.max_time** = 500: set timeout for packet latency thresholding (usec) { 0: }
- bool **latency.packet.fastpath** = false: fastpath expensive packets (max_time exceeded)
- enum **latency.packet.action** = none: event action if packet times out and is fastpathed { none | alert | log | alert_and_log }
- int **latency.rule.max_time** = 500: set timeout for rule evaluation (usec) { 0: }
- bool **latency.rule.suspend** = false: temporarily suspend expensive rules
- int **latency.rule.suspend_threshold** = 5: set threshold for number of timeouts before suspending a rule { 1: }
- int **latency.rule.max_suspend_time** = 30000: set max time for suspending a rule (ms, 0 means permanently disable rule) { 0: }
- enum **latency.rule.action** = none: event action for rule latency enable and suspend events { none | alert | log | alert_and_log }

Rules:

- **134:1** (latency) rule tree suspended due to latency
- **134:2** (latency) rule tree re-enabled after suspend timeout
- **134:3** (latency) packet fastpathed due to latency

Peg counts:

- **latency.total_packets**: total packets monitored (sum)
- **latency.total_usecs**: total usecs elapsed (sum)
- **latency.max_usecs**: maximum usecs elapsed (sum)
- **latency.packet_timeouts**: packets that timed out (sum)
- **latency.total_rule_evals**: total rule evals monitored (sum)
- **latency.rule_eval_timeouts**: rule evals that timed out (sum)
- **latency.rule_tree_enables**: rule tree re-enables (sum)

6.17 memory

What: memory management configuration

Type: basic

Usage: global

Configuration:

- int **memory.cap** = 0: set the per-packet-thread cap on memory (bytes, 0 to disable) { 0: }
- bool **memory.soft** = false: always succeed in allocating memory, even if above the cap
- int **memory.threshold** = 0: set the per-packet-thread threshold for preemptive cleanup actions (percent, 0 to disable) { 0: }

6.18 network

What: configure basic network parameters

Type: basic

Usage: context

Configuration:

- multi **network.checksum_drop** = none: drop if checksum is bad { all | ip | noip | tcp | notcp | udp | noudp | icmp | noicmp | none }
- multi **network.checksum_eval** = none: checksums to verify { all | ip | noip | tcp | notcp | udp | noudp | icmp | noicmp | none }
- bool **network.decode_drops** = false: enable dropping of packets by the decoder
- int **network.id** = 0: correlate unified2 events with configuration { 0:65535 }
- int **network.min_ttl** = 1: alert / normalize packets with lower TTL / hop limit (you must enable rules and / or normalization also) { 1:255 }
- int **network.new_ttl** = 1: use this value for responses and when normalizing { 1:255 }
- int **network.layers** = 40: the maximum number of protocols that Snort can correctly decode { 3:255 }
- int **network.max_ip6_extensions** = 0: the maximum number of IP6 options Snort will process for a given IPv6 layer before raising 116:456 (0 = unlimited) { 0:255 }
- int **network.max_ip_layers** = 0: the maximum number of IP layers Snort will process for a given packet before raising 116:293 (0 = unlimited) { 0:255 }

6.19 output

What: configure general output parameters

Type: basic

Usage: global

Configuration:

- bool **output.dump_chars_only** = false: turns on character dumps (same as -C)
- bool **output.dump_payload** = false: dumps application layer (same as -d)
- bool **output.dump_payload_verbose** = false: dumps raw packet starting at link layer (same as -X)

- int **output.event_trace.max_data** = 0: maximum amount of packet data to capture { 0:65535 }
- bool **output.quiet** = false: suppress non-fatal information (still show alerts, same as -q)
- string **output.logdir** = .: where to put log files (same as -l)
- bool **output.obfuscate** = false: obfuscate the logged IP addresses (same as -O)
- bool **output.obfuscate_pii** = false: mask all but the last 4 characters of credit card and social security numbers
- bool **output.show_year** = false: include year in timestamp in the alert and log files (same as -y)
- int **output.tagged_packet_limit** = 256: maximum number of packets tagged for non-packet metrics { 0: }
- bool **output.verbose** = false: be verbose (same as -v)
- bool **output.wide_hex_dump** = true: output 20 bytes per lines instead of 16 when dumping buffers

6.20 packet_tracer

What: generate debug trace messages for packets

Type: basic

Usage: global

Configuration:

- bool **packet_tracer.enable** = false: enable summary output of state that determined packet verdict
- enum **packet_tracer.output** = console: select where to send packet trace { console | file }

Commands:

- **packet_tracer.enable**(proto, src_ip, src_port, dst_ip, dst_port): enable packet tracer debugging
- **packet_tracer.disable**(): disable packet tracer

6.21 packets

What: configure basic packet handling

Type: basic

Usage: global

Configuration:

- bool **packets.address_space_agnostic** = false: determines whether DAQ address space info is used to track fragments and connections
- string **packets.bpf_file**: file with BPF to select traffic for Snort
- int **packets.limit** = 0: maximum number of packets to process before stopping (0 is unlimited) { 0: }
- int **packets.skip** = 0: number of packets to skip before before processing { 0: }
- bool **packets.vlan_agnostic** = false: determines whether VLAN info is used to track fragments and connections

6.22 process

What: configure basic process setup

Type: basic

Usage: global

Configuration:

- string **process.chroot**: set chroot directory (same as -t)
- string **process.threads[].cpuset**: pin the associated thread to this cpuset
- int **process.threads[].thread** = 0: set cpu affinity for the <cur_thread_num> thread that runs { 0: }
- bool **process.daemon** = false: fork as a daemon (same as -D)
- bool **process.dirty_pig** = false: shutdown without internal cleanup
- string **process.set_gid**: set group ID (same as -g)
- string **process.set_uid**: set user ID (same as -u)
- string **process.umask**: set process umask (same as -m)
- bool **process.utc** = false: use UTC instead of local time for timestamps

6.23 profiler

What: configure profiling of rules and/or modules

Type: basic

Usage: global

Configuration:

- bool **profiler.modules.show** = true: show module time profile stats
- int **profiler.modules.count** = 0: limit results to count items per level (0 = no limit) { 0: }
- enum **profiler.modules.sort** = total_time: sort by given field { none | checks | avg_check | total_time }
- int **profiler.modules.max_depth** = -1: limit depth to max_depth (-1 = no limit) { -1: }
- bool **profiler.memory.show** = true: show module memory profile stats
- int **profiler.memory.count** = 0: limit results to count items per level (0 = no limit) { 0: }
- enum **profiler.memory.sort** = total_used: sort by given field { none | allocations | total_used | avg_allocation }
- int **profiler.memory.max_depth** = -1: limit depth to max_depth (-1 = no limit) { -1: }
- bool **profiler.rules.show** = true: show rule time profile stats
- int **profiler.rules.count** = 0: print results to given level (0 = all) { 0: }
- enum **profiler.rules.sort** = total_time: sort by given field { none | checks | avg_check | total_time | matches | no_matches | avg_match | avg_no_match }

6.24 rate_filter

What: configure rate filters (which change rule actions)

Type: basic

Usage: detect

Configuration:

- int **rate_filter[].gid** = 1: rule generator ID { 0: }
- int **rate_filter[].sid** = 1: rule signature ID { 0: }
- enum **rate_filter[].track** = by_src: filter only matching source or destination addresses { by_src | by_dst | by_rule }
- int **rate_filter[].count** = 1: number of events in interval before tripping { 0: }
- int **rate_filter[].seconds** = 1: count interval { 0: }
- enum **rate_filter[].new_action** = alert: take this action on future hits until timeout { log | pass | alert | drop | block | reset }
- int **rate_filter[].timeout** = 1: count interval { 0: }
- string **rate_filter[].apply_to**: restrict filter to these addresses according to track

6.25 references

What: define reference systems used in rules

Type: basic

Usage: global

Configuration:

- string **references[].name**: name used with reference rule option
- string **references[].url**: where this reference is defined

6.26 rule_state

What: enable/disable specific IPS rules

Type: basic

Usage: detect

Configuration:

- int **rule_state.gid** = 0: rule generator ID { 0: }
- int **rule_state.sid** = 0: rule signature ID { 0: }
- bool **rule_state.enable** = true: enable or disable rule in all policies

6.27 search_engine

What: configure fast pattern matcher

Type: basic

Usage: global

Configuration:

- int **search_engine.bleedover_port_limit** = 1024: maximum ports in rule before demotion to any-any port group { 1: }
- bool **search_engine.bleedover_warnings_enabled** = false: print warning if a rule is demoted to any-any port group
- bool **search_engine.enable_single_rule_group** = false: put all rules into one group
- bool **search_engine.debug** = false: print verbose fast pattern info
- bool **search_engine.debug_print_nocontent_rule_tests** = false: print rule group info during packet evaluation
- bool **search_engine.debug_print_rule_group_build_details** = false: print rule group info during compilation
- bool **search_engine.debug_print_rule_groups_uncompiled** = false: prints uncompiled rule group information
- bool **search_engine.debug_print_rule_groups_compiled** = false: prints compiled rule group information
- int **search_engine.max_pattern_len** = 0: truncate patterns when compiling into state machine (0 means no maximum) { 0: }
- int **search_engine.max_queue_events** = 5: maximum number of matching fast pattern states to queue per packet { 2:100 }
- bool **search_engine.detect_raw_tcp** = false: detect on TCP payload before reassembly
- dynamic **search_engine.search_method** = ac_bnfa: set fast pattern algorithm - choose available search engine { ac_banded | ac_bnfa | ac_full | ac_sparse | ac_sparse_bands | ac_std | hyperscan | lowmem }
- bool **search_engine.search_optimize** = true: tweak state machine construction for better performance
- bool **search_engine.show_fast_patterns** = false: print fast pattern info for each rule
- bool **search_engine.split_any_any** = true: evaluate any-any rules separately to save memory

Peg counts:

- **search_engine.max_queued**: maximum fast pattern matches queued for further evaluation (sum)
- **search_engine.total_flushed**: fast pattern matches discarded due to overflow (sum)
- **search_engine.total_inserts**: total fast pattern hits (sum)
- **search_engine.total_unique**: total unique fast pattern hits (sum)
- **search_engine.non_qualified_events**: total non-qualified events (sum)
- **search_engine.qualified_events**: total qualified events (sum)
- **search_engine.searched_bytes**: total bytes searched (sum)

6.28 side_channel

What: implement the side-channel asynchronous messaging subsystem

Type: basic

Usage: global

Configuration:

- bit_list **side_channel.ports**: side channel message port list { 65535 }
- string **side_channel.connectors[].connector**: connector handle
- string **side_channel.connector**: connector handle

Peg counts:

- **side_channel.packets**: total packets (sum)

6.29 snort

What: command line configuration and shell commands

Type: basic

Usage: global

Configuration:

- string **snort.-?**: <option prefix> output matching command line option quick help (same as --help-options) { (optional) }
- string **snort.-A**: <mode> set alert mode: none, cmg, or alert_*
- addr **snort.-B** = 255.255.255.255/32: <mask> obfuscated IP addresses in alerts and packet dumps using CIDR mask
- implied **snort.-C**: print out payloads with character data only (no hex)
- string **snort.-c**: <conf> use this configuration
- implied **snort.-D**: run Snort in background (daemon) mode
- implied **snort.-d**: dump the Application Layer
- implied **snort.-e**: display the second layer header info
- implied **snort.-f**: turn off fflush() calls after binary log writes
- int **snort.-G**: <0xid> (same as --logid) { 0:65535 }
- string **snort.-g**: <gname> run snort gid as <gname> group (or gid) after initialization
- implied **snort.-H**: make hash tables deterministic
- string **snort.-i**: <iface>... list of interfaces
- port **snort.-j**: <port> to listen for Telnet connections
- enum **snort.-k** = all: <mode> checksum mode; default is all { all|noip|notcp|noudp|noicmp|none }
- string **snort.-L**: <mode> logging mode (none, dump, pcap, or log_*)
- string **snort.-l**: <logdir> log to this directory instead of current directory
- implied **snort.-M**: log messages to syslog (not alerts)

- int **snort.-m**: <umask> set umask = <umask> { 0: }
- int **snort.-n**: <count> stop after count packets { 0: }
- implied **snort.-O**: obfuscate the logged IP addresses
- implied **snort.-Q**: enable inline mode operation
- implied **snort.-q**: quiet mode - Don't show banner and status report
- string **snort.-R**: <rules> include this rules file in the default policy
- string **snort.-r**: <pcap>... (same as --pcap-list)
- string **snort.-S**: <x=v> set config variable x equal to value v
- int **snort.-s** = 1514: <snap> (same as --snaplen); default is 1514 { 68:65535 }
- implied **snort.-T**: test and report on the current Snort configuration
- string **snort.-t**: <dir> chroots process to <dir> after initialization
- implied **snort.-U**: use UTC for timestamps
- string **snort.-u**: <uname> run snort as <uname> or <uid> after initialization
- implied **snort.-V**: (same as --version)
- implied **snort.-v**: be verbose
- implied **snort.-W**: lists available interfaces
- implied **snort.-X**: dump the raw packet data starting at the link layer
- implied **snort.-x**: same as --pedantic
- implied **snort.-y**: include year in timestamp in the alert and log files
- int **snort.-z** = 1: <count> maximum number of packet threads (same as --max-packet-threads); 0 gets the number of CPU cores reported by the system; default is 1 { 0: }
- implied **snort.--alert-before-pass**: process alert, drop, sdrop, or reject before pass; default is pass before alert, drop,...
- string **snort.--bpf**: <filter options> are standard BPF options, as seen in TCPDump
- string **snort.--c2x**: output hex for given char (see also --x2c)
- string **snort.--control-socket**: <file> to create unix socket
- implied **snort.--create-pidfile**: create PID file, even when not in Daemon mode
- string **snort.--daq**: <type> select packet acquisition module (default is pcap)
- string **snort.--daq-dir**: <dir> tell snort where to find desired DAQ
- implied **snort.--daq-list**: list packet acquisition modules available in optional dir, default is static modules only
- string **snort.--daq-var**: <name=value> specify extra DAQ configuration variable
- implied **snort.--dirty-pig**: don't flush packets on shutdown
- string **snort.--dump-builtin-rules**: [<module prefix>] output stub rules for selected modules { (optional) }
- implied **snort.--dump-dynamic-rules**: output stub rules for all loaded rules libraries
- string **snort.--dump-defaults**: [<module prefix>] output module defaults in Lua format { (optional) }
- implied **snort.--dump-version**: output the version, the whole version, and only the version

- implied **snort.--enable-inline-test**: enable Inline-Test Mode Operation
- implied **snort.--gen-msg-map**: dump builtin rules in gen-msg.map format for use by other tools
- implied **snort.--help**: list command line options
- string **snort.--help-commands**: [<module prefix>] output matching commands { (optional) }
- string **snort.--help-config**: [<module prefix>] output matching config options { (optional) }
- string **snort.--help-counts**: [<module prefix>] output matching peg counts { (optional) }
- string **snort.--help-module**: <module> output description of given module
- implied **snort.--help-modules**: list all available modules with brief help
- string **snort.--help-options**: [<option prefix>] output matching command line option quick help (same as -?) { (optional) }
- implied **snort.--help-plugins**: list all available plugins with brief help
- implied **snort.--help-signals**: dump available control signals
- int **snort.--id-offset** = 0: offset to add to instance IDs when logging to files { 0:65535 }
- implied **snort.--id-subdir**: create/use instance subdirectories in logdir instead of instance filename prefix
- implied **snort.--id-zero**: use id prefix / subdirectory even with one packet thread
- implied **snort.--list-buffers**: output available inspection buffers
- string **snort.--list-builtin**: [<module prefix>] output matching builtin rules { (optional) }
- string **snort.--list-gids**: [<module prefix>] output matching generators { (optional) }
- string **snort.--list-modules**: [<module type>] list all known modules of given type { (optional) }
- implied **snort.--list-plugins**: list all known plugins
- string **snort.--lua**: <chunk> extend/override conf with chunk; may be repeated
- int **snort.--logid**: <0xid> log Identifier to uniquely id events for multiple snorts (same as -G) { 0:65535 }
- implied **snort.--markup**: output help in asciidoc compatible format
- int **snort.--max-packet-threads** = 1: <count> configure maximum number of packet threads (same as -z) { 0: }
- implied **snort.--mem-check**: like -T but also compile search engines
- implied **snort.--nostamps**: don't include timestamps in log file names
- implied **snort.--nolock-pidfile**: do not try to lock Snort PID file
- implied **snort.--pause**: wait for resume/quit command before processing packets/terminating
- implied **snort.--parsing-follows-files**: parse relative paths from the perspective of the current configuration file
- string **snort.--pcap-file**: <file> file that contains a list of pcaps to read - read mode is implied
- string **snort.--pcap-list**: <list> a space separated list of pcaps to read - read mode is implied
- string **snort.--pcap-dir**: <dir> a directory to recurse to look for pcaps - read mode is implied
- string **snort.--pcap-filter**: <filter> filter to apply when getting pcaps from file or directory
- int **snort.--pcap-loop**: <count> read all pcaps <count> times; 0 will read until Snort is terminated { -1: }
- implied **snort.--pcap-no-filter**: reset to use no filter when getting pcaps from file or directory
- implied **snort.--pcap-reload**: if reading multiple pcaps, reload snort config between pcaps

- implied **snort.--pcap-show**: print a line saying what pcap is currently being read
- implied **snort.--pedantic**: warnings are fatal
- string **snort.--plugin-path**: <path> where to find plugins
- implied **snort.--process-all-events**: process all action groups
- string **snort.--rule**: <rules> to be added to configuration; may be repeated
- implied **snort.--rule-to-hex**: output so rule header to stdout for text rule on stdin
- string **snort.--rule-to-text** = [SnortFoo]: output plain so rule header to stdout for text rule on stdin { 16 }
- string **snort.--run-prefix**: <pfx> prepend this to each output file
- string **snort.--script-path**: <path> to a luajit script or directory containing luajit scripts
- implied **snort.--shell**: enable the interactive command line
- implied **snort.--piglet**: enable piglet test harness mode
- implied **snort.--show-plugins**: list module and plugin versions
- int **snort.--skip**: <n> skip 1st n packets { 0: }
- int **snort.--snaplen** = 1514: <snap> set snaplen of packet (same as -s) { 68:65535 }
- implied **snort.--stdin-rules**: read rules from stdin until EOF or a line starting with END is read
- implied **snort.--talos**: enable Talos inline rule test mode (same as --tweaks talos -Q -q)
- implied **snort.--treat-drop-as-alert**: converts drop, sdrop, and reject rules into alert rules during startup
- implied **snort.--treat-drop-as-ignore**: use drop, sdrop, and reject rules to ignore session traffic when not inline
- string **snort.--tweaks**: tune configuration
- string **snort.--catch-test**: comma separated list of cat unit test tags or *all*
- implied **snort.--version**: show version number (same as -V)
- implied **snort.--warn-all**: enable all warnings
- implied **snort.--warn-conf**: warn about configuration issues
- implied **snort.--warn-daq**: warn about DAQ issues, usually related to mode
- implied **snort.--warn-flowbits**: warn about flowbits that are checked but not set and vice-versa
- implied **snort.--warn-hosts**: warn about host table issues
- implied **snort.--warn-plugins**: warn about issues that prevent plugins from loading
- implied **snort.--warn-rules**: warn about duplicate rules and rule parsing issues
- implied **snort.--warn-scripts**: warn about issues discovered while processing Lua scripts
- implied **snort.--warn-symbols**: warn about unknown symbols in your Lua config
- implied **snort.--warn-vars**: warn about variable definition and usage issues
- int **snort.--x2c**: output ASCII char for given hex (see also --c2x)
- string **snort.--x2s**: output ASCII string for given byte code (see also --x2c)
- implied **snort.--trace**: turn on main loop debug trace
- int **snort.trace**: mask for enabling debug traces in module

Commands:

- **snort.show_plugins**(): show available plugins
- **snort.delete_inspector**(inspector): delete an inspector from the default policy
- **snort.dump_stats**(): show summary statistics
- **snort.rotate_stats**(): roll perfmonitor log files
- **snort.reload_config**(filename): load new configuration
- **snort.reload_policy**(filename): reload part or all of the default policy
- **snort.reload_module**(module): reload module
- **snort.reload_daq**(): reload daq module
- **snort.reload_hosts**(filename): load a new hosts table
- **snort.pause**(): suspend packet processing
- **snort.resume**(): continue packet processing
- **snort.detach**(): exit shell w/o shutdown
- **snort.quit**(): shutdown and dump-stats
- **snort.help**(): this output

Peg counts:

- **snort.local_commands**: total local commands processed (sum)
- **snort.remote_commands**: total remote commands processed (sum)
- **snort.signals**: total signals processed (sum)
- **snort.conf_reloads**: number of times configuration was reloaded (sum)
- **snort.policy_reloads**: number of times policies were reloaded (sum)
- **snort.inspector_deletions**: number of times inspectors were deleted (sum)
- **snort.daq_reloads**: number of times daq configuration was reloaded (sum)
- **snort.attribute_table_reloads**: number of times hosts table was reloaded (sum)
- **snort.attribute_table_hosts**: total number of hosts in table (sum)

6.30 suppress

What: configure event suppressions

Type: basic

Usage: detect

Configuration:

- int **suppress[].gid** = 0: rule generator ID { 0: }
- int **suppress[].sid** = 0: rule signature ID { 0: }
- enum **suppress[].track**: suppress only matching source or destination addresses { by_src | by_dst }
- string **suppress[].ip**: restrict suppression to these addresses according to track

7 Codec Modules

Codec is short for coder / decoder. These modules are used for basic protocol decoding, anomaly detection, and construction of active responses.

7.1 arp

What: support for address resolution protocol

Type: codec

Usage: context

Rules:

- **116:109** (arp) truncated ARP

7.2 auth

What: support for IP authentication header

Type: codec

Usage: context

Rules:

- **116:465** (auth) truncated authentication header
- **116:466** (auth) bad authentication header length

7.3 ciscometadata

What: support for cisco metadata

Type: codec

Usage: context

Rules:

- **116:468** (ciscometadata) truncated Cisco Metadata header
- **116:469** (ciscometadata) invalid Cisco Metadata option length
- **116:470** (ciscometadata) invalid Cisco Metadata option type
- **116:471** (ciscometadata) invalid Cisco Metadata SGT

7.4 eapol

What: support for extensible authentication protocol over LAN

Type: codec

Usage: context

Rules:

- **116:110** (eapol) truncated EAP header
- **116:111** (eapol) EAP key truncated
- **116:112** (eapol) EAP header truncated

7.5 erspan2

What: support for encapsulated remote switched port analyzer - type 2

Type: codec

Usage: context

Rules:

- **116:462** (erspan2) ERSpan header version mismatch
- **116:463** (erspan2) captured length < ERSpan type2 header length

7.6 erspan3

What: support for encapsulated remote switched port analyzer - type 3

Type: codec

Usage: context

Rules:

- **116:464** (erspan3) captured < ERSpan type3 header length

7.7 esp

What: support for encapsulating security payload

Type: codec

Usage: context

Configuration:

- bool **esp.decode_esp** = false: enable for inspection of esp traffic that has authentication but not encryption

Rules:

- **116:294** (esp) truncated encapsulated security payload header

7.8 eth

What: support for ethernet protocol (DLT 1) (DLT 51)

Type: codec

Usage: context

Rules:

- **116:424** (eth) truncated ethernet header

7.9 fabricpath

What: support for fabricpath

Type: codec

Usage: context

Rules:

- **116:467** (fabricpath) truncated FabricPath header

7.10 gre

What: support for generic routing encapsulation

Type: codec

Usage: context

Rules:

- **116:160** (gre) GRE header length > payload length
- **116:161** (gre) multiple encapsulations in packet
- **116:162** (gre) invalid GRE version
- **116:163** (gre) invalid GRE header
- **116:164** (gre) invalid GRE v.1 PPTP header
- **116:165** (gre) GRE trans header length > payload length

7.11 gtp

What: support for general-packet-radio-service tunneling protocol

Type: codec

Usage: context

Rules:

- **116:297** (gtp) two or more GTP encapsulation layers present
- **116:298** (gtp) GTP header length is invalid

7.12 icmp4

What: support for Internet control message protocol v4

Type: codec

Usage: context

Rules:

- **116:105** (icmp4) ICMP header truncated
- **116:106** (icmp4) ICMP timestamp header truncated
- **116:107** (icmp4) ICMP address header truncated
- **116:250** (icmp4) ICMP original IP header truncated
- **116:251** (icmp4) ICMP version and original IP header versions differ
- **116:252** (icmp4) ICMP original datagram length < original IP header length
- **116:253** (icmp4) ICMP original IP payload < 64 bits
- **116:254** (icmp4) ICMP original IP payload > 576 bytes
- **116:255** (icmp4) ICMP original IP fragmented and offset not 0
- **116:415** (icmp4) ICMP4 packet to multicast dest address

- **116:416** (icmp4) ICMP4 packet to broadcast dest address
- **116:418** (icmp4) ICMP4 type other
- **116:434** (icmp4) ICMP ping Nmap
- **116:435** (icmp4) ICMP icmpenum v1.1.1
- **116:436** (icmp4) ICMP redirect host
- **116:437** (icmp4) ICMP redirect net
- **116:438** (icmp4) ICMP traceroute ipopts
- **116:439** (icmp4) ICMP source quench
- **116:440** (icmp4) broadscan smurf scanner
- **116:441** (icmp4) ICMP destination unreachable communication administratively prohibited
- **116:442** (icmp4) ICMP destination unreachable communication with destination host is administratively prohibited
- **116:443** (icmp4) ICMP destination unreachable communication with destination network is administratively prohibited
- **116:451** (icmp4) ICMP path MTU denial of service attempt
- **116:452** (icmp4) Linux ICMP header DOS attempt
- **116:426** (icmp4) truncated ICMP4 header

Peg counts:

- **icmp4.bad_checksum**: non-zero icmp checksums (sum)

7.13 icmp6

What: support for Internet control message protocol v6

Type: codec

Usage: context

Rules:

- **116:427** (icmp6) truncated ICMPv6 header
- **116:431** (icmp6) ICMPv6 type not decoded
- **116:432** (icmp6) ICMPv6 packet to multicast address
- **116:285** (icmp6) ICMPv6 packet of type 2 (message too big) with MTU field < 1280
- **116:286** (icmp6) ICMPv6 packet of type 1 (destination unreachable) with non-RFC 2463 code
- **116:287** (icmp6) ICMPv6 router solicitation packet with a code not equal to 0
- **116:288** (icmp6) ICMPv6 router advertisement packet with a code not equal to 0
- **116:289** (icmp6) ICMPv6 router solicitation packet with the reserved field not equal to 0
- **116:290** (icmp6) ICMPv6 router advertisement packet with the reachable time field set > 1 hour
- **116:457** (icmp6) ICMPv6 packet of type 1 (destination unreachable) with non-RFC 4443 code
- **116:460** (icmp6) ICMPv6 node info query/response packet with a code greater than 2
- **116:474** (icmp6) ICMPv6 not encapsulated in IPv6

Peg counts:

- **icmp6.bad_icmp6_checksum**: nonzero icmp6 checksums (sum)

7.14 igmp

What: support for Internet group management protocol

Type: codec

Usage: context

Rules:

- **116:455** (igmp) DOS IGMP IP options validation attempt

7.15 ipv4

What: support for Internet protocol v4 (DLT 228)

Type: codec

Usage: context

Rules:

- **116:1** (ipv4) not IPv4 datagram
- **116:2** (ipv4) IPv4 header length < minimum
- **116:3** (ipv4) IPv4 datagram length < header field
- **116:4** (ipv4) IPv4 options found with bad lengths
- **116:5** (ipv4) truncated IPv4 options
- **116:6** (ipv4) IPv4 datagram length > captured length
- **116:404** (ipv4) IPv4 packet with zero TTL
- **116:405** (ipv4) IPv4 packet with bad frag bits (both MF and DF set)
- **116:407** (ipv4) IPv4 packet frag offset + length exceed maximum
- **116:408** (ipv4) IPv4 packet from *current net* source address
- **116:409** (ipv4) IPv4 packet to *current net* dest address
- **116:410** (ipv4) IPv4 packet from multicast source address
- **116:411** (ipv4) IPv4 packet from reserved source address
- **116:412** (ipv4) IPv4 packet to reserved dest address
- **116:413** (ipv4) IPv4 packet from broadcast source address
- **116:414** (ipv4) IPv4 packet to broadcast dest address
- **116:428** (ipv4) IPv4 packet below TTL limit
- **116:430** (ipv4) IPv4 packet both DF and offset set
- **116:448** (ipv4) IPv4 reserved bit set
- **116:444** (ipv4) IPv4 option set
- **116:425** (ipv4) truncated IPv4 header

Peg counts:

- **ipv4.bad_checksum**: nonzero ip checksums (sum)

7.16 ipv6

What: support for Internet protocol v6 (DLT 229)

Type: codec

Usage: context

Rules:

- **116:270** (ipv6) IPv6 packet below TTL limit
- **116:271** (ipv6) IPv6 header claims to not be IPv6
- **116:272** (ipv6) IPv6 truncated extension header
- **116:273** (ipv6) IPv6 truncated header
- **116:274** (ipv6) IPv6 datagram length < header field
- **116:275** (ipv6) IPv6 datagram length > captured length
- **116:276** (ipv6) IPv6 packet with destination address ::0
- **116:277** (ipv6) IPv6 packet with multicast source address
- **116:278** (ipv6) IPv6 packet with reserved multicast destination address
- **116:279** (ipv6) IPv6 header includes an undefined option type
- **116:280** (ipv6) IPv6 address includes an unassigned multicast scope value
- **116:281** (ipv6) IPv6 header includes an invalid value for the *next header* field
- **116:282** (ipv6) IPv6 header includes a routing extension header followed by a hop-by-hop header
- **116:283** (ipv6) IPv6 header includes two routing extension headers
- **116:292** (ipv6) IPv6 header has destination options followed by a routing header
- **116:291** (ipv6) IPV6 tunneled over IPv4, IPv6 header truncated, possible Linux kernel attack
- **116:295** (ipv6) IPv6 header includes an option which is too big for the containing header
- **116:296** (ipv6) IPv6 packet includes out-of-order extension headers
- **116:429** (ipv6) IPv6 packet has zero hop limit
- **116:453** (ipv6) ISATAP-addressed IPv6 traffic spoofing attempt
- **116:458** (ipv6) bogus fragmentation packet, possible BSD attack
- **116:461** (ipv6) IPv6 routing type 0 extension header
- **116:456** (ipv6) too many IPv6 extension headers
- **116:475** (ipv6) IPv6 mobility header includes an invalid value for the *payload protocol* field

7.17 llc

What: support for logical link control

Type: codec

Usage: context

Rules:

- **116:131** (llc) bad LLC header
- **116:132** (llc) bad extra LLC info

7.18 mpls

What: support for multiprotocol label switching

Type: codec

Usage: context

Configuration:

- bool **mpls.enable_mpls_multicast** = false: enables support for MPLS multicast
- bool **mpls.enable_mpls_overlapping_ip** = false: enable if private network addresses overlap and must be differentiated by MPLS label(s)
- int **mpls.max_mpls_stack_depth** = -1: set MPLS stack depth { -1: }
- enum **mpls.mpls_payload_type** = ip4: set encapsulated payload type { eth | ip4 | ip6 }

Rules:

- **116:170** (mpls) bad MPLS frame
- **116:171** (mpls) MPLS label 0 appears in non-bottom header
- **116:172** (mpls) MPLS label 1 appears in bottom header
- **116:173** (mpls) MPLS label 2 appears in non-bottom header
- **116:174** (mpls) MPLS label 3 appears in header
- **116:175** (mpls) MPLS label 4, 5,.. or 15 appears in header
- **116:176** (mpls) too many MPLS headers

Peg counts:

- **mpls.total_packets**: total mpls labeled packets processed (sum)
- **mpls.total_bytes**: total mpls labeled bytes processed (sum)

7.19 pbb

What: support for 802.1ah protocol

Type: codec

Usage: context

Rules:

- **116:424** (pbb) truncated ethernet header

7.20 pgm

What: support for pragmatic general multicast

Type: codec

Usage: context

Rules:

- **116:454** (pgm) PGM nak list overflow attempt

7.21 pppoe

What: support for point-to-point protocol over ethernet

Type: codec

Usage: context

Rules:

- **116:120** (pppoe) bad PPPOE frame detected

7.22 tcp

What: support for transmission control protocol

Type: codec

Usage: context

Rules:

- **116:45** (tcp) TCP packet length is smaller than 20 bytes
- **116:46** (tcp) TCP data offset is less than 5
- **116:47** (tcp) TCP header length exceeds packet length
- **116:54** (tcp) TCP options found with bad lengths
- **116:55** (tcp) truncated TCP options
- **116:56** (tcp) T/TCP detected
- **116:57** (tcp) obsolete TCP options found
- **116:58** (tcp) experimental TCP options found
- **116:59** (tcp) TCP window scale option found with length > 14
- **116:400** (tcp) XMAS attack detected
- **116:401** (tcp) Nmap XMAS attack detected
- **116:419** (tcp) TCP urgent pointer exceeds payload length or no payload
- **116:420** (tcp) TCP SYN with FIN
- **116:421** (tcp) TCP SYN with RST
- **116:422** (tcp) TCP PDU missing ack for established session
- **116:423** (tcp) TCP has no SYN, ACK, or RST
- **116:433** (tcp) DDOS shaft SYN flood
- **116:446** (tcp) TCP port 0 traffic
- **116:402** (tcp) DOS NAPTHA vulnerability detected
- **116:403** (tcp) SYN to multicast address

Peg counts:

- **tcp.bad_tcp4_checksum**: nonzero tcp over ip checksums (sum)
- **tcp.bad_tcp6_checksum**: nonzero tcp over ipv6 checksums (sum)

7.23 token_ring

What: support for token ring decoding

Type: codec

Usage: context

Rules:

- **116:140** (token_ring) bad Token Ring header
- **116:141** (token_ring) bad Token Ring ETHLLC header
- **116:142** (token_ring) bad Token Ring MRLEN header
- **116:143** (token_ring) bad Token Ring MR header

7.24 udp

What: support for user datagram protocol

Type: codec

Usage: context

Configuration:

- bool **udp.deep_teredo_inspection** = false: look for Teredo on all UDP ports (default is only 3544)
- bool **udp.enable_gtp** = false: decode GTP encapsulations
- bit_list **udp.gtp_ports** = 2152 3386: set GTP ports { 65535 }

Rules:

- **116:95** (udp) truncated UDP header
- **116:96** (udp) invalid UDP header, length field < 8
- **116:97** (udp) short UDP packet, length field > payload length
- **116:98** (udp) long UDP packet, length field < payload length
- **116:406** (udp) invalid IPv6 UDP packet, checksum zero
- **116:445** (udp) large UDP packet (> 4000 bytes)
- **116:447** (udp) UDP port 0 traffic

Peg counts:

- **udp.bad_udp4_checksum**: nonzero udp over ipv4 checksums (sum)
- **udp.bad_udp6_checksum**: nonzero udp over ipv6 checksums (sum)

7.25 vlan

What: support for local area network

Type: codec

Usage: context

Rules:

- **116:130** (vlan) bad VLAN frame

7.26 wlan

What: support for wireless local area network protocol (DLT 105)

Type: codec

Usage: context

Rules:

- **116:133** (wlan) bad 802.11 LLC header
- **116:134** (wlan) bad 802.11 extra LLC info

8 Connector Modules

Connectors support High Availability communication links.

8.1 file_connector

What: implement the file based connector

Type: connector

Usage: global

Configuration:

- string **file_connector.connector**: connector name
- string **file_connector.name**: channel name
- enum **file_connector.format**: file format { binary | text }
- enum **file_connector.direction**: usage { receive | transmit | duplex }

Peg counts:

- **file_connector.messages**: total messages (sum)

8.2 tcp_connector

What: implement the tcp stream connector

Type: connector

Usage: global

Configuration:

- string **tcp_connector.connector**: connector name
- string **tcp_connector.address**: address
- port **tcp_connector.base_port**: base port number
- enum **tcp_connector.setup**: stream establishment { call | answer }

Peg counts:

- **tcp_connector.messages**: total messages (sum)

9 Inspector Modules

These modules perform a variety of functions, including analysis of protocols beyond basic decoding.

9.1 appid

What: application and service identification

Type: inspector

Usage: context

Configuration:

- int **appid.first_decrypted_packet_debug** = 0: the first packet of an already decrypted SSL flow (debug single session only) { 0: }
- int **appid.memcap** = 0: disregard - not implemented { 0: }
- bool **appid.log_stats** = false: enable logging of appid statistics
- int **appid.app_stats_period** = 300: time period for collecting and logging appid statistics { 0: }
- int **appid.app_stats_rollover_size** = 20971520: max file size for appid stats before rolling over the log file { 0: }
- int **appid.app_stats_rollover_time** = 86400: max time period for collection appid stats before rolling over the log file { 0: }
- string **appid.app_detector_dir**: directory to load appid detectors from
- int **appid.instance_id** = 0: instance id - ignored { 0: }
- bool **appid.debug** = false: enable appid debug logging
- bool **appid.dump_ports** = false: enable dump of appid port information
- string **appid.tp_appid_path**: path to third party appid dynamic library
- string **appid.tp_appid_config**: path to third party appid configuration file
- bool **appid.log_all_sessions** = false: enable logging of all appid sessions
- int **appid.trace**: mask for enabling debug traces in module

Commands:

- **appid.enable_debug**(proto, src_ip, src_port, dst_ip, dst_port): enable appid debugging
- **appid.disable_debug**(): disable appid debugging

Peg counts:

- **appid.packets**: count of packets received (sum)
- **appid.processed_packets**: count of packets processed (sum)
- **appid.ignored_packets**: count of packets ignored (sum)
- **appid.total_sessions**: count of sessions created (sum)
- **appid.appid_unknown**: count of sessions where appid could not be determined (sum)

9.2 arp_spoof

What: detect ARP attacks and anomalies

Type: inspector

Usage: inspect

Configuration:

- ip4 **arp_spoof.hosts[].ip**: host ip address
- mac **arp_spoof.hosts[].mac**: host mac address

Rules:

- **112:1** (arp_spoof) unicast ARP request
- **112:2** (arp_spoof) ethernet/ARP mismatch request for source
- **112:3** (arp_spoof) ethernet/ARP mismatch request for destination
- **112:4** (arp_spoof) attempted ARP cache overwrite attack

Peg counts:

- **arp_spoof.packets**: total packets (sum)

9.3 back_orifice

What: back orifice detection

Type: inspector

Usage: inspect

Rules:

- **105:1** (back_orifice) BO traffic detected
- **105:2** (back_orifice) BO client traffic detected
- **105:3** (back_orifice) BO server traffic detected
- **105:4** (back_orifice) BO Snort buffer attack

Peg counts:

- **back_orifice.packets**: total packets (sum)

9.4 binder

What: configure processing based on CIDRs, ports, services, etc.

Type: inspector

Usage: inspect

Configuration:

- int **binder[].when.ips_policy_id** = 0: unique ID for selection of this config by external logic { 0: }

- bit_list **binder[].when.ifaces**: list of interface indices { 255 }
- bit_list **binder[].when.vlans**: list of VLAN IDs { 4095 }
- addr_list **binder[].when.nets**: list of networks
- addr_list **binder[].when.src_nets**: list of source networks
- addr_list **binder[].when.dst_nets**: list of destination networks
- enum **binder[].when.proto**: protocol { any | ip | icmp | tcp | udp | user | file }
- bit_list **binder[].when.ports**: list of ports { 65535 }
- bit_list **binder[].when.src_ports**: list of source ports { 65535 }
- bit_list **binder[].when.dst_ports**: list of destination ports { 65535 }
- int **binder[].when.src_zone**: source zone { 0:2147483647 }
- int **binder[].when.dst_zone**: destination zone { 0:2147483647 }
- enum **binder[].when.role** = any: use the given configuration on one or any end of a session { client | server | any }
- string **binder[].when.service**: override default configuration
- enum **binder[].use.action** = inspect: what to do with matching traffic { reset | block | allow | inspect }
- string **binder[].use.file**: use configuration in given file
- string **binder[].use.inspection_policy**: use inspection policy from given file
- string **binder[].use.ips_policy**: use ips policy from given file
- string **binder[].use.network_policy**: use network policy from given file
- string **binder[].use.service**: override automatic service identification
- string **binder[].use.type**: select module for binding
- string **binder[].use.name**: symbol name (defaults to type)

Peg counts:

- **binder.packets**: initial bindings (sum)
- **binder.resets**: reset bindings (sum)
- **binder.blocks**: block bindings (sum)
- **binder.allows**: allow bindings (sum)
- **binder.inspects**: inspect bindings (sum)

9.5 data_log

What: log selected published data to data.log

Type: inspector

Usage: inspect

Configuration:

- select **data_log.key** = http_request_header_event : name of the event to log { http_request_header_event | http_response_header_event }
- int **data_log.limit** = 0: set maximum size in MB before rollover (0 is unlimited) { 0: }

Peg counts:

- **data_log.packets**: total packets (sum)

9.6 dce_http_proxy

What: dce over http inspection - client to/from proxy

Type: inspector

Usage: inspect

Peg counts:

- **dce_http_proxy.http_proxy_sessions**: successful http proxy sessions (sum)
- **dce_http_proxy.http_proxy_session_failures**: failed http proxy sessions (sum)

9.7 dce_http_server

What: dce over http inspection - proxy to/from server

Type: inspector

Usage: inspect

Peg counts:

- **dce_http_server.http_server_sessions**: successful http server sessions (sum)
- **dce_http_server.http_server_session_failures**: failed http server sessions (sum)

9.8 dce_smb

What: dce over smb inspection

Type: inspector

Usage: inspect

Configuration:

- bool **dce_smb.disable_defrag** = false: Disable DCE/RPC defragmentation
- int **dce_smb.max_frag_len** = 65535: Maximum fragment size for defragmentation { 1514:65535 }
- int **dce_smb.reassemble_threshold** = 0: Minimum bytes received before performing reassembly { 0:65535 }
- enum **dce_smb.smb_fingerprint_policy** = none: Target based SMB policy to use { none | client | server | both }
- enum **dce_smb.policy** = WinXP: Target based policy to use { Win2000 | WinXP | WinVista | Win2003 | Win2008 | Win7 | Samba | Samba-3.0.37 | Samba-3.0.22 | Samba-3.0.20 }
- int **dce_smb.smb_max_chain** = 3: SMB max chain size { 0:255 }
- int **dce_smb.smb_max_compound** = 3: SMB max compound size { 0:255 }
- multi **dce_smb.valid_smb_versions** = all: Valid SMB versions { v1 | v2 | all }
- enum **dce_smb.smb_file_inspection** = off: SMB file inspection { off | on | only }
- int **dce_smb.smb_file_depth** = 16384: SMB file depth for file data { -1: }
- string **dce_smb.smb_invalid_shares**: SMB shares to alert on
- bool **dce_smb.smb_legacy_mode** = false: inspect only SMBv1
- int **dce_smb.trace**: mask for enabling debug traces in module

Rules:

- **133:2** (dce_smb) SMB - bad NetBIOS session service session type
- **133:3** (dce_smb) SMB - bad SMB message type
- **133:4** (dce_smb) SMB - bad SMB Id (not \xffSMB for SMB1 or not \xfeSMB for SMB2)
- **133:5** (dce_smb) SMB - bad word count or structure size
- **133:6** (dce_smb) SMB - bad byte count
- **133:7** (dce_smb) SMB - bad format type
- **133:8** (dce_smb) SMB - bad offset
- **133:9** (dce_smb) SMB - zero total data count
- **133:10** (dce_smb) SMB - NetBIOS data length less than SMB header length
- **133:12** (dce_smb) SMB - remaining NetBIOS data length less than command byte count
- **133:13** (dce_smb) SMB - remaining NetBIOS data length less than command data size
- **133:14** (dce_smb) SMB - remaining total data count less than this command data size
- **133:15** (dce_smb) SMB - total data sent (STDu64) greater than command total data expected
- **133:16** (dce_smb) SMB - byte count less than command data size (STDu64)
- **133:17** (dce_smb) SMB - invalid command data size for byte count
- **133:18** (dce_smb) SMB - excessive tree connect requests with pending tree connect responses
- **133:19** (dce_smb) SMB - excessive read requests with pending read responses
- **133:20** (dce_smb) SMB - excessive command chaining
- **133:21** (dce_smb) SMB - multiple chained tree connect requests
- **133:22** (dce_smb) SMB - multiple chained tree connect requests
- **133:23** (dce_smb) SMB - chained/compounded login followed by logoff
- **133:24** (dce_smb) SMB - chained/compounded tree connect followed by tree disconnect
- **133:25** (dce_smb) SMB - chained/compounded open pipe followed by close pipe
- **133:26** (dce_smb) SMB - invalid share access
- **133:44** (dce_smb) SMB - invalid SMB version 1 seen
- **133:45** (dce_smb) SMB - invalid SMB version 2 seen
- **133:46** (dce_smb) SMB - invalid user, tree connect, file binding
- **133:47** (dce_smb) SMB - excessive command compounding
- **133:48** (dce_smb) SMB - zero data count
- **133:50** (dce_smb) SMB - maximum number of outstanding requests exceeded
- **133:51** (dce_smb) SMB - outstanding requests with same MID
- **133:52** (dce_smb) SMB - deprecated dialect negotiated
- **133:53** (dce_smb) SMB - deprecated command used

- **133:54** (dce_smb) SMB - unusual command used
- **133:55** (dce_smb) SMB - invalid setup count for command
- **133:56** (dce_smb) SMB - client attempted multiple dialect negotiations on session
- **133:57** (dce_smb) SMB - client attempted to create or set a file's attributes to readonly/hidden/system
- **133:58** (dce_smb) SMB - file offset provided is greater than file size specified
- **133:59** (dce_smb) SMB - next command specified in SMB2 header is beyond payload boundary

Peg counts:

- **dce_smb.events**: total events (sum)
- **dce_smb.pdus**: total connection-oriented PDUs (sum)
- **dce_smb.binds**: total connection-oriented binds (sum)
- **dce_smb.bind_acks**: total connection-oriented binds acks (sum)
- **dce_smb.alter_contexts**: total connection-oriented alter contexts (sum)
- **dce_smb.alter_context_responses**: total connection-oriented alter context responses (sum)
- **dce_smb.bind_naks**: total connection-oriented bind naks (sum)
- **dce_smb.requests**: total connection-oriented requests (sum)
- **dce_smb.responses**: total connection-oriented responses (sum)
- **dce_smb.cancels**: total connection-oriented cancels (sum)
- **dce_smb.orphaned**: total connection-oriented orphaned (sum)
- **dce_smb.faults**: total connection-oriented faults (sum)
- **dce_smb.auth3s**: total connection-oriented auth3s (sum)
- **dce_smb.shutdowns**: total connection-oriented shutdowns (sum)
- **dce_smb.rejects**: total connection-oriented rejects (sum)
- **dce_smb.ms_rpc_http_pdus**: total connection-oriented MS requests to send RPC over HTTP (sum)
- **dce_smb.other_requests**: total connection-oriented other requests (sum)
- **dce_smb.other_responses**: total connection-oriented other responses (sum)
- **dce_smb.request_fragments**: total connection-oriented request fragments (sum)
- **dce_smb.response_fragments**: total connection-oriented response fragments (sum)
- **dce_smb.client_max_fragment_size**: connection-oriented client maximum fragment size (sum)
- **dce_smb.client_min_fragment_size**: connection-oriented client minimum fragment size (sum)
- **dce_smb.client_segs_reassembled**: total connection-oriented client segments reassembled (sum)
- **dce_smb.client_frags_reassembled**: total connection-oriented client fragments reassembled (sum)
- **dce_smb.server_max_fragment_size**: connection-oriented server maximum fragment size (sum)
- **dce_smb.server_min_fragment_size**: connection-oriented server minimum fragment size (sum)
- **dce_smb.server_segs_reassembled**: total connection-oriented server segments reassembled (sum)

- **dce_smb.server_frags_reassembled**: total connection-oriented server fragments reassembled (sum)
- **dce_smb.sessions**: total smb sessions (sum)
- **dce_smb.packets**: total smb packets (sum)
- **dce_smb.ignored_bytes**: total ignored bytes (sum)
- **dce_smb.smb_client_segs_reassembled**: total smb client segments reassembled (sum)
- **dce_smb.smb_server_segs_reassembled**: total smb server segments reassembled (sum)
- **dce_smb.max_outstanding_requests**: total smb maximum outstanding requests (sum)
- **dce_smb.files_processed**: total smb files processed (sum)
- **dce_smb.smbv2_create**: total number of SMBv2 create packets seen (sum)
- **dce_smb.smbv2_write**: total number of SMBv2 write packets seen (sum)
- **dce_smb.smbv2_read**: total number of SMBv2 read packets seen (sum)
- **dce_smb.smbv2_set_info**: total number of SMBv2 set info packets seen (sum)
- **dce_smb.smbv2_tree_connect**: total number of SMBv2 tree connect packets seen (sum)
- **dce_smb.smbv2_tree_disconnect**: total number of SMBv2 tree disconnect packets seen (sum)
- **dce_smb.smbv2_close**: total number of SMBv2 close packets seen (sum)
- **dce_smb.concurrent_sessions**: total concurrent sessions (now)
- **dce_smb.max_concurrent_sessions**: maximum concurrent sessions (max)

9.9 dce_tcp

What: dce over tcp inspection

Type: inspector

Usage: inspect

Configuration:

- bool **dce_tcp.disable_defrag** = false: Disable DCE/RPC defragmentation
- int **dce_tcp.max_frag_len** = 65535: Maximum fragment size for defragmentation { 1514:65535 }
- int **dce_tcp.reassemble_threshold** = 0: Minimum bytes received before performing reassembly { 0:65535 }
- enum **dce_tcp.policy** = WinXP: Target based policy to use { Win2000 | WinXP | WinVista | Win2003 | Win2008 | Win7 | Samba | Samba-3.0.37 | Samba-3.0.22 | Samba-3.0.20 }

Rules:

- **133:27** (dce_tcp) connection oriented DCE/RPC - invalid major version
- **133:28** (dce_tcp) connection oriented DCE/RPC - invalid minor version
- **133:29** (dce_tcp) connection-oriented DCE/RPC - invalid PDU type
- **133:30** (dce_tcp) connection-oriented DCE/RPC - fragment length less than header size
- **133:32** (dce_tcp) connection-oriented DCE/RPC - no context items specified
- **133:33** (dce_tcp) connection-oriented DCE/RPC -no transfer syntaxes specified

- **133:34** (dce_tcp) connection-oriented DCE/RPC - fragment length on non-last fragment less than maximum negotiated fragment transmit size for client
- **133:35** (dce_tcp) connection-oriented DCE/RPC - fragment length greater than maximum negotiated fragment transmit size
- **133:36** (dce_tcp) connection-oriented DCE/RPC - alter context byte order different from bind
- **133:37** (dce_tcp) connection-oriented DCE/RPC - call id of non first/last fragment different from call id established for fragmented request
- **133:38** (dce_tcp) connection-oriented DCE/RPC - opnum of non first/last fragment different from opnum established for fragmented request
- **133:39** (dce_tcp) connection-oriented DCE/RPC - context id of non first/last fragment different from context id established for fragmented request

Peg counts:

- **dce_tcp.events**: total events (sum)
- **dce_tcp.pdus**: total connection-oriented PDUs (sum)
- **dce_tcp.binds**: total connection-oriented binds (sum)
- **dce_tcp.bind_acks**: total connection-oriented binds acks (sum)
- **dce_tcp.alter_contexts**: total connection-oriented alter contexts (sum)
- **dce_tcp.alter_context_responses**: total connection-oriented alter context responses (sum)
- **dce_tcp.bind_naks**: total connection-oriented bind naks (sum)
- **dce_tcp.requests**: total connection-oriented requests (sum)
- **dce_tcp.responses**: total connection-oriented responses (sum)
- **dce_tcp.cancels**: total connection-oriented cancels (sum)
- **dce_tcp.orphaned**: total connection-oriented orphaned (sum)
- **dce_tcp.faults**: total connection-oriented faults (sum)
- **dce_tcp.auth3s**: total connection-oriented auth3s (sum)
- **dce_tcp.shutdowns**: total connection-oriented shutdowns (sum)
- **dce_tcp.rejects**: total connection-oriented rejects (sum)
- **dce_tcp.ms_rpc_http_pdus**: total connection-oriented MS requests to send RPC over HTTP (sum)
- **dce_tcp.other_requests**: total connection-oriented other requests (sum)
- **dce_tcp.other_responses**: total connection-oriented other responses (sum)
- **dce_tcp.request_fragments**: total connection-oriented request fragments (sum)
- **dce_tcp.response_fragments**: total connection-oriented response fragments (sum)
- **dce_tcp.client_max_fragment_size**: connection-oriented client maximum fragment size (sum)
- **dce_tcp.client_min_fragment_size**: connection-oriented client minimum fragment size (sum)
- **dce_tcp.client_segs_reassembled**: total connection-oriented client segments reassembled (sum)
- **dce_tcp.client_frags_reassembled**: total connection-oriented client fragments reassembled (sum)
- **dce_tcp.server_max_fragment_size**: connection-oriented server maximum fragment size (sum)

- **dce_tcp.server_min_fragment_size**: connection-oriented server minimum fragment size (sum)
- **dce_tcp.server_segs_reassembled**: total connection-oriented server segments reassembled (sum)
- **dce_tcp.server_frags_reassembled**: total connection-oriented server fragments reassembled (sum)
- **dce_tcp.tcp_sessions**: total tcp sessions (sum)
- **dce_tcp.tcp_packets**: total tcp packets (sum)
- **dce_tcp.concurrent_sessions**: total concurrent sessions (now)
- **dce_tcp.max_concurrent_sessions**: maximum concurrent sessions (max)

9.10 dce_udp

What: dce over udp inspection

Type: inspector

Usage: inspect

Configuration:

- bool **dce_udp.disable_defrag** = false: Disable DCE/RPC defragmentation
- int **dce_udp.max_frag_len** = 65535: Maximum fragment size for defragmentation { 1514:65535 }
- int **dce_udp.trace**: mask for enabling debug traces in module

Rules:

- **133:40** (dce_udp) connection-less DCE/RPC - invalid major version
- **133:41** (dce_udp) connection-less DCE/RPC - invalid PDU type
- **133:42** (dce_udp) connection-less DCE/RPC - data length less than header size
- **133:43** (dce_udp) connection-less DCE/RPC - bad sequence number

Peg counts:

- **dce_udp.events**: total events (sum)
- **dce_udp.udp_sessions**: total udp sessions (sum)
- **dce_udp.udp_packets**: total udp packets (sum)
- **dce_udp.requests**: total connection-less requests (sum)
- **dce_udp.acks**: total connection-less acks (sum)
- **dce_udp.cancels**: total connection-less cancels (sum)
- **dce_udp.client_facks**: total connection-less client facks (sum)
- **dce_udp.ping**: total connection-less ping (sum)
- **dce_udp.responses**: total connection-less responses (sum)
- **dce_udp.rejects**: total connection-less rejects (sum)
- **dce_udp.cancel_acks**: total connection-less cancel acks (sum)
- **dce_udp.server_facks**: total connection-less server facks (sum)

- **dce_udp.faults**: total connection-less faults (sum)
- **dce_udp.no_calls**: total connection-less no calls (sum)
- **dce_udp.working**: total connection-less working (sum)
- **dce_udp.other_requests**: total connection-less other requests (sum)
- **dce_udp.other_responses**: total connection-less other responses (sum)
- **dce_udp.fragments**: total connection-less fragments (sum)
- **dce_udp.max_fragment_size**: connection-less maximum fragment size (sum)
- **dce_udp.frags_reassembled**: total connection-less fragments reassembled (sum)
- **dce_udp.max_seqnum**: max connection-less seqnum (sum)
- **dce_udp.concurrent_sessions**: total concurrent sessions (now)
- **dce_udp.max_concurrent_sessions**: maximum concurrent sessions (max)

9.11 dnp3

What: dnp3 inspection

Type: inspector

Usage: inspect

Configuration:

- bool **dnp3.check_crc** = false: validate checksums in DNP3 link layer frames

Rules:

- **145:1** (dnp3) DNP3 link-layer frame contains bad CRC
- **145:2** (dnp3) DNP3 link-layer frame was dropped
- **145:3** (dnp3) DNP3 transport-layer segment was dropped during reassembly
- **145:4** (dnp3) DNP3 reassembly buffer was cleared without reassembling a complete message
- **145:5** (dnp3) DNP3 link-layer frame uses a reserved address
- **145:6** (dnp3) DNP3 application-layer fragment uses a reserved function code

Peg counts:

- **dnp3.total_packets**: total packets (sum)
- **dnp3.udp_packets**: total udp packets (sum)
- **dnp3.tcp_pdus**: total tcp pdus (sum)
- **dnp3.dnp3_link_layer_frames**: total dnp3 link layer frames (sum)
- **dnp3.dnp3_application_pdus**: total dnp3 application pdus (sum)
- **dnp3.concurrent_sessions**: total concurrent dnp3 sessions (now)
- **dnp3.max_concurrent_sessions**: maximum concurrent dnp3 sessions (max)

9.12 dns

What: dns inspection

Type: inspector

Usage: inspect

Rules:

- **131:1** (dns) obsolete DNS RR types
- **131:2** (dns) experimental DNS RR types
- **131:3** (dns) DNS client rdata txt overflow

Peg counts:

- **dns.packets**: total packets processed (sum)
- **dns.requests**: total dns requests (sum)
- **dns.responses**: total dns responses (sum)
- **dns.concurrent_sessions**: total concurrent dns sessions (now)
- **dns.max_concurrent_sessions**: maximum concurrent dns sessions (max)

9.13 domain_filter

What: alert on configured HTTP domains

Type: inspector

Usage: inspect

Configuration:

- string **domain_filter.file**: file with list of domains identifying hosts to be filtered
- string **domain_filter.hosts**: list of domains identifying hosts to be filtered

Rules:

- **175:1** (domain_filter) configured domain detected

Peg counts:

- **domain_filter.checked**: domains checked (sum)
- **domain_filter.filtered**: domains filtered (sum)

9.14 dpx

What: dynamic inspector example

Type: inspector

Usage: inspect

Configuration:

- port **dpx.port**: port to check
- int **dpx.max** = 0: maximum payload before alert { 0:65535 }

Rules:

- **256:1** (dpx) too much data sent to port

Peg counts:

- **dpx.packets**: total packets (sum)

9.15 file_id

What: configure file identification

Type: inspector

Usage: global

Configuration:

- int **file_id.type_depth** = 1460: stop type ID at this point { 0: }
- int **file_id.signature_depth** = 10485760: stop signature at this point { 0: }
- int **file_id.block_timeout** = 86400: stop blocking after this many seconds { 0: }
- int **file_id.lookup_timeout** = 2: give up on lookup after this many seconds { 0: }
- bool **file_id.block_timeout_lookup** = false: block if lookup times out
- int **file_id.capture_memcap** = 100: memcap for file capture in megabytes { 0: }
- int **file_id.capture_max_size** = 1048576: stop file capture beyond this point { 0: }
- int **file_id.capture_min_size** = 0: stop file capture if file size less than this { 0: }
- int **file_id.capture_block_size** = 32768: file capture block size in bytes { 8: }
- int **file_id.max_files_cached** = 65536: maximal number of files cached in memory { 8: }
- bool **file_id.enable_type** = true: enable type ID
- bool **file_id.enable_signature** = true: enable signature calculation
- bool **file_id.enable_capture** = false: enable file capture
- int **file_id.show_data_depth** = 100: print this many octets { 0: }
- int **file_id.file_rules[].rev** = 0: rule revision { 0: }
- string **file_id.file_rules[].msg**: information about the file type
- string **file_id.file_rules[].type**: file type name

- int **file_id.file_rules[].id** = 0: file type id { 0: }
- string **file_id.file_rules[].category**: file type category
- string **file_id.file_rules[].group**: comma separated list of groups associated with file type
- string **file_id.file_rules[].version**: file type version
- string **file_id.file_rules[].magic[].content**: file magic content
- int **file_id.file_rules[].magic[].offset** = 0: file magic offset { 0: }
- int **file_id.file_policy[].when.file_type_id** = 0: unique ID for file type in file magic rule { 0: }
- string **file_id.file_policy[].when.sha256**: SHA 256
- enum **file_id.file_policy[].use.verdict** = unknown: what to do with matching traffic { unknown | log | stop | block | reset }
- bool **file_id.file_policy[].use.enable_file_type** = false: true/false → enable/disable file type identification
- bool **file_id.file_policy[].use.enable_file_signature** = false: true/false → enable/disable file signature
- bool **file_id.file_policy[].use.enable_file_capture** = false: true/false → enable/disable file capture
- bool **file_id.trace_type** = false: enable runtime dump of type info
- bool **file_id.trace_signature** = false: enable runtime dump of signature info
- bool **file_id.trace_stream** = false: enable runtime dump of file data
- int **file_id.verdict_delay** = 0: number of queries to return final verdict { 0: }

Peg counts:

- **file_id.total_files**: number of files processed (sum)
- **file_id.total_file_data**: number of file data bytes processed (sum)
- **file_id.cache_failures**: number of file cache add failures (sum)

9.16 file_log

What: log file event to file.log

Type: inspector

Usage: inspect

Configuration:

- bool **file_log.log_pkt_time** = true: log the packet time when event generated
- bool **file_log.log_sys_time** = false: log the system time when event generated

Peg counts:

- **file_log.total_events**: total file events (sum)

9.17 ftp_client

What: FTP client configuration module for use with ftp_server

Type: inspector

Usage: inspect

Configuration:

- bool **ftp_client.bounce** = false: check for bounces
- addr **ftp_client.bounce_to[].address** = 1.0.0.0/32: allowed IP address in CIDR format
- port **ftp_client.bounce_to[].port** = 20: allowed port { 1: }
- port **ftp_client.bounce_to[].last_port**: optional allowed range from port to last_port inclusive { 0: }
- bool **ftp_client.ignore_telnet_erase_cmds** = false: ignore erase character and erase line commands when normalizing
- int **ftp_client.max_resp_len** = -1: maximum FTP response accepted by client { -1: }
- bool **ftp_client.telnet_cmds** = false: detect Telnet escape sequences on FTP control channel

9.18 ftp_data

What: FTP data channel handler

Type: inspector

Usage: inspect

Peg counts:

- **ftp_data.packets**: total packets (sum)

9.19 ftp_server

What: main FTP module; ftp_client should also be configured

Type: inspector

Usage: inspect

Configuration:

- string **ftp_server.chk_str_fmt**: check the formatting of the given commands
- string **ftp_server.data_chan_cmds**: check the formatting of the given commands
- string **ftp_server.data_rest_cmds**: check the formatting of the given commands
- string **ftp_server.data_xfer_cmds**: check the formatting of the given commands
- string **ftp_server.directory_cmds[].dir_cmd**: directory command
- int **ftp_server.directory_cmds[].rsp_code** = 200: expected successful response code for command { 200: }
- string **ftp_server.file_put_cmds**: check the formatting of the given commands
- string **ftp_server.file_get_cmds**: check the formatting of the given commands
- string **ftp_server.encr_cmds**: check the formatting of the given commands
- string **ftp_server.login_cmds**: check the formatting of the given commands

- bool **ftp_server.check_encrypted** = false: check for end of encryption
- string **ftp_server.cmd_validity[].command**: command string
- string **ftp_server.cmd_validity[].format**: format specification
- int **ftp_server.cmd_validity[].length** = 0: specify non-default maximum for command { 0: }
- int **ftp_server.def_max_param_len** = 100: default maximum length of commands handled by server; 0 is unlimited { 1: }
- bool **ftp_server.encrypted_traffic** = false: check for encrypted Telnet and FTP
- string **ftp_server.ftp_cmds**: specify additional commands supported by server beyond RFC 959
- bool **ftp_server.ignore_data_chan** = false: do not inspect FTP data channels
- bool **ftp_server.ignore_telnet_erase_cmds** = false: ignore erase character and erase line commands when normalizing
- bool **ftp_server.print_cmds** = false: print command configurations on start up
- bool **ftp_server.telnet_cmds** = false: detect Telnet escape sequences of FTP control channel

Rules:

- **125:1** (ftp_server) TELNET cmd on FTP command channel
- **125:2** (ftp_server) invalid FTP command
- **125:3** (ftp_server) FTP command parameters were too long
- **125:4** (ftp_server) FTP command parameters were malformed
- **125:5** (ftp_server) FTP command parameters contained potential string format
- **125:6** (ftp_server) FTP response message was too long
- **125:7** (ftp_server) FTP traffic encrypted
- **125:8** (ftp_server) FTP bounce attempt
- **125:9** (ftp_server) evasive (incomplete) TELNET cmd on FTP command channel

Peg counts:

- **ftp_server.total_packets**: total packets (sum)
- **ftp_server.concurrent_sessions**: total concurrent FTP sessions (now)
- **ftp_server.max_concurrent_sessions**: maximum concurrent FTP sessions (max)

9.20 gtp_inspect

What: gtp control channel inspection

Type: inspector

Usage: inspect

Configuration:

- int **gtp_inspect[].version** = 2: GTP version { 0:2 }
- int **gtp_inspect[].messages[].type** = 0: message type code { 0:255 }
- string **gtp_inspect[].messages[].name**: message name

- int **gtp_inspect[].infos[].type** = 0: information element type code { 0:255 }
- string **gtp_inspect[].infos[].name**: information element name
- int **gtp_inspect[].infos[].length** = 0: information element type code { 0:255 }
- int **gtp_inspect.trace**: mask for enabling debug traces in module

Rules:

- **143:1** (gtp_inspect) message length is invalid
- **143:2** (gtp_inspect) information element length is invalid
- **143:3** (gtp_inspect) information elements are out of order

Peg counts:

- **gtp_inspect.sessions**: total sessions processed (sum)
- **gtp_inspect.concurrent_sessions**: total concurrent gtp sessions (now)
- **gtp_inspect.max_concurrent_sessions**: maximum concurrent gtp sessions (max)
- **gtp_inspect.events**: requests (sum)
- **gtp_inspect.unknown_types**: unknown message types (sum)
- **gtp_inspect.unknown_infos**: unknown information elements (sum)

9.21 http2_inspect

What: HTTP/2 inspector

Type: inspector

Usage: inspect

Rules:

Peg counts:

- **http2_inspect.flows**: HTTP connections inspected (sum)
- **http2_inspect.concurrent_sessions**: total concurrent HTTP/2 sessions (now)
- **http2_inspect.max_concurrent_sessions**: maximum concurrent HTTP/2 sessions (max)

9.22 http_inspect

What: HTTP inspector

Type: inspector

Usage: inspect

Configuration:

- int **http_inspect.request_depth** = -1: maximum request message body bytes to examine (-1 no limit) { -1: }
- int **http_inspect.response_depth** = -1: maximum response message body bytes to examine (-1 no limit) { -1: }
- bool **http_inspect.unzip** = true: decompress gzip and deflate message bodies

- bool **http_inspect.normalize_utf** = true: normalize charset utf encodings in response bodies
- bool **http_inspect.decompress_pdf** = false: decompress pdf files in response bodies
- bool **http_inspect.decompress_swf** = false: decompress swf files in response bodies
- bool **http_inspect.normalize_javascript** = false: normalize javascript in response bodies
- int **http_inspect.max_javascript_whitespaces** = 200: maximum consecutive whitespaces allowed within the Javascript obfuscated data { 1:65535 }
- bit_list **http_inspect.bad_characters**: alert when any of specified bytes are present in URI after percent decoding { 255 }
- string **http_inspect.ignore_unreserved**: do not alert when the specified unreserved characters are percent-encoded in a URI.Unreserved characters are 0-9, a-z, A-Z, period, underscore, tilde, and minus. { (optional) }
- bool **http_inspect.percent_u** = false: normalize %uNNNN and %UNNNN encodings
- bool **http_inspect.utf8** = true: normalize 2-byte and 3-byte UTF-8 characters to a single byte
- bool **http_inspect.utf8_bare_byte** = false: when doing UTF-8 character normalization include bytes that were not percent encoded
- bool **http_inspect.iis_unicode** = false: use IIS unicode code point mapping to normalize characters
- string **http_inspect.iis_unicode_map_file**: file containing code points for IIS unicode. { (optional) }
- int **http_inspect.iis_unicode_code_page** = 1252: code page to use from the IIS unicode map file { 0:65535 }
- bool **http_inspect.iis_double_decode** = false: perform double decoding of percent encodings to normalize characters
- int **http_inspect.oversize_dir_length** = 300: maximum length for URL directory { 1:65535 }
- bool **http_inspect.backslash_to_slash** = false: replace \ with / when normalizing URIs
- bool **http_inspect.plus_to_space** = true: replace + with <sp> when normalizing URIs
- bool **http_inspect.simplify_path** = true: reduce URI directory path to simplest form
- bool **http_inspect.test_input** = false: read HTTP messages from text file
- bool **http_inspect.test_output** = false: print out HTTP section data
- int **http_inspect.print_amount** = 1200: number of characters to print from a Field { 1:1000000 }
- bool **http_inspect.print_hex** = false: nonprinting characters printed in [HH] format instead of using an asterisk
- bool **http_inspect.show_pegs** = true: display peg counts with test output
- bool **http_inspect.show_scan** = false: display scanned segments

Rules:

- **119:1** (http_inspect) ascii encoding
- **119:2** (http_inspect) double decoding attack
- **119:3** (http_inspect) u encoding
- **119:4** (http_inspect) bare byte unicode encoding
- **119:5** (http_inspect) obsolete event—deleted
- **119:6** (http_inspect) UTF-8 encoding
- **119:7** (http_inspect) unicode map code point encoding in URI

- **119:8** (http_inspect) multi_slash encoding
- **119:9** (http_inspect) backslash used in URI path
- **119:10** (http_inspect) self directory traversal
- **119:11** (http_inspect) directory traversal
- **119:12** (http_inspect) apache whitespace (tab)
- **119:13** (http_inspect) HTTP header line terminated by LF without a CR
- **119:14** (http_inspect) non-RFC defined char
- **119:15** (http_inspect) oversize request-uri directory
- **119:16** (http_inspect) oversize chunk encoding
- **119:17** (http_inspect) unauthorized proxy use detected
- **119:18** (http_inspect) webroot directory traversal
- **119:19** (http_inspect) long header
- **119:20** (http_inspect) max header fields
- **119:21** (http_inspect) multiple content length
- **119:22** (http_inspect) obsolete event—deleted
- **119:23** (http_inspect) invalid IP in true-client-IP/XFF header
- **119:24** (http_inspect) multiple host hdrs detected
- **119:25** (http_inspect) hostname exceeds 255 characters
- **119:26** (http_inspect) too much whitespace in header (not implemented yet)
- **119:27** (http_inspect) client consecutive small chunk sizes
- **119:28** (http_inspect) POST or PUT w/o content-length or chunks
- **119:29** (http_inspect) multiple true ips in a session
- **119:30** (http_inspect) both true-client-IP and XFF hdrs present
- **119:31** (http_inspect) unknown method
- **119:32** (http_inspect) simple request
- **119:33** (http_inspect) unescaped space in HTTP URI
- **119:34** (http_inspect) too many pipelined requests
- **119:101** (http_inspect) anomalous http server on undefined HTTP port
- **119:102** (http_inspect) invalid status code in HTTP response
- **119:103** (http_inspect) unused event number—should not appear
- **119:104** (http_inspect) HTTP response has UTF charset that failed to normalize
- **119:105** (http_inspect) HTTP response has UTF-7 charset
- **119:106** (http_inspect) HTTP response gzip decompression failed
- **119:107** (http_inspect) server consecutive small chunk sizes
- **119:108** (http_inspect) unused event number—should not appear

- **119:109** (http_inspect) javascript obfuscation levels exceeds 1
- **119:110** (http_inspect) javascript whitespaces exceeds max allowed
- **119:111** (http_inspect) multiple encodings within javascript obfuscated data
- **119:112** (http_inspect) SWF file zlib decompression failure
- **119:113** (http_inspect) SWF file LZMA decompression failure
- **119:114** (http_inspect) PDF file deflate decompression failure
- **119:115** (http_inspect) PDF file unsupported compression type
- **119:116** (http_inspect) PDF file cascaded compression
- **119:117** (http_inspect) PDF file parse failure
- **119:201** (http_inspect) not HTTP traffic
- **119:202** (http_inspect) chunk length has excessive leading zeros
- **119:203** (http_inspect) white space before or between messages
- **119:204** (http_inspect) request message without URI
- **119:205** (http_inspect) control character in reason phrase
- **119:206** (http_inspect) illegal extra whitespace in start line
- **119:207** (http_inspect) corrupted HTTP version
- **119:208** (http_inspect) unknown HTTP version
- **119:209** (http_inspect) format error in HTTP header
- **119:210** (http_inspect) chunk header options present
- **119:211** (http_inspect) URI badly formatted
- **119:212** (http_inspect) unrecognized type of percent encoding in URI
- **119:213** (http_inspect) HTTP chunk misformatted
- **119:214** (http_inspect) white space adjacent to chunk length
- **119:215** (http_inspect) white space within header name
- **119:216** (http_inspect) excessive gzip compression
- **119:217** (http_inspect) gzip decompression failed
- **119:218** (http_inspect) HTTP 0.9 requested followed by another request
- **119:219** (http_inspect) HTTP 0.9 request following a normal request
- **119:220** (http_inspect) message has both Content-Length and Transfer-Encoding
- **119:221** (http_inspect) status code implying no body combined with Transfer-Encoding or nonzero Content-Length
- **119:222** (http_inspect) Transfer-Encoding not ending with chunked
- **119:223** (http_inspect) Transfer-Encoding with encodings before chunked
- **119:224** (http_inspect) misformatted HTTP traffic
- **119:225** (http_inspect) unsupported Content-Encoding used
- **119:226** (http_inspect) unknown Content-Encoding used

- **119:227** (http_inspect) multiple Content-Encodings applied
- **119:228** (http_inspect) server response before client request
- **119:229** (http_inspect) PDF/SWF decompression of server response too big
- **119:230** (http_inspect) nonprinting character in HTTP message header name
- **119:231** (http_inspect) bad Content-Length value in HTTP header
- **119:232** (http_inspect) HTTP header line wrapped
- **119:233** (http_inspect) HTTP header line terminated by CR without a LF
- **119:234** (http_inspect) chunk terminated by nonstandard separator
- **119:235** (http_inspect) chunk length terminated by LF without CR
- **119:236** (http_inspect) more than one response with 100 status code
- **119:237** (http_inspect) 100 status code not in response to Expect header
- **119:238** (http_inspect) 1XX status code other than 100 or 101
- **119:239** (http_inspect) Expect header sent without a message body
- **119:240** (http_inspect) HTTP 1.0 message with Transfer-Encoding header
- **119:241** (http_inspect) Content-Transfer-Encoding used as HTTP header
- **119:242** (http_inspect) illegal field in chunked message trailers
- **119:243** (http_inspect) header field inappropriately appears twice or has two values
- **119:244** (http_inspect) invalid value chunked in Content-Encoding header
- **119:245** (http_inspect) 206 response sent to a request without a Range header
- **119:246** (http_inspect) *HTTP* in version field not all upper case
- **119:247** (http_inspect) white space embedded in critical header value
- **119:248** (http_inspect) gzip compressed data followed by unexpected non-gzip data

Peg counts:

- **http_inspect.flows**: HTTP connections inspected (sum)
- **http_inspect.scans**: TCP segments scanned looking for HTTP messages (sum)
- **http_inspect.reassembles**: TCP segments combined into HTTP messages (sum)
- **http_inspect.inspections**: total message sections inspected (sum)
- **http_inspect.requests**: HTTP request messages inspected (sum)
- **http_inspect.responses**: HTTP response messages inspected (sum)
- **http_inspect.get_requests**: GET requests inspected (sum)
- **http_inspect.head_requests**: HEAD requests inspected (sum)
- **http_inspect.post_requests**: POST requests inspected (sum)
- **http_inspect.put_requests**: PUT requests inspected (sum)
- **http_inspect.delete_requests**: DELETE requests inspected (sum)

- **http_inspect.connect_requests**: CONNECT requests inspected (sum)
- **http_inspect.options_requests**: OPTIONS requests inspected (sum)
- **http_inspect.trace_requests**: TRACE requests inspected (sum)
- **http_inspect.other_requests**: other request methods inspected (sum)
- **http_inspect.request_bodies**: POST, PUT, and other requests with message bodies (sum)
- **http_inspect.chunked**: chunked message bodies (sum)
- **http_inspect.uri_normalizations**: URIs needing to be normalization (sum)
- **http_inspect.uri_path**: URIs with path problems (sum)
- **http_inspect.uri_coding**: URIs with character coding problems (sum)
- **http_inspect.concurrent_sessions**: total concurrent http sessions (now)
- **http_inspect.max_concurrent_sessions**: maximum concurrent http sessions (max)

9.23 imap

What: imap inspection

Type: inspector

Usage: inspect

Configuration:

- int **imap.b64_decode_depth** = 1460: base64 decoding depth (-1 no limit) { -1:65535 }
- int **imap.bitenc_decode_depth** = 1460: non-Encoded MIME attachment extraction depth (-1 no limit) { -1:65535 }
- int **imap.qp_decode_depth** = 1460: quoted Printable decoding depth (-1 no limit) { -1:65535 }
- int **imap.uu_decode_depth** = 1460: Unix-to-Unix decoding depth (-1 no limit) { -1:65535 }

Rules:

- **141:1** (imap) unknown IMAP3 command
- **141:2** (imap) unknown IMAP3 response
- **141:4** (imap) base64 decoding failed
- **141:5** (imap) quoted-printable decoding failed
- **141:7** (imap) Unix-to-Unix decoding failed

Peg counts:

- **imap.packets**: total packets processed (sum)
- **imap.sessions**: total imap sessions (sum)
- **imap.concurrent_sessions**: total concurrent imap sessions (now)
- **imap.max_concurrent_sessions**: maximum concurrent imap sessions (max)
- **imap.b64_attachments**: total base64 attachments decoded (sum)
- **imap.b64_decoded_bytes**: total base64 decoded bytes (sum)

- **imap.qp_attachments**: total quoted-printable attachments decoded (sum)
- **imap.qp_decoded_bytes**: total quoted-printable decoded bytes (sum)
- **imap.uu_attachments**: total uu attachments decoded (sum)
- **imap.uu_decoded_bytes**: total uu decoded bytes (sum)
- **imap.non_encoded_attachments**: total non-encoded attachments extracted (sum)
- **imap.non_encoded_bytes**: total non-encoded extracted bytes (sum)

9.24 modbus

What: modbus inspection

Type: inspector

Usage: inspect

Rules:

- **144:1** (modbus) length in Modbus MBAP header does not match the length needed for the given function
- **144:2** (modbus) Modbus protocol ID is non-zero
- **144:3** (modbus) reserved Modbus function code in use

Peg counts:

- **modbus.sessions**: total sessions processed (sum)
- **modbus.frames**: total Modbus messages (sum)
- **modbus.concurrent_sessions**: total concurrent modbus sessions (now)
- **modbus.max_concurrent_sessions**: maximum concurrent modbus sessions (max)

9.25 normalizer

What: packet scrubbing for inline mode

Type: inspector

Usage: inspect

Configuration:

- bool **normalizer.ip4.base** = true: clear options
- bool **normalizer.ip4.df** = false: clear don't frag flag
- bool **normalizer.ip4.rf** = false: clear reserved flag
- bool **normalizer.ip4.tos** = false: clear tos / differentiated services byte
- bool **normalizer.ip4.trim** = false: truncate excess payload beyond datagram length
- bool **normalizer.tcp.base** = true: clear reserved bits and option padding and fix urgent pointer / flags issues
- bool **normalizer.tcp.block** = true: allow packet drops during TCP normalization
- bool **normalizer.tcp.urp** = true: adjust urgent pointer if beyond segment length

- bool **normalizer.tcp.ips** = false: ensure consistency in retransmitted data
- select **normalizer.tcp.ecn** = off: clear ecn for all packets | sessions w/o ecn setup { off | packet | stream }
- bool **normalizer.tcp.pad** = true: clear any option padding bytes
- bool **normalizer.tcp.trim_syn** = false: remove data on SYN
- bool **normalizer.tcp.trim_rst** = false: remove any data from RST packet
- bool **normalizer.tcp.trim_win** = false: trim data to window
- bool **normalizer.tcp.trim_mss** = false: trim data to MSS
- bool **normalizer.tcp.trim** = false: enable all of the TCP trim options
- bool **normalizer.tcp.opts** = true: clear all options except mss, wscale, timestamp, and any explicitly allowed
- bool **normalizer.tcp.req_urg** = true: clear the urgent pointer if the urgent flag is not set
- bool **normalizer.tcp.req_pay** = true: clear the urgent pointer and the urgent flag if there is no payload
- bool **normalizer.tcp.rsv** = true: clear the reserved bits in the TCP header
- bool **normalizer.tcp.req_urp** = true: clear the urgent flag if the urgent pointer is not set
- multi **normalizer.tcp.allow_names**: don't clear given option names { sack | echo | partial_order | conn_count | alt_checksum | md5 }
- string **normalizer.tcp.allow_codes**: don't clear given option codes
- bool **normalizer.ip6** = false: clear reserved flag
- bool **normalizer.icmp4** = false: clear reserved flag
- bool **normalizer.icmp6** = false: clear reserved flag

Peg counts:

- **normalizer.test_ip4_trim**: test eth packets trimmed to datagram size (sum)
- **normalizer.ip4_trim**: eth packets trimmed to datagram size (sum)
- **normalizer.test_ip4_tos**: test type of service normalizations (sum)
- **normalizer.ip4_tos**: type of service normalizations (sum)
- **normalizer.test_ip4_df**: test don't frag bit normalizations (sum)
- **normalizer.ip4_df**: don't frag bit normalizations (sum)
- **normalizer.test_ip4_rf**: test reserved flag bit clears (sum)
- **normalizer.ip4_rf**: reserved flag bit clears (sum)
- **normalizer.test_ip4_ttl**: test time-to-live normalizations (sum)
- **normalizer.ip4_ttl**: time-to-live normalizations (sum)
- **normalizer.test_ip4_opts**: test ip4 options cleared (sum)
- **normalizer.ip4_opts**: ip4 options cleared (sum)
- **normalizer.test_icmp4_echo**: test icmp4 ping normalizations (sum)
- **normalizer.icmp4_echo**: icmp4 ping normalizations (sum)
- **normalizer.test_ip6_hops**: test ip6 hop limit normalizations (sum)

- **normalizer.ip6_hops**: ip6 hop limit normalizations (sum)
- **normalizer.test_ip6_options**: test ip6 options cleared (sum)
- **normalizer.ip6_options**: ip6 options cleared (sum)
- **normalizer.test_icmp6_echo**: test icmp6 echo normalizations (sum)
- **normalizer.icmp6_echo**: icmp6 echo normalizations (sum)
- **normalizer.test_tcp_syn_options**: test SYN only options cleared from non-SYN packets (sum)
- **normalizer.tcp_syn_options**: SYN only options cleared from non-SYN packets (sum)
- **normalizer.test_tcp_options**: test packets with options cleared (sum)
- **normalizer.tcp_options**: packets with options cleared (sum)
- **normalizer.test_tcp_padding**: test packets with padding cleared (sum)
- **normalizer.tcp_padding**: packets with padding cleared (sum)
- **normalizer.test_tcp_reserved**: test packets with reserved bits cleared (sum)
- **normalizer.tcp_reserved**: packets with reserved bits cleared (sum)
- **normalizer.test_tcp_nonce**: test packets with nonce bit cleared (sum)
- **normalizer.tcp_nonce**: packets with nonce bit cleared (sum)
- **normalizer.test_tcp_urgent_ptr**: test packets without data with urgent pointer cleared (sum)
- **normalizer.tcp_urgent_ptr**: packets without data with urgent pointer cleared (sum)
- **normalizer.test_tcp_ecn_pkt**: test packets with ECN bits cleared (sum)
- **normalizer.tcp_ecn_pkt**: packets with ECN bits cleared (sum)
- **normalizer.test_tcp_ts_ecr**: test timestamp cleared on non-ACKs (sum)
- **normalizer.tcp_ts_ecr**: timestamp cleared on non-ACKs (sum)
- **normalizer.test_tcp_req_urg**: test cleared urgent pointer when urgent flag is not set (sum)
- **normalizer.tcp_req_urg**: cleared urgent pointer when urgent flag is not set (sum)
- **normalizer.test_tcp_req_pay**: test cleared urgent pointer and urgent flag when there is no payload (sum)
- **normalizer.tcp_req_pay**: cleared urgent pointer and urgent flag when there is no payload (sum)
- **normalizer.test_tcp_req_urp**: test cleared the urgent flag if the urgent pointer is not set (sum)
- **normalizer.tcp_req_urp**: cleared the urgent flag if the urgent pointer is not set (sum)
- **normalizer.test_tcp_trim_syn**: test tcp segments trimmed on SYN (sum)
- **normalizer.tcp_trim_syn**: tcp segments trimmed on SYN (sum)
- **normalizer.test_tcp_trim_rst**: test RST packets with data trimmed (sum)
- **normalizer.tcp_trim_rst**: RST packets with data trimmed (sum)
- **normalizer.test_tcp_trim_win**: test data trimmed to window (sum)
- **normalizer.tcp_trim_win**: data trimmed to window (sum)
- **normalizer.test_tcp_trim_mss**: test data trimmed to MSS (sum)
- **normalizer.tcp_trim_mss**: data trimmed to MSS (sum)

- **normalizer.test_tcp_ecn_session**: test ECN bits cleared (sum)
- **normalizer.tcp_ecn_session**: ECN bits cleared (sum)
- **normalizer.test_tcp_ts_nop**: test timestamp options cleared (sum)
- **normalizer.tcp_ts_nop**: timestamp options cleared (sum)
- **normalizer.test_tcp_ips_data**: test normalized segments (sum)
- **normalizer.tcp_ips_data**: normalized segments (sum)
- **normalizer.test_tcp_block**: test blocked segments (sum)
- **normalizer.tcp_block**: blocked segments (sum)

9.26 packet_capture

What: raw packet dumping facility

Type: inspector

Usage: global

Configuration:

- bool **packet_capture.enable** = false: initially enable packet dumping
- string **packet_capture.filter**: bpf filter to use for packet dump

Commands:

- **packet_capture.enable**(filter): dump raw packets
- **packet_capture.disable**(): stop packet dump

Peg counts:

- **packet_capture.processed**: packets processed against filter (sum)
- **packet_capture.captured**: packets matching dumped after matching filter (sum)

9.27 perf_monitor

What: performance monitoring and flow statistics collection

Type: inspector

Usage: global

Configuration:

- bool **perf_monitor.base** = true: enable base statistics { nullptr }
- bool **perf_monitor.cpu** = false: enable cpu statistics { nullptr }
- bool **perf_monitor.flow** = false: enable traffic statistics
- bool **perf_monitor.flow_ip** = false: enable statistics on host pairs
- int **perf_monitor.packets** = 10000: minimum packets to report { 0: }
- int **perf_monitor.seconds** = 60: report interval { 1: }

- int **perf_monitor.flow_ip_memcap** = 52428800: maximum memory in bytes for flow tracking { 8200: }
- int **perf_monitor.max_file_size** = 1073741824: files will be rolled over if they exceed this size { 4096: }
- int **perf_monitor.flow_ports** = 1023: maximum ports to track { 0:65535 }
- enum **perf_monitor.output** = file: output location for stats { file | console }
- string **perf_monitor.modules[].name**: name of the module
- string **perf_monitor.modules[].pegs**: list of statistics to track or empty for all counters
- enum **perf_monitor.format** = csv: output format for stats { csv | text | json | flatbuffers }
- bool **perf_monitor.summary** = false: output summary at shutdown

Peg counts:

- **perf_monitor.packets**: total packets (sum)

9.28 pop

What: pop inspection

Type: inspector

Usage: inspect

Configuration:

- int **pop.b64_decode_depth** = 1460: base64 decoding depth (-1 no limit) { -1:65535 }
- int **pop.bitenc_decode_depth** = 1460: Non-Encoded MIME attachment extraction depth (-1 no limit) { -1:65535 }
- int **pop.qp_decode_depth** = 1460: Quoted Printable decoding depth (-1 no limit) { -1:65535 }
- int **pop.uu_decode_depth** = 1460: Unix-to-Unix decoding depth (-1 no limit) { -1:65535 }

Rules:

- **142:1** (pop) unknown POP3 command
- **142:2** (pop) unknown POP3 response
- **142:4** (pop) base64 decoding failed
- **142:5** (pop) quoted-printable decoding failed
- **142:7** (pop) Unix-to-Unix decoding failed

Peg counts:

- **pop.packets**: total packets processed (sum)
- **pop.sessions**: total pop sessions (sum)
- **pop.concurrent_sessions**: total concurrent pop sessions (now)
- **pop.max_concurrent_sessions**: maximum concurrent pop sessions (max)
- **pop.b64_attachments**: total base64 attachments decoded (sum)
- **pop.b64_decoded_bytes**: total base64 decoded bytes (sum)

- **pop.qp_attachments**: total quoted-printable attachments decoded (sum)
- **pop.qp_decoded_bytes**: total quoted-printable decoded bytes (sum)
- **pop.uu_attachments**: total uu attachments decoded (sum)
- **pop.uu_decoded_bytes**: total uu decoded bytes (sum)
- **pop.non_encoded_attachments**: total non-encoded attachments extracted (sum)
- **pop.non_encoded_bytes**: total non-encoded extracted bytes (sum)

9.29 port_scan

What: detect various ip, icmp, tcp, and udp port or protocol scans

Type: inspector

Usage: global

Configuration:

- int **port_scan.memcap** = 1048576: maximum tracker memory in bytes { 1: }
- multi **port_scan.protos** = all: choose the protocols to monitor { tcp | udp | icmp | ip | all }
- multi **port_scan.scan_types** = all: choose type of scans to look for { portscan | portsweep | decoy_portscan | distributed_portscan | all }
- string **port_scan.watch_ip**: list of CIDRs with optional ports to watch
- string **port_scan.ignore_scanners**: list of CIDRs with optional ports to ignore if the source of scan alerts
- string **port_scan.ignore_scanned**: list of CIDRs with optional ports to ignore if the destination of scan alerts
- bool **port_scan.alert_all** = false: alert on all events over threshold within window if true; else alert on first only
- bool **port_scan.include_midstream** = false: list of CIDRs with optional ports
- int **port_scan.tcp_ports.scans** = 100: scan attempts { 0: }
- int **port_scan.tcp_ports.rejects** = 15: scan attempts with negative response { 0: }
- int **port_scan.tcp_ports.nets** = 25: number of times address changed from prior attempt { 0: }
- int **port_scan.tcp_ports.ports** = 25: number of times port (or proto) changed from prior attempt { 0: }
- int **port_scan.tcp_decoy.scans** = 100: scan attempts { 0: }
- int **port_scan.tcp_decoy.rejects** = 15: scan attempts with negative response { 0: }
- int **port_scan.tcp_decoy.nets** = 25: number of times address changed from prior attempt { 0: }
- int **port_scan.tcp_decoy.ports** = 25: number of times port (or proto) changed from prior attempt { 0: }
- int **port_scan.tcp_sweep.scans** = 100: scan attempts { 0: }
- int **port_scan.tcp_sweep.rejects** = 15: scan attempts with negative response { 0: }
- int **port_scan.tcp_sweep.nets** = 25: number of times address changed from prior attempt { 0: }
- int **port_scan.tcp_sweep.ports** = 25: number of times port (or proto) changed from prior attempt { 0: }
- int **port_scan.tcp_dist.scans** = 100: scan attempts { 0: }
- int **port_scan.tcp_dist.rejects** = 15: scan attempts with negative response { 0: }

- int **port_scan.tcp_dist.nets** = 25: number of times address changed from prior attempt { 0: }
- int **port_scan.tcp_dist.ports** = 25: number of times port (or proto) changed from prior attempt { 0: }
- int **port_scan.udp_ports.scans** = 100: scan attempts { 0: }
- int **port_scan.udp_ports.rejects** = 15: scan attempts with negative response { 0: }
- int **port_scan.udp_ports.nets** = 25: number of times address changed from prior attempt { 0: }
- int **port_scan.udp_ports.ports** = 25: number of times port (or proto) changed from prior attempt { 0: }
- int **port_scan.udp_decoy.scans** = 100: scan attempts { 0: }
- int **port_scan.udp_decoy.rejects** = 15: scan attempts with negative response { 0: }
- int **port_scan.udp_decoy.nets** = 25: number of times address changed from prior attempt { 0: }
- int **port_scan.udp_decoy.ports** = 25: number of times port (or proto) changed from prior attempt { 0: }
- int **port_scan.udp_sweep.scans** = 100: scan attempts { 0: }
- int **port_scan.udp_sweep.rejects** = 15: scan attempts with negative response { 0: }
- int **port_scan.udp_sweep.nets** = 25: number of times address changed from prior attempt { 0: }
- int **port_scan.udp_sweep.ports** = 25: number of times port (or proto) changed from prior attempt { 0: }
- int **port_scan.udp_dist.scans** = 100: scan attempts { 0: }
- int **port_scan.udp_dist.rejects** = 15: scan attempts with negative response { 0: }
- int **port_scan.udp_dist.nets** = 25: number of times address changed from prior attempt { 0: }
- int **port_scan.udp_dist.ports** = 25: number of times port (or proto) changed from prior attempt { 0: }
- int **port_scan.ip_proto.scans** = 100: scan attempts { 0: }
- int **port_scan.ip_proto.rejects** = 15: scan attempts with negative response { 0: }
- int **port_scan.ip_proto.nets** = 25: number of times address changed from prior attempt { 0: }
- int **port_scan.ip_proto.ports** = 25: number of times port (or proto) changed from prior attempt { 0: }
- int **port_scan.ip_decoy.scans** = 100: scan attempts { 0: }
- int **port_scan.ip_decoy.rejects** = 15: scan attempts with negative response { 0: }
- int **port_scan.ip_decoy.nets** = 25: number of times address changed from prior attempt { 0: }
- int **port_scan.ip_decoy.ports** = 25: number of times port (or proto) changed from prior attempt { 0: }
- int **port_scan.ip_sweep.scans** = 100: scan attempts { 0: }
- int **port_scan.ip_sweep.rejects** = 15: scan attempts with negative response { 0: }
- int **port_scan.ip_sweep.nets** = 25: number of times address changed from prior attempt { 0: }
- int **port_scan.ip_sweep.ports** = 25: number of times port (or proto) changed from prior attempt { 0: }
- int **port_scan.ip_dist.scans** = 100: scan attempts { 0: }
- int **port_scan.ip_dist.rejects** = 15: scan attempts with negative response { 0: }
- int **port_scan.ip_dist.nets** = 25: number of times address changed from prior attempt { 0: }
- int **port_scan.ip_dist.ports** = 25: number of times port (or proto) changed from prior attempt { 0: }
- int **port_scan.icmp_sweep.scans** = 100: scan attempts { 0: }

- int **port_scan.icmp_sweep.rejects** = 15: scan attempts with negative response { 0: }
- int **port_scan.icmp_sweep.nets** = 25: number of times address changed from prior attempt { 0: }
- int **port_scan.icmp_sweep.ports** = 25: number of times port (or proto) changed from prior attempt { 0: }
- int **port_scan.tcp_window** = 0: detection interval for all TCP scans { 0: }
- int **port_scan.udp_window** = 0: detection interval for all UDP scans { 0: }
- int **port_scan.ip_window** = 0: detection interval for all IP scans { 0: }
- int **port_scan.icmp_window** = 0: detection interval for all ICMP scans { 0: }

Rules:

- **122:1** (port_scan) TCP portscan
- **122:2** (port_scan) TCP decoy portscan
- **122:3** (port_scan) TCP portsweep
- **122:4** (port_scan) TCP distributed portscan
- **122:5** (port_scan) TCP filtered portscan
- **122:6** (port_scan) TCP filtered decoy portscan
- **122:7** (port_scan) TCP filtered portsweep
- **122:8** (port_scan) TCP filtered distributed portscan
- **122:9** (port_scan) IP protocol scan
- **122:10** (port_scan) IP decoy protocol scan
- **122:11** (port_scan) IP protocol sweep
- **122:12** (port_scan) IP distributed protocol scan
- **122:13** (port_scan) IP filtered protocol scan
- **122:14** (port_scan) IP filtered decoy protocol scan
- **122:15** (port_scan) IP filtered protocol sweep
- **122:16** (port_scan) IP filtered distributed protocol scan
- **122:17** (port_scan) UDP portscan
- **122:18** (port_scan) UDP decoy portscan
- **122:19** (port_scan) UDP portsweep
- **122:20** (port_scan) UDP distributed portscan
- **122:21** (port_scan) UDP filtered portscan
- **122:22** (port_scan) UDP filtered decoy portscan
- **122:23** (port_scan) UDP filtered portsweep
- **122:24** (port_scan) UDP filtered distributed portscan
- **122:25** (port_scan) ICMP sweep
- **122:26** (port_scan) ICMP filtered sweep
- **122:27** (port_scan) open port

Peg counts:

- **port_scan.packets**: total packets (sum)

9.30 reg_test

What: The regression test inspector (rti) is used when special packet handling is required for a reg test

Type: inspector

Usage: context

Configuration:

- bool **reg_test.test_daq_retry** = true: test daq packet retry feature

Peg counts:

- **reg_test.packets**: total packets (sum)
- **reg_test.retry_requests**: total retry packets requested (sum)
- **reg_test.retry_packets**: total retried packets received (sum)

9.31 reputation

What: reputation inspection

Type: inspector

Usage: global

Configuration:

- string **reputation.blacklist**: blacklist file name with IP lists
- string **reputation.list_dir**: directory for IP lists and manifest file
- int **reputation.memcap** = 500: maximum total MB of memory allocated { 1:4095 }
- enum **reputation.nested_ip** = inner: IP to use when there is IP encapsulation { inner|outer|all }
- enum **reputation.priority** = whitelist: defines priority when there is a decision conflict during run-time { blacklist|whitelist }
- bool **reputation.scan_local** = false: inspect local address defined in RFC 1918
- enum **reputation.white** = unblack: specify the meaning of whitelist { unblack|trust }
- string **reputation.whitelist**: whitelist file name with IP lists

Rules:

- **136:1** (reputation) packets blacklisted
- **136:2** (reputation) packets whitelisted
- **136:3** (reputation) packets monitored

Peg counts:

- **reputation.packets**: total packets processed (sum)
- **reputation.blacklisted**: number of packets blacklisted (sum)
- **reputation.whitelisted**: number of packets whitelisted (sum)
- **reputation.monitored**: number of packets monitored (sum)
- **reputation.memory_allocated**: total memory allocated (sum)

9.32 rpc_decode

What: RPC inspector

Type: inspector

Usage: inspect

Rules:

- **106:1** (rpc_decode) fragmented RPC records
- **106:2** (rpc_decode) multiple RPC records
- **106:3** (rpc_decode) large RPC record fragment
- **106:4** (rpc_decode) incomplete RPC segment
- **106:5** (rpc_decode) zero-length RPC fragment

Peg counts:

- **rpc_decode.total_packets**: total packets (sum)
- **rpc_decode.concurrent_sessions**: total concurrent rpc sessions (now)
- **rpc_decode.max_concurrent_sessions**: maximum concurrent rpc sessions (max)

9.33 sip

What: sip inspection

Type: inspector

Usage: inspect

Configuration:

- bool **sip.ignore_call_channel** = false: enables the support for ignoring audio/video data channel
- int **sip.max_call_id_len** = 256: maximum call id field size { 0:65535 }
- int **sip.max_contact_len** = 256: maximum contact field size { 0:65535 }
- int **sip.max_content_len** = 1024: maximum content length of the message body { 0:65535 }
- int **sip.max_dialogs** = 4: maximum number of dialogs within one stream session { 1:4194303 }
- int **sip.max_from_len** = 256: maximum from field size { 0:65535 }
- int **sip.max_requestName_len** = 20: maximum request name field size { 0:65535 }
- int **sip.max_to_len** = 256: maximum to field size { 0:65535 }
- int **sip.max_uri_len** = 256: maximum request uri field size { 0:65535 }
- int **sip.max_via_len** = 1024: maximum via field size { 0:65535 }
- string **sip.methods** = invite cancel ack bye register options: list of methods to check in SIP messages

Rules:

- **140:2** (sip) empty request URI
- **140:3** (sip) URI is too long

- **140:4** (sip) empty call-Id
- **140:5** (sip) Call-Id is too long
- **140:6** (sip) CSeq number is too large or negative
- **140:7** (sip) request name in CSeq is too long
- **140:8** (sip) empty From header
- **140:9** (sip) From header is too long
- **140:10** (sip) empty To header
- **140:11** (sip) To header is too long
- **140:12** (sip) empty Via header
- **140:13** (sip) Via header is too long
- **140:14** (sip) empty Contact
- **140:15** (sip) contact is too long
- **140:16** (sip) content length is too large or negative
- **140:17** (sip) multiple SIP messages in a packet
- **140:18** (sip) content length mismatch
- **140:19** (sip) request name is invalid
- **140:20** (sip) Invite replay attack
- **140:21** (sip) illegal session information modification
- **140:22** (sip) response status code is not a 3 digit number
- **140:23** (sip) empty Content-type header
- **140:24** (sip) SIP version is invalid
- **140:25** (sip) mismatch in METHOD of request and the CSEQ header
- **140:26** (sip) method is unknown
- **140:27** (sip) maximum dialogs within a session reached

Peg counts:

- **sip.packets**: total packets (sum)
- **sip.sessions**: total sessions (sum)
- **sip.concurrent_sessions**: total concurrent SIP sessions (now)
- **sip.max_concurrent_sessions**: maximum concurrent SIP sessions (max)
- **sip.events**: events generated (sum)
- **sip.dialogs**: total dialogs (sum)
- **sip.ignored_channels**: total channels ignored (sum)
- **sip.ignored_sessions**: total sessions ignored (sum)
- **sip.total_requests**: total requests (sum)

- **sip.invite**: invite (sum)
- **sip.cancel**: cancel (sum)
- **sip.ack**: ack (sum)
- **sip.bye**: bye (sum)
- **sip.register**: register (sum)
- **sip.options**: options (sum)
- **sip.refer**: refer (sum)
- **sip.subscribe**: subscribe (sum)
- **sip.update**: update (sum)
- **sip.join**: join (sum)
- **sip.info**: info (sum)
- **sip.message**: message (sum)
- **sip.notify**: notify (sum)
- **sip.prack**: prack (sum)
- **sip.total_responses**: total responses (sum)
- **sip.code_1xx**: 1xx (sum)
- **sip.code_2xx**: 2xx (sum)
- **sip.code_3xx**: 3xx (sum)
- **sip.code_4xx**: 4xx (sum)
- **sip.code_5xx**: 5xx (sum)
- **sip.code_6xx**: 6xx (sum)
- **sip.code_7xx**: 7xx (sum)
- **sip.code_8xx**: 8xx (sum)
- **sip.code_9xx**: 9xx (sum)

9.34 smtp

What: smtp inspection

Type: inspector

Usage: inspect

Configuration:

- string **smtp.alt_max_command_line_len[].command**: command string
- int **smtp.alt_max_command_line_len[].length** = 0: specify non-default maximum for command { 0: }
- string **smtp.auth_cmds**: commands that initiate an authentication exchange
- int **smtp.b64_decode_depth** = 1460: depth used to decode the base64 encoded MIME attachments (-1 no limit) { -1:65535 }
- string **smtp.binary_data_cmds**: commands that initiate sending of data and use a length value after the command

- int **smtp.bitenc_decode_depth** = 1460: depth used to extract the non-encoded MIME attachments (-1 no limit) { -1:65535 }
- string **smtp.data_cmds**: commands that initiate sending of data with an end of data delimiter
- int **smtp.email_hdrs_log_depth** = 1464: depth for logging email headers { 0:20480 }
- bool **smtp.ignore_data** = false: ignore data section of mail
- bool **smtp.ignore_tls_data** = false: ignore TLS-encrypted data when processing rules
- string **smtp.invalid_cmds**: alert if this command is sent from client side
- bool **smtp.log_email_hdrs** = false: log the SMTP email headers extracted from SMTP data
- bool **smtp.log_filename** = false: log the MIME attachment filenames extracted from the Content-Disposition header within the MIME body
- bool **smtp.log_mailfrom** = false: log the sender's email address extracted from the MAIL FROM command
- bool **smtp.log_rcptto** = false: log the recipient's email address extracted from the RCPT TO command
- int **smtp.max_auth_command_line_len** = 1000: max auth command Line Length { 0:65535 }
- int **smtp.max_command_line_len** = 0: max Command Line Length { 0:65535 }
- int **smtp.max_header_line_len** = 0: max SMTP DATA header line { 0:65535 }
- int **smtp.max_response_line_len** = 0: max SMTP response line { 0:65535 }
- enum **smtp.normalize** = none: turns on/off normalization { none | cmds | all }
- string **smtp.normalize_cmds**: list of commands to normalize
- int **smtp.qp_decode_depth** = 1460: quoted-Printable decoding depth (-1 no limit) { -1:65535 }
- int **smtp.uu_decode_depth** = 1460: Unix-to-Unix decoding depth (-1 no limit) { -1:65535 }
- string **smtp.valid_cmds**: list of valid commands
- enum **smtp.xlink2state** = alert: enable/disable xlink2state alert { disable | alert | drop }

Rules:

- **124:1** (smtp) attempted command buffer overflow
- **124:2** (smtp) attempted data header buffer overflow
- **124:3** (smtp) attempted response buffer overflow
- **124:4** (smtp) attempted specific command buffer overflow
- **124:5** (smtp) unknown command
- **124:6** (smtp) illegal command
- **124:7** (smtp) attempted header name buffer overflow
- **124:8** (smtp) attempted X-Link2State command buffer overflow
- **124:10** (smtp) base64 decoding failed
- **124:11** (smtp) quoted-printable decoding failed
- **124:13** (smtp) Unix-to-Unix decoding failed
- **124:14** (smtp) Cyrus SASL authentication attack
- **124:15** (smtp) attempted authentication command buffer overflow

Peg counts:

- **smtp.packets**: total packets processed (sum)
- **smtp.sessions**: total smtp sessions (sum)
- **smtp.concurrent_sessions**: total concurrent smtp sessions (now)
- **smtp.max_concurrent_sessions**: maximum concurrent smtp sessions (max)
- **smtp.b64_attachments**: total base64 attachments decoded (sum)
- **smtp.b64_decoded_bytes**: total base64 decoded bytes (sum)
- **smtp.qp_attachments**: total quoted-printable attachments decoded (sum)
- **smtp.qp_decoded_bytes**: total quoted-printable decoded bytes (sum)
- **smtp.uu_attachments**: total uu attachments decoded (sum)
- **smtp.uu_decoded_bytes**: total uu decoded bytes (sum)
- **smtp.non_encoded_attachments**: total non-encoded attachments extracted (sum)
- **smtp.non_encoded_bytes**: total non-encoded extracted bytes (sum)

9.35 ssh

What: ssh inspection

Type: inspector

Usage: inspect

Configuration:

- int **ssh.max_encrypted_packets** = 25: ignore session after this many encrypted packets { 0:65535 }
- int **ssh.max_client_bytes** = 19600: number of unanswered bytes before alerting on challenge-response overflow or CRC32 { 0:65535 }
- int **ssh.max_server_version_len** = 80: limit before alerting on secure CRT server version string overflow { 0:255 }

Rules:

- **128:1** (ssh) challenge-response overflow exploit
- **128:2** (ssh) SSH1 CRC32 exploit
- **128:3** (ssh) server version string overflow
- **128:5** (ssh) bad message direction
- **128:6** (ssh) payload size incorrect for the given payload
- **128:7** (ssh) failed to detect SSH version string

Peg counts:

- **ssh.packets**: total packets (sum)
- **ssh.concurrent_sessions**: total concurrent ssh sessions (now)
- **ssh.max_concurrent_sessions**: maximum concurrent ssh sessions (max)

9.36 ssl

What: ssl inspection

Type: inspector

Usage: inspect

Configuration:

- bool **ssl.trust_servers** = false: disables requirement that application (encrypted) data must be observed on both sides
- int **ssl.max_heartbeat_length** = 0: maximum length of heartbeat record allowed { 0:65535 }

Rules:

- **137:1** (ssl) invalid client HELLO after server HELLO detected
- **137:2** (ssl) invalid server HELLO without client HELLO detected
- **137:3** (ssl) heartbeat read overrun attempt detected
- **137:4** (ssl) large heartbeat response detected

Peg counts:

- **ssl.packets**: total packets processed (sum)
- **ssl.decoded**: ssl packets decoded (sum)
- **ssl.client_hello**: total client hellos (sum)
- **ssl.server_hello**: total server hellos (sum)
- **ssl.certificate**: total ssl certificates (sum)
- **ssl.server_done**: total server done (sum)
- **ssl.client_key_exchange**: total client key exchanges (sum)
- **ssl.server_key_exchange**: total server key exchanges (sum)
- **ssl.change_cipher**: total change cipher records (sum)
- **ssl.finished**: total handshakes finished (sum)
- **ssl.client_application**: total client application records (sum)
- **ssl.server_application**: total server application records (sum)
- **ssl.alert**: total ssl alert records (sum)
- **ssl.unrecognized_records**: total unrecognized records (sum)
- **ssl.handshakes_completed**: total completed ssl handshakes (sum)
- **ssl.bad_handshakes**: total bad handshakes (sum)
- **ssl.sessions_ignored**: total sessions ignore (sum)
- **ssl.detection_disabled**: total detection disabled (sum)
- **ssl.concurrent_sessions**: total concurrent ssl sessions (now)
- **ssl.max_concurrent_sessions**: maximum concurrent ssl sessions (max)

9.37 stream

What: common flow tracking

Type: inspector

Usage: global

Configuration:

- int **stream.footprint** = 0: use zero for production, non-zero for testing at given size (for TCP and user) { 0: }
- bool **stream.ip_frags_only** = false: don't process non-frag flows
- int **stream.ip_cache.max_sessions** = 16384: maximum simultaneous sessions tracked before pruning { 2: }
- int **stream.ip_cache.pruning_timeout** = 30: minimum inactive time before being eligible for pruning { 1: }
- int **stream.ip_cache.idle_timeout** = 180: maximum inactive time before retiring session tracker { 1: }
- int **stream.icmp_cache.max_sessions** = 65536: maximum simultaneous sessions tracked before pruning { 2: }
- int **stream.icmp_cache.pruning_timeout** = 30: minimum inactive time before being eligible for pruning { 1: }
- int **stream.icmp_cache.idle_timeout** = 180: maximum inactive time before retiring session tracker { 1: }
- int **stream.tcp_cache.max_sessions** = 262144: maximum simultaneous sessions tracked before pruning { 2: }
- int **stream.tcp_cache.pruning_timeout** = 30: minimum inactive time before being eligible for pruning { 1: }
- int **stream.tcp_cache.idle_timeout** = 3600: maximum inactive time before retiring session tracker { 1: }
- int **stream.udp_cache.max_sessions** = 131072: maximum simultaneous sessions tracked before pruning { 2: }
- int **stream.udp_cache.pruning_timeout** = 30: minimum inactive time before being eligible for pruning { 1: }
- int **stream.udp_cache.idle_timeout** = 180: maximum inactive time before retiring session tracker { 1: }
- int **stream.user_cache.max_sessions** = 1024: maximum simultaneous sessions tracked before pruning { 2: }
- int **stream.user_cache.pruning_timeout** = 30: minimum inactive time before being eligible for pruning { 1: }
- int **stream.user_cache.idle_timeout** = 180: maximum inactive time before retiring session tracker { 1: }
- int **stream.file_cache.max_sessions** = 128: maximum simultaneous sessions tracked before pruning { 2: }
- int **stream.file_cache.pruning_timeout** = 30: minimum inactive time before being eligible for pruning { 1: }
- int **stream.file_cache.idle_timeout** = 180: maximum inactive time before retiring session tracker { 1: }
- int **stream.trace**: mask for enabling debug traces in module

Rules:

- **135:1** (stream) TCP SYN received
- **135:2** (stream) TCP session established
- **135:3** (stream) TCP session cleared

Peg counts:

- **stream.ip_flows**: total ip sessions (sum)
- **stream.ip_total_prunes**: total ip sessions pruned (sum)

- **stream.ip_idle_prunes**: ip sessions pruned due to timeout (sum)
- **stream.ip_excess_prunes**: ip sessions pruned due to excess (sum)
- **stream.ip_uni_prunes**: ip uni sessions pruned (sum)
- **stream.ip_preemptive_prunes**: ip sessions pruned during preemptive pruning (sum)
- **stream.ip_memcap_prunes**: ip sessions pruned due to memcap (sum)
- **stream.ip_ha_prunes**: ip sessions pruned by high availability sync (sum)
- **stream.icmp_flows**: total icmp sessions (sum)
- **stream.icmp_total_prunes**: total icmp sessions pruned (sum)
- **stream.icmp_idle_prunes**: icmp sessions pruned due to timeout (sum)
- **stream.icmp_excess_prunes**: icmp sessions pruned due to excess (sum)
- **stream.icmp_uni_prunes**: icmp uni sessions pruned (sum)
- **stream.icmp_preemptive_prunes**: icmp sessions pruned during preemptive pruning (sum)
- **stream.icmp_memcap_prunes**: icmp sessions pruned due to memcap (sum)
- **stream.icmp_ha_prunes**: icmp sessions pruned by high availability sync (sum)
- **stream.tcp_flows**: total tcp sessions (sum)
- **stream.tcp_total_prunes**: total tcp sessions pruned (sum)
- **stream.tcp_idle_prunes**: tcp sessions pruned due to timeout (sum)
- **stream.tcp_excess_prunes**: tcp sessions pruned due to excess (sum)
- **stream.tcp_uni_prunes**: tcp uni sessions pruned (sum)
- **stream.tcp_preemptive_prunes**: tcp sessions pruned during preemptive pruning (sum)
- **stream.tcp_memcap_prunes**: tcp sessions pruned due to memcap (sum)
- **stream.tcp_ha_prunes**: tcp sessions pruned by high availability sync (sum)
- **stream.udp_flows**: total udp sessions (sum)
- **stream.udp_total_prunes**: total udp sessions pruned (sum)
- **stream.udp_idle_prunes**: udp sessions pruned due to timeout (sum)
- **stream.udp_excess_prunes**: udp sessions pruned due to excess (sum)
- **stream.udp_uni_prunes**: udp uni sessions pruned (sum)
- **stream.udp_preemptive_prunes**: udp sessions pruned during preemptive pruning (sum)
- **stream.udp_memcap_prunes**: udp sessions pruned due to memcap (sum)
- **stream.udp_ha_prunes**: udp sessions pruned by high availability sync (sum)
- **stream.user_flows**: total user sessions (sum)
- **stream.user_total_prunes**: total user sessions pruned (sum)
- **stream.user_idle_prunes**: user sessions pruned due to timeout (sum)
- **stream.user_excess_prunes**: user sessions pruned due to excess (sum)
- **stream.user_uni_prunes**: user uni sessions pruned (sum)

- **stream.user_preemptive_prunes**: user sessions pruned during preemptive pruning (sum)
- **stream.user_memcap_prunes**: user sessions pruned due to memcap (sum)
- **stream.user_ha_prunes**: user sessions pruned by high availability sync (sum)
- **stream.file_flows**: total file sessions (sum)
- **stream.file_total_prunes**: total file sessions pruned (sum)
- **stream.file_idle_prunes**: file sessions pruned due to timeout (sum)
- **stream.file_excess_prunes**: file sessions pruned due to excess (sum)
- **stream.file_uni_prunes**: file uni sessions pruned (sum)
- **stream.file_preemptive_prunes**: file sessions pruned during preemptive pruning (sum)
- **stream.file_memcap_prunes**: file sessions pruned due to memcap (sum)
- **stream.file_ha_prunes**: file sessions pruned by high availability sync (sum)

9.38 stream_file

What: stream inspector for file flow tracking and processing

Type: inspector

Usage: inspect

Configuration:

- bool **stream_file.upload** = false: indicate file transfer direction

9.39 stream_icmp

What: stream inspector for ICMP flow tracking

Type: inspector

Usage: inspect

Configuration:

- int **stream_icmp.session_timeout** = 30: session tracking timeout { 1:86400 }

Peg counts:

- **stream_icmp.sessions**: total icmp sessions (sum)
- **stream_icmp.max**: max icmp sessions (max)
- **stream_icmp.created**: icmp session trackers created (sum)
- **stream_icmp.released**: icmp session trackers released (sum)
- **stream_icmp.timeouts**: icmp session timeouts (sum)
- **stream_icmp.prunes**: icmp session prunes (sum)

9.40 stream_ip

What: stream inspector for IP flow tracking and defragmentation

Type: inspector

Usage: inspect

Configuration:

- int **stream_ip.max_frags** = 8192: maximum number of simultaneous fragments being tracked { 1: }
- int **stream_ip.max_overlaps** = 0: maximum allowed overlaps per datagram; 0 is unlimited { 0: }
- int **stream_ip.min_frag_length** = 0: alert if fragment length is below this limit before or after trimming { 0: }
- int **stream_ip.min_ttl** = 1: discard fragments with TTL below the minimum { 1:255 }
- enum **stream_ip.policy** = linux: fragment reassembly policy { first | linux | bsd | bsd_right | last | windows | solaris }
- int **stream_ip.session_timeout** = 30: session tracking timeout { 1:86400 }
- int **stream_ip.trace**: mask for enabling debug traces in module

Rules:

- **123:1** (stream_ip) inconsistent IP options on fragmented packets
- **123:2** (stream_ip) teardrop attack
- **123:3** (stream_ip) short fragment, possible DOS attempt
- **123:4** (stream_ip) fragment packet ends after defragmented packet
- **123:5** (stream_ip) zero-byte fragment packet
- **123:6** (stream_ip) bad fragment size, packet size is negative
- **123:7** (stream_ip) bad fragment size, packet size is greater than 65536
- **123:8** (stream_ip) fragmentation overlap
- **123:11** (stream_ip) TTL value less than configured minimum, not using for reassembly
- **123:12** (stream_ip) excessive fragment overlap
- **123:13** (stream_ip) tiny fragment

Peg counts:

- **stream_ip.sessions**: total ip sessions (sum)
- **stream_ip.max**: max ip sessions (max)
- **stream_ip.created**: ip session trackers created (sum)
- **stream_ip.released**: ip session trackers released (sum)
- **stream_ip.timeouts**: ip session timeouts (sum)
- **stream_ip.prunes**: ip session prunes (sum)
- **stream_ip.total_frags**: total fragments (sum)
- **stream_ip.current_frags**: current fragments (now)

- **stream_ip.max_frags**: max fragments (sum)
- **stream_ip.reassembled**: reassembled datagrams (sum)
- **stream_ip.discards**: fragments discarded (sum)
- **stream_ip.frag_timeouts**: datagrams abandoned (sum)
- **stream_ip.overlaps**: overlapping fragments (sum)
- **stream_ip.anomalies**: anomalies detected (sum)
- **stream_ip.alerts**: alerts generated (sum)
- **stream_ip.drops**: fragments dropped (sum)
- **stream_ip.trackers_added**: datagram trackers created (sum)
- **stream_ip.trackers_freed**: datagram trackers released (sum)
- **stream_ip.trackers_cleared**: datagram trackers cleared (sum)
- **stream_ip.trackers_completed**: datagram trackers completed (sum)
- **stream_ip.nodes_inserted**: fragments added to tracker (sum)
- **stream_ip.nodes_deleted**: fragments deleted from tracker (sum)
- **stream_ip.reassembled_bytes**: total reassembled bytes (sum)
- **stream_ip.fragmented_bytes**: total fragmented bytes (sum)

9.41 stream_tcp

What: stream inspector for TCP flow tracking and stream normalization and reassembly

Type: inspector

Usage: inspect

Configuration:

- int **stream_tcp.flush_factor** = 0: flush upon seeing a drop in segment size after given number of non-decreasing segments { 0: }
- int **stream_tcp.max_window** = 0: maximum allowed TCP window { 0:1073725440 }
- int **stream_tcp.overlap_limit** = 0: maximum number of allowed overlapping segments per session { 0:255 }
- int **stream_tcp.max_pdu** = 16384: maximum reassembled PDU size { 1460:32768 }
- enum **stream_tcp.policy** = bsd: determines operating system characteristics like reassembly { first | last | linux | old_linux | bsd | macos | solaris | irix | hpux11 | hpux10 | windows | win_2003 | vista | proxy }
- bool **stream_tcp.reassemble_async** = true: queue data for reassembly before traffic is seen in both directions
- int **stream_tcp.require_3whs** = -1: don't track midstream sessions after given seconds from start up; -1 tracks all { -1:86400 }
- bool **stream_tcp.show_rebuilt_packets** = false: enable cmg like output of reassembled packets
- int **stream_tcp.queue_limit.max_bytes** = 1048576: don't queue more than given bytes per session and direction { 0: }
- int **stream_tcp.queue_limit.max_segments** = 2621: don't queue more than given segments per session and direction { 0: }
- int **stream_tcp.small_segments.count** = 0: limit number of small segments queued { 0:2048 }

- int **stream_tcp.small_segments.maximum_size** = 0: limit number of small segments queued { 0:2048 }
- int **stream_tcp.session_timeout** = 30: session tracking timeout { 1:86400 }

Rules:

- **129:1** (stream_tcp) SYN on established session
- **129:2** (stream_tcp) data on SYN packet
- **129:3** (stream_tcp) data sent on stream not accepting data
- **129:4** (stream_tcp) TCP timestamp is outside of PAWS window
- **129:5** (stream_tcp) bad segment, adjusted size ⇐ 0 (deprecated)
- **129:6** (stream_tcp) window size (after scaling) larger than policy allows
- **129:7** (stream_tcp) limit on number of overlapping TCP packets reached
- **129:8** (stream_tcp) data sent on stream after TCP reset sent
- **129:9** (stream_tcp) TCP client possibly hijacked, different ethernet address
- **129:10** (stream_tcp) TCP server possibly hijacked, different ethernet address
- **129:11** (stream_tcp) TCP data with no TCP flags set
- **129:12** (stream_tcp) consecutive TCP small segments exceeding threshold
- **129:13** (stream_tcp) 4-way handshake detected
- **129:14** (stream_tcp) TCP timestamp is missing
- **129:15** (stream_tcp) reset outside window
- **129:16** (stream_tcp) FIN number is greater than prior FIN
- **129:17** (stream_tcp) ACK number is greater than prior FIN
- **129:18** (stream_tcp) data sent on stream after TCP reset received
- **129:19** (stream_tcp) TCP window closed before receiving data
- **129:20** (stream_tcp) TCP session without 3-way handshake

Peg counts:

- **stream_tcp.sessions**: total tcp sessions (sum)
- **stream_tcp.max**: max tcp sessions (max)
- **stream_tcp.created**: tcp session trackers created (sum)
- **stream_tcp.released**: tcp session trackers released (sum)
- **stream_tcp.timeouts**: tcp session timeouts (sum)
- **stream_tcp.prunes**: tcp session prunes (sum)
- **stream_tcp.instantiated**: new sessions instantiated (sum)
- **stream_tcp.setups**: session initializations (sum)
- **stream_tcp.restarts**: sessions restarted (sum)
- **stream_tcp.resyns**: SYN received on established session (sum)

- **stream_tcp.discards**: tcp packets discarded (sum)
- **stream_tcp.events**: events generated (sum)
- **stream_tcp.ignored**: tcp packets ignored (sum)
- **stream_tcp.untracked**: tcp packets not tracked (sum)
- **stream_tcp.syn_trackers**: tcp session tracking started on syn (sum)
- **stream_tcp.syn_ack_trackers**: tcp session tracking started on syn-ack (sum)
- **stream_tcp.three_way_trackers**: tcp session tracking started on ack (sum)
- **stream_tcp.data_trackers**: tcp session tracking started on data (sum)
- **stream_tcp.segs_queued**: total segments queued (sum)
- **stream_tcp.segs_released**: total segments released (sum)
- **stream_tcp.segs_split**: tcp segments split when reassembling PDUs (sum)
- **stream_tcp.segs_used**: queued tcp segments applied to reassembled PDUs (sum)
- **stream_tcp.rebuilt_packets**: total reassembled PDUs (sum)
- **stream_tcp.rebuilt_buffers**: rebuilt PDU sections (sum)
- **stream_tcp.rebuilt_bytes**: total rebuilt bytes (sum)
- **stream_tcp.overlaps**: overlapping segments queued (sum)
- **stream_tcp.gaps**: missing data between PDUs (sum)
- **stream_tcp.exceeded_max_segs**: number of times the maximum queued segment limit was reached (sum)
- **stream_tcp.exceeded_max_bytes**: number of times the maximum queued byte limit was reached (sum)
- **stream_tcp.internal_events**: 135:X events generated (sum)
- **stream_tcp.client_cleanups**: number of times data from server was flushed when session released (sum)
- **stream_tcp.server_cleanups**: number of times data from client was flushed when session released (sum)
- **stream_tcp.memory**: current memory in use (now)
- **stream_tcp.initializing**: number of sessions currently initializing (now)
- **stream_tcp.established**: number of sessions currently established (now)
- **stream_tcp.closing**: number of sessions currently closing (now)
- **stream_tcp.syns**: number of syn packets (sum)
- **stream_tcp.syn_acks**: number of syn-ack packets (sum)
- **stream_tcp.resets**: number of reset packets (sum)
- **stream_tcp.fins**: number of fin packets (sum)

9.42 stream_udp

What: stream inspector for UDP flow tracking

Type: inspector

Usage: inspect

Configuration:

- int **stream_udp.session_timeout** = 30: session tracking timeout { 1:86400 }

Peg counts:

- **stream_udp.sessions**: total udp sessions (sum)
- **stream_udp.max**: max udp sessions (max)
- **stream_udp.created**: udp session trackers created (sum)
- **stream_udp.released**: udp session trackers released (sum)
- **stream_udp.timeouts**: udp session timeouts (sum)
- **stream_udp.prunes**: udp session prunes (sum)
- **stream_udp.ignored**: udp packets ignored (sum)

9.43 stream_user

What: stream inspector for user flow tracking and reassembly

Type: inspector

Usage: inspect

Configuration:

- int **stream_user.session_timeout** = 30: session tracking timeout { 1:86400 }
- int **stream_user.trace**: mask for enabling debug traces in module

9.44 telnet

What: telnet inspection and normalization

Type: inspector

Usage: inspect

Configuration:

- int **telnet.ayt_attack_thresh** = -1: alert on this number of consecutive Telnet AYT commands { -1: }
- bool **telnet.check_encrypted** = false: check for end of encryption
- bool **telnet.encrypted_traffic** = false: check for encrypted Telnet and FTP
- bool **telnet.normalize** = false: eliminate escape sequences

Rules:

- **126:1** (telnet) consecutive Telnet AYT commands beyond threshold

- **126:2** (telnet) Telnet traffic encrypted
- **126:3** (telnet) Telnet subnegotiation begin command without subnegotiation end

Peg counts:

- **telnet.total_packets**: total packets (sum)
- **telnet.concurrent_sessions**: total concurrent Telnet sessions (now)
- **telnet.max_concurrent_sessions**: maximum concurrent Telnet sessions (max)

9.45 wizard

What: inspector that implements port-independent protocol identification

Type: inspector

Usage: inspect

Configuration:

- string **wizard.hexes[].service**: name of service
- select **wizard.hexes[].proto** = tcp: protocol to scan { tcp | udp }
- bool **wizard.hexes[].client_first** = true: which end initiates data transfer
- string **wizard.hexes[].to_server[].hex**: sequence of data with wild chars (?)
- string **wizard.hexes[].to_client[].hex**: sequence of data with wild chars (?)
- string **wizard.spells[].service**: name of service
- select **wizard.spells[].proto** = tcp: protocol to scan { tcp | udp }
- bool **wizard.spells[].client_first** = true: which end initiates data transfer
- string **wizard.spells[].to_server[].spell**: sequence of data with wild cards (*)
- string **wizard.spells[].to_client[].spell**: sequence of data with wild cards (*)
- multi **wizard.curses**: enable service identification based on internal algorithm { dce_smb | dce_udp | dce_tcp }

Peg counts:

- **wizard.tcp_scans**: tcp payload scans (sum)
- **wizard.tcp_hits**: tcp identifications (sum)
- **wizard.udp_scans**: udp payload scans (sum)
- **wizard.udp_hits**: udp identifications (sum)
- **wizard.user_scans**: user payload scans (sum)
- **wizard.user_hits**: user identifications (sum)

10 IPS Action Modules

IPS actions allow you to perform custom actions when events are generated. Unlike loggers, these are invoked before thresholding and can be used to control external agents.

Externally defined actions must be configured to become available to the parser. For the reject rule, you can set reject = { } to get the rule to parse.

10.1 react

What: send response to client and terminate session

Type: ips_action

Usage: detect

Configuration:

- bool **react.msg** = false: use rule msg in response page instead of default message
- string **react.page**: file containing HTTP response (headers and body)

10.2 reject

What: terminate session with TCP reset or ICMP unreachable

Type: ips_action

Usage: detect

Configuration:

- enum **reject.reset**: send TCP reset to one or both ends { source|dest|both }
- enum **reject.control**: send ICMP unreachable(s) { network|host|port|all }

10.3 rewrite

What: overwrite packet contents

Type: ips_action

Usage: detect

Configuration:

- bool **rewrite.disable_replace** = false: disable replace of packet contents with rewrite rules

11 IPS Option Modules

IPS options are the building blocks of IPS rules.

11.1 ack

What: rule option to match on TCP ack numbers

Type: ips_option

Usage: detect

Configuration:

- interval **ack.~range**: check if TCP ack value is *value* | *min*<>*max* | <*max* | >*min* { 0: }

11.2 appids

What: detection option for application ids

Type: ips_option

Usage: detect

Configuration:

- string **appids.~**: comma separated list of application names

11.3 asn1

What: rule option for asn1 detection

Type: ips_option

Usage: detect

Configuration:

- implied **asn1.bitstring_overflow**: detects invalid bitstring encodings that are known to be remotely exploitable
- implied **asn1.double_overflow**: detects a double ASCII encoding that is larger than a standard buffer
- implied **asn1.print**: dump decode data to console; always true
- int **asn1.oversize_length**: compares ASN.1 type lengths with the supplied argument { 0: }
- int **asn1.absolute_offset**: absolute offset from the beginning of the packet { 0: }
- int **asn1.relative_offset**: relative offset from the cursor

11.4 base64_decode

What: rule option to decode base64 data - must be used with base64_data option

Type: ips_option

Usage: detect

Configuration:

- int **base64_decode.bytes**: number of base64 encoded bytes to decode { 1: }
- int **base64_decode.offset** = 0: bytes past start of buffer to start decoding { 0: }
- implied **base64_decode.relative**: apply offset to cursor instead of start of buffer

11.5 bufferlen

What: rule option to check length of current buffer

Type: ips_option

Usage: detect

Configuration:

- interval **bufferlen.~range**: check that length of current buffer is in given range { 0:65535 }

11.6 byte_extract

What: rule option to convert data to an integer variable

Type: ips_option

Usage: detect

Configuration:

- int **byte_extract.~count**: number of bytes to pick up from the buffer { 1:10 }
- int **byte_extract.~offset**: number of bytes into the buffer to start processing { -65535:65535 }
- string **byte_extract.~name**: name of the variable that will be used in other rule options
- implied **byte_extract.relative**: offset from cursor instead of start of buffer
- int **byte_extract.multiplier** = 1: scale extracted value by given amount { 1:65535 }
- int **byte_extract.align** = 0: round the number of converted bytes up to the next 2- or 4-byte boundary { 0:4 }
- implied **byte_extract.big**: big endian
- implied **byte_extract.little**: little endian
- implied **byte_extract.dce**: dcerpc2 determines endianness
- implied **byte_extract.string**: convert from string
- implied **byte_extract.hex**: convert from hex string
- implied **byte_extract.oct**: convert from octal string
- implied **byte_extract.dec**: convert from decimal string
- int **byte_extract.bitmask**: applies as an AND to the extracted value before storage in *name* { 0x1:0xFFFFFFFF }

11.7 byte_jump

What: rule option to move the detection cursor

Type: ips_option

Usage: detect

Configuration:

- int **byte_jump.~count**: number of bytes to pick up from the buffer { 0:10 }
- string **byte_jump.~offset**: variable name or number of bytes into the buffer to start processing
- implied **byte_jump.relative**: offset from cursor instead of start of buffer
- implied **byte_jump.from_beginning**: jump from start of buffer instead of cursor
- implied **byte_jump.from_end**: jump backward from end of buffer
- int **byte_jump.multiplier** = 1: scale extracted value by given amount { 1:65535 }
- int **byte_jump.align** = 0: round the number of converted bytes up to the next 2- or 4-byte boundary { 0:4 }
- string **byte_jump.post_offset**: skip forward or backward (positive or negative value) by variable name or number of bytes after the other jump options have been applied
- implied **byte_jump.big**: big endian

- implied **byte_jump.little**: little endian
- implied **byte_jump.dce**: dcerpc2 determines endianness
- implied **byte_jump.string**: convert from string
- implied **byte_jump.hex**: convert from hex string
- implied **byte_jump.oct**: convert from octal string
- implied **byte_jump.dec**: convert from decimal string
- int **byte_jump.bitmask**: applies as an AND prior to evaluation { 0x1:0xFFFFFFFF }

11.8 byte_math

What: rule option to perform mathematical operations on extracted value and a specified value or existing variable

Type: ips_option

Usage: detect

Configuration:

- int **byte_math.bytes**: number of bytes to pick up from the buffer { 1:10 }
- string **byte_math.offset**: number of bytes into the buffer to start processing
- enum **byte_math.oper**: mathematical operation to perform { +|-|*|/|<<|>> }
- string **byte_math.rvalue**: value to use mathematical operation against
- string **byte_math.result**: name of the variable to store the result
- implied **byte_math.relative**: offset from cursor instead of start of buffer
- enum **byte_math.endian**: specify big/little endian { big|little }
- implied **byte_math.dce**: dcerpc2 determines endianness
- enum **byte_math.string**: convert extracted string to dec/hex/oct { hex|dec|oct }
- int **byte_math.bitmask**: applies as bitwise AND to the extracted value before storage in *name* { 0x1:0xFFFFFFFF }

11.9 byte_test

What: rule option to convert data to integer and compare

Type: ips_option

Usage: detect

Configuration:

- int **byte_test.~count**: number of bytes to pick up from the buffer { 1:10 }
- string **byte_test.~operator**: operation to perform to test the value
- string **byte_test.~compare**: variable name or value to test the converted result against
- string **byte_test.~offset**: variable name or number of bytes into the payload to start processing
- implied **byte_test.relative**: offset from cursor instead of start of buffer
- implied **byte_test.big**: big endian

- implied **byte_test.little**: little endian
- implied **byte_test.dce**: dcerpc2 determines endianness
- implied **byte_test.string**: convert from string
- implied **byte_test.hex**: convert from hex string
- implied **byte_test.oct**: convert from octal string
- implied **byte_test.dec**: convert from decimal string
- int **byte_test.bitmask**: applies as an AND prior to evaluation { 0x1:0xFFFFFFFF }

11.10 classtype

What: general rule option for rule classification

Type: ips_option

Usage: detect

Configuration:

- string **classtype.~**: classification for this rule

11.11 content

What: payload rule option for basic pattern matching

Type: ips_option

Usage: detect

Configuration:

- string **content.~data**: data to match
- implied **content.nocase**: case insensitive match
- implied **content.fast_pattern**: use this content in the fast pattern matcher instead of the content selected by default
- int **content.fast_pattern_offset** = 0: number of leading characters of this content the fast pattern matcher should exclude { 0: }
- int **content.fast_pattern_length**: maximum number of characters from this content the fast pattern matcher should use { 1: }
- string **content.offset**: var or number of bytes from start of buffer to start search
- string **content.depth**: var or maximum number of bytes to search from beginning of buffer
- string **content.distance**: var or number of bytes from cursor to start search
- string **content.within**: var or maximum number of bytes to search from cursor

11.12 cvs

What: payload rule option for detecting specific attacks

Type: ips_option

Usage: detect

Configuration:

- implied **cvs.invalid-entry**: looks for an invalid Entry string

11.13 dce_iface

What: detection option to check dcerpc interface

Type: ips_option

Usage: detect

Configuration:

- string **dce_iface.uuid**: match given dcerpc uuid
- interval **dce_iface.version**: interface version { 0: }
- implied **dce_iface.any_frag**: match on any fragment

11.14 dce_opnum

What: detection option to check dcerpc operation number

Type: ips_option

Usage: detect

Configuration:

- string **dce_opnum.~**: match given dcerpc operation number, range or list

11.15 dce_stub_data

What: sets the cursor to dcerpc stub data

Type: ips_option

Usage: detect

11.16 detection_filter

What: rule option to require multiple hits before a rule generates an event

Type: ips_option

Usage: detect

Configuration:

- enum **detection_filter.track**: track hits by source or destination IP address { by_src | by_dst }
- int **detection_filter.count**: hits in interval before allowing the rule to fire { 1: }
- int **detection_filter.seconds**: length of interval to count hits { 1: }

11.17 dnp3_data

What: sets the cursor to dnp3 data

Type: ips_option

Usage: detect

11.18 dnp3_func

What: detection option to check DNP3 function code

Type: ips_option

Usage: detect

Configuration:

- string **dnp3_func.~**: match DNP3 function code or name

11.19 dnp3_ind

What: detection option to check DNP3 indicator flags

Type: ips_option

Usage: detect

Configuration:

- string **dnp3_ind.~**: match given DNP3 indicator flags

11.20 dnp3_obj

What: detection option to check DNP3 object headers

Type: ips_option

Usage: detect

Configuration:

- int **dnp3_obj.group** = 0: match given DNP3 object header group { 0:255 }
- int **dnp3_obj.var** = 0: match given DNP3 object header var { 0:255 }

11.21 dsize

What: rule option to test payload size

Type: ips_option

Usage: detect

Configuration:

- interval **dsize.~range**: check if packet payload size is in the given range { 0:65535 }

11.22 file_data

What: rule option to set detection cursor to file data

Type: ips_option

Usage: detect

11.23 file_type

What: rule option to check file type

Type: ips_option

Usage: detect

Configuration:

- string **file_type.~**: list of file type IDs to match

11.24 flags

What: rule option to test TCP control flags

Type: ips_option

Usage: detect

Configuration:

- string **flags.~test_flags**: these flags are tested
- string **flags.~mask_flags**: these flags are don't cares

11.25 flow

What: rule option to check session properties

Type: ips_option

Usage: detect

Configuration:

- implied **flow.to_client**: match on server responses
- implied **flow.to_server**: match on client requests
- implied **flow.from_client**: same as to_server
- implied **flow.from_server**: same as to_client
- implied **flow.established**: match only during data transfer phase
- implied **flow.not_established**: match only outside data transfer phase
- implied **flow.stateless**: match regardless of stream state
- implied **flow.no_stream**: match on raw packets only
- implied **flow.only_stream**: match on reassembled packets only
- implied **flow.no_frag**: match on raw packets only
- implied **flow.only_frag**: match on defragmented packets only

11.26 flowbits

What: rule option to set and test arbitrary boolean flags

Type: ips_option

Usage: detect

Configuration:

- string **flowbits.~command**: set|reset|isset|etc.
- string **flowbits.~arg1**: bits or group
- string **flowbits.~arg2**: group if arg1 is bits

11.27 fragbits

What: rule option to test IP frag flags

Type: ips_option

Usage: detect

Configuration:

- string **fragbits.~flags**: these flags are tested

11.28 fragoffset

What: rule option to test IP frag offset

Type: ips_option

Usage: detect

Configuration:

- interval **fragoffset.~range**: check if ip fragment offset is in given range { 0:8192 }

11.29 gid

What: rule option specifying rule generator

Type: ips_option

Usage: detect

Configuration:

- int **gid.~**: generator id { 1: }

11.30 gtp_info

What: rule option to check gtp info element

Type: ips_option

Usage: detect

Configuration:

- string **gtp_info.~**: info element to match

11.31 gtp_type

What: rule option to check gtp types

Type: ips_option

Usage: detect

Configuration:

- string **gtp_type.~**: list of types to match

11.32 gtp_version

What: rule option to check GTP version

Type: ips_option

Usage: detect

Configuration:

- int **gtp_version.~**: version to match { 0:2 }

11.33 http2_frame_data

What: rule option to see HTTP/2 frame body

Type: ips_option

Usage: detect

11.34 http2_frame_header

What: rule option to see 9-octet HTTP/2 frame header

Type: ips_option

Usage: detect

11.35 http_client_body

What: rule option to set the detection cursor to the request body

Type: ips_option

Usage: detect

11.36 http_cookie

What: rule option to set the detection cursor to the HTTP cookie

Type: ips_option

Usage: detect

Configuration:

- implied **http_cookie.request**: match against the cookie from the request message even when examining the response
- implied **http_cookie.with_body**: parts of this rule examine HTTP message body
- implied **http_cookie.with_trailer**: parts of this rule examine HTTP message trailers

11.37 http_header

What: rule option to set the detection cursor to the normalized headers

Type: ips_option

Usage: detect

Configuration:

- string **http_header.field**: restrict to given header. Header name is case insensitive.
- implied **http_header.request**: match against the headers from the request message even when examining the response
- implied **http_header.with_body**: parts of this rule examine HTTP message body
- implied **http_header.with_trailer**: parts of this rule examine HTTP message trailers

11.38 http_method

What: rule option to set the detection cursor to the HTTP request method

Type: ips_option

Usage: detect

Configuration:

- implied **http_method.with_body**: parts of this rule examine HTTP message body
- implied **http_method.with_trailer**: parts of this rule examine HTTP message trailers

11.39 http_raw_body

What: rule option to set the detection cursor to the unnormalized message body

Type: ips_option

Usage: detect

11.40 http_raw_cookie

What: rule option to set the detection cursor to the unnormalized cookie

Type: ips_option

Usage: detect

Configuration:

- implied **http_raw_cookie.request**: match against the cookie from the request message even when examining the response
- implied **http_raw_cookie.with_body**: parts of this rule examine HTTP message body
- implied **http_raw_cookie.with_trailer**: parts of this rule examine HTTP message trailers

11.41 http_raw_header

What: rule option to set the detection cursor to the unnormalized headers

Type: ips_option

Usage: detect

Configuration:

- implied **http_raw_header.request**: match against the headers from the request message even when examining the response
- implied **http_raw_header.with_body**: parts of this rule examine HTTP message body
- implied **http_raw_header.with_trailer**: parts of this rule examine HTTP message trailers

11.42 http_raw_request

What: rule option to set the detection cursor to the unnormalized request line

Type: ips_option

Usage: detect

Configuration:

- implied **http_raw_request.with_body**: parts of this rule examine HTTP message body
- implied **http_raw_request.with_trailer**: parts of this rule examine HTTP message trailers

11.43 http_raw_status

What: rule option to set the detection cursor to the unnormalized status line

Type: ips_option

Usage: detect

Configuration:

- implied **http_raw_status.with_body**: parts of this rule examine HTTP message body
- implied **http_raw_status.with_trailer**: parts of this rule examine HTTP message trailers

11.44 http_raw_trailer

What: rule option to set the detection cursor to the unnormalized trailers

Type: ips_option

Usage: detect

Configuration:

- implied **http_raw_trailer.request**: match against the trailers from the request message even when examining the response
- implied **http_raw_trailer.with_header**: parts of this rule examine HTTP response message headers (must be combined with request)
- implied **http_raw_trailer.with_body**: parts of this rule examine HTTP response message body (must be combined with request)

11.45 http_raw_uri

What: rule option to set the detection cursor to the unnormalized URI

Type: ips_option

Usage: detect

Configuration:

- implied **http_raw_uri.with_body**: parts of this rule examine HTTP message body
- implied **http_raw_uri.with_trailer**: parts of this rule examine HTTP message trailers
- implied **http_raw_uri.scheme**: match against scheme section of URI only
- implied **http_raw_uri.host**: match against host section of URI only
- implied **http_raw_uri.port**: match against port section of URI only
- implied **http_raw_uri.path**: match against path section of URI only
- implied **http_raw_uri.query**: match against query section of URI only
- implied **http_raw_uri.fragment**: match against fragment section of URI only

11.46 http_stat_code

What: rule option to set the detection cursor to the HTTP status code

Type: ips_option

Usage: detect

Configuration:

- implied **http_stat_code.with_body**: parts of this rule examine HTTP message body
- implied **http_stat_code.with_trailer**: parts of this rule examine HTTP message trailers

11.47 http_stat_msg

What: rule option to set the detection cursor to the HTTP status message

Type: ips_option

Usage: detect

Configuration:

- implied **http_stat_msg.with_body**: parts of this rule examine HTTP message body
- implied **http_stat_msg.with_trailer**: parts of this rule examine HTTP message trailers

11.48 http_trailer

What: rule option to set the detection cursor to the normalized trailers

Type: ips_option

Usage: detect

Configuration:

- string **http_trailer.field**: restrict to given trailer
- implied **http_trailer.request**: match against the trailers from the request message even when examining the response
- implied **http_trailer.with_header**: parts of this rule examine HTTP response message headers (must be combined with request)
- implied **http_trailer.with_body**: parts of this rule examine HTTP message body (must be combined with request)

11.49 http_true_ip

What: rule option to set the detection cursor to the final client IP address

Type: ips_option

Usage: detect

Configuration:

- implied **http_true_ip.with_body**: parts of this rule examine HTTP message body
- implied **http_true_ip.with_trailer**: parts of this rule examine HTTP message trailers

11.50 http_uri

What: rule option to set the detection cursor to the normalized URI buffer

Type: ips_option

Usage: detect

Configuration:

- implied **http_uri.with_body**: parts of this rule examine HTTP message body
- implied **http_uri.with_trailer**: parts of this rule examine HTTP message trailers
- implied **http_uri.scheme**: match against scheme section of URI only
- implied **http_uri.host**: match against host section of URI only
- implied **http_uri.port**: match against port section of URI only
- implied **http_uri.path**: match against path section of URI only
- implied **http_uri.query**: match against query section of URI only
- implied **http_uri.fragment**: match against fragment section of URI only

11.51 http_version

What: rule option to set the detection cursor to the version buffer

Type: ips_option

Usage: detect

Configuration:

- implied **http_version.request**: match against the version from the request message even when examining the response
- implied **http_version.with_body**: parts of this rule examine HTTP message body
- implied **http_version.with_trailer**: parts of this rule examine HTTP message trailers

11.52 icmp_id

What: rule option to check ICMP ID

Type: ips_option

Usage: detect

Configuration:

- interval **icmp_id.~range**: check if ICMP ID is in given range { 0:65535 }

11.53 icmp_seq

What: rule option to check ICMP sequence number

Type: ips_option

Usage: detect

Configuration:

- interval **icmp_seq.~range**: check if ICMP sequence number is in given range { 0:65535 }

11.54 icode

What: rule option to check ICMP code

Type: ips_option

Usage: detect

Configuration:

- interval **icode.~range**: check if ICMP code is in given range is { 0:255 }

11.55 id

What: rule option to check the IP ID field

Type: ips_option

Usage: detect

Configuration:

- interval **id.~range**: check if the IP ID is in the given range { 0: }

11.56 ip_proto

What: rule option to check the IP protocol number

Type: ips_option

Usage: detect

Configuration:

- string **ip_proto.~proto**: [!|>|<] name or number

11.57 ipopts

What: rule option to check for IP options

Type: ips_option

Usage: detect

Configuration:

- select **ipopts.~opt**: output format { rr|eol|nop|ts|sec|esec|lsrr|lsrre|ssrr|satid|any }

11.58 isdataat

What: rule option to check for the presence of payload data

Type: ips_option

Usage: detect

Configuration:

- string **isdataat.~length**: num | !num
- implied **isdataat.relative**: offset from cursor instead of start of buffer

11.59 itype

What: rule option to check ICMP type

Type: ips_option

Usage: detect

Configuration:

- interval **itype.~range**: check if ICMP type is in given range { 0:255 }

11.60 md5

What: payload rule option for hash matching

Type: ips_option

Usage: detect

Configuration:

- string **md5.~hash**: data to match
- int **md5.length**: number of octets in plain text { 1:65535 }
- string **md5.offset**: var or number of bytes from start of buffer to start search
- implied **md5.relative** = false: offset from cursor instead of start of buffer

11.61 metadata

What: rule option for conveying arbitrary name, value data within the rule text

Type: ips_option

Usage: detect

Configuration:

- string **metadata.***: comma-separated list of arbitrary name value pairs

11.62 modbus_data

What: rule option to set cursor to modbus data

Type: ips_option

Usage: detect

11.63 modbus_func

What: rule option to check modbus function code

Type: ips_option

Usage: detect

Configuration:

- string **modbus_func.~**: function code to match

11.64 modbus_unit

What: rule option to check Modbus unit ID

Type: ips_option

Usage: detect

Configuration:

- int **modbus_unit.~**: Modbus unit ID { 0:255 }

11.65 msg

What: rule option summarizing rule purpose output with events

Type: ips_option

Usage: detect

Configuration:

- string **msg.~**: message describing rule

11.66 mss

What: detection for TCP maximum segment size

Type: ips_option

Usage: detect

Configuration:

- interval **mss.~range**: check if TCP MSS is in given range { 0:65535 }

11.67 pcre

What: rule option for matching payload data with pcre

Type: ips_option

Usage: detect

Configuration:

- string **pcre.~re**: Snort regular expression

11.68 pkt_data

What: rule option to set the detection cursor to the normalized packet data

Type: ips_option

Usage: detect

11.69 pkt_num

What: alert on raw packet number

Type: ips_option

Usage: detect

Configuration:

- interval **pkt_num.~range**: check if packet number is in given range { 1: }

11.70 priority

What: rule option for prioritizing events

Type: ips_option

Usage: detect

Configuration:

- int **priority.~**: relative severity level; 1 is highest priority { 1: }

11.71 raw_data

What: rule option to set the detection cursor to the raw packet data

Type: ips_option

Usage: detect

11.72 reference

What: rule option to indicate relevant attack identification system

Type: ips_option

Usage: detect

Configuration:

- string **reference.~scheme**: reference scheme
- string **reference.~id**: reference id

11.73 regex

What: rule option for matching payload data with hyperscan regex

Type: ips_option

Usage: detect

Configuration:

- string **regex.~re**: hyperscan regular expression
- implied **regex.dotall**: matching a . will not exclude newlines
- implied **regex.fast_pattern**: use this content in the fast pattern matcher instead of the content selected by default
- implied **regex.multiline**: ^ and $ anchors match any newlines in data
- implied **regex.nocase**: case insensitive match
- implied **regex.relative**: start search from end of last match instead of start of buffer

11.74 rem

What: rule option to convey an arbitrary comment in the rule body

Type: ips_option

Usage: detect

Configuration:

- string **rem.~**: comment

11.75 replace

What: rule option to overwrite payload data; use with rewrite action

Type: ips_option

Usage: detect

Configuration:

- string **replace.~**: byte code to replace with

11.76 rev

What: rule option to indicate current revision of signature

Type: ips_option

Usage: detect

Configuration:

- int **rev.~**: revision { 1: }

11.77 rpc

What: rule option to check SUNRPC CALL parameters

Type: ips_option

Usage: detect

Configuration:

- int **rpc.~app**: application number
- string **rpc.~ver**: version number or * for any
- string **rpc.~proc**: procedure number or * for any

11.78 sd_pattern

What: rule option for detecting sensitive data

Type: ips_option

Usage: detect

Configuration:

- string **sd_pattern.~pattern**: The pattern to search for
- int **sd_pattern.threshold**: number of matches before alerting { 1 }

Peg counts:

- **sd_pattern.below_threshold**: sd_pattern matched but missed threshold (sum)
- **sd_pattern.pattern_not_found**: sd_pattern did not not match (sum)
- **sd_pattern.terminated**: hyperscan terminated (sum)

11.79 seq

What: rule option to check TCP sequence number

Type: ips_option

Usage: detect

Configuration:

- interval **seq.~range**: check if TCP sequence number is in given range { 0: }

11.80 service

What: rule option to specify list of services for grouping rules

Type: ips_option

Usage: detect

Configuration:

- string **service.***: one or more comma-separated service names

11.81 session

What: rule option to check user data from TCP sessions

Type: ips_option

Usage: detect

Configuration:

- enum **session.~mode**: output format { printable|binary|all }

11.82 sha256

What: payload rule option for hash matching

Type: ips_option

Usage: detect

Configuration:

- string **sha256.~hash**: data to match
- int **sha256.length**: number of octets in plain text { 1:65535 }
- string **sha256.offset**: var or number of bytes from start of buffer to start search
- implied **sha256.relative** = false: offset from cursor instead of start of buffer

11.83 sha512

What: payload rule option for hash matching

Type: ips_option

Usage: detect

Configuration:

- string **sha512.~hash**: data to match
- int **sha512.length**: number of octets in plain text { 1:65535 }
- string **sha512.offset**: var or number of bytes from start of buffer to start search
- implied **sha512.relative** = false: offset from cursor instead of start of buffer

11.84 sid

What: rule option to indicate signature number

Type: ips_option

Usage: detect

Configuration:

- int **sid.~**: signature id { 1: }

11.85 sip_body

What: rule option to set the detection cursor to the request body

Type: ips_option

Usage: detect

11.86 sip_header

What: rule option to set the detection cursor to the SIP header buffer

Type: ips_option

Usage: detect

11.87 sip_method

What: detection option for sip stat code

Type: ips_option

Usage: detect

Configuration:

- string **sip_method.*method**: sip method

11.88 sip_stat_code

What: detection option for sip stat code

Type: ips_option

Usage: detect

Configuration:

- int **sip_stat_code.*code**: stat code { 1:999 }

11.89 so

What: rule option to call custom eval function

Type: ips_option

Usage: detect

Configuration:

- string **so.~func**: name of eval function

11.90 soid

What: rule option to specify a shared object rule ID

Type: ips_option

Usage: detect

Configuration:

- string **soid.~**: SO rule ID is unique key, eg <gid>_<sid>_<rev> like 3_45678_9

11.91 ssl_state

What: detection option for ssl state

Type: ips_option

Usage: detect

Configuration:

- implied **ssl_state.client_hello**: check for client hello
- implied **ssl_state.server_hello**: check for server hello
- implied **ssl_state.client_keyx**: check for client keyx
- implied **ssl_state.server_keyx**: check for server keyx
- implied **ssl_state.unknown**: check for unknown record
- implied **ssl_state.!client_hello**: check for records that are not client hello
- implied **ssl_state.!server_hello**: check for records that are not server hello
- implied **ssl_state.!client_keyx**: check for records that are not client keyx
- implied **ssl_state.!server_keyx**: check for records that are not server keyx
- implied **ssl_state.!unknown**: check for records that are not unknown

11.92 ssl_version

What: detection option for ssl version

Type: ips_option

Usage: detect

Configuration:

- implied **ssl_version.sslv2**: check for sslv2
- implied **ssl_version.sslv3**: check for sslv3
- implied **ssl_version.tls1.0**: check for tls1.0
- implied **ssl_version.tls1.1**: check for tls1.1
- implied **ssl_version.tls1.2**: check for tls1.2
- implied **ssl_version.!sslv2**: check for records that are not sslv2
- implied **ssl_version.!sslv3**: check for records that are not sslv3
- implied **ssl_version.!tls1.0**: check for records that are not tls1.0
- implied **ssl_version.!tls1.1**: check for records that are not tls1.1
- implied **ssl_version.!tls1.2**: check for records that are not tls1.2

11.93 stream_reassemble

What: detection option for stream reassembly control

Type: ips_option

Usage: detect

Configuration:

- enum **stream_reassemble.action**: stop or start stream reassembly { disable|enable }
- enum **stream_reassemble.direction**: action applies to the given direction(s) { client|server|both }
- implied **stream_reassemble.noalert**: don't alert when rule matches
- implied **stream_reassemble.fastpath**: optionally whitelist the remainder of the session

11.94 stream_size

What: detection option for stream size checking

Type: ips_option

Usage: detect

Configuration:

- interval **stream_size.~range**: check if the stream size is in the given range { 0: }
- enum **stream_size.~direction**: compare applies to the given direction(s) { either|to_server|to_client|both }

11.95 tag

What: rule option to log additional packets

Type: ips_option

Usage: detect

Configuration:

- enum **tag.~**: log all packets in session or all packets to or from host { session|host_src|host_dst }
- int **tag.packets**: tag this many packets { 1: }
- int **tag.seconds**: tag for this many seconds { 1: }
- int **tag.bytes**: tag for this many bytes { 1: }

11.96 target

What: rule option to indicate target of attack

Type: ips_option

Usage: detect

Configuration:

- enum **target.~**: indicate the target of the attack { src_ip | dst_ip }

11.97 tos

What: rule option to check type of service field

Type: ips_option

Usage: detect

Configuration:

- interval **tos.~range**: check if IP TOS is in given range { 0:255 }

11.98 ttl

What: rule option to check time to live field

Type: ips_option

Usage: detect

Configuration:

- interval **ttl.~range**: check if IP TTL is in the given range { 0:255 }

11.99 urg

What: detection for TCP urgent pointer

Type: ips_option

Usage: detect

Configuration:

- interval **urg.~range**: check if tcp urgent offset is in given range { 0:65535 }

11.100 window

What: rule option to check TCP window field

Type: ips_option

Usage: detect

Configuration:

- interval **window.~range**: check if TCP window size is in given range { 0:65535 }

11.101 wscale

What: detection for TCP window scale

Type: ips_option

Usage: detect

Configuration:

- interval **wscale.~range**: check if TCP window scale is in given range { 0:65535 }

12 Search Engine Modules

Search engines perform multipattern searching of packets and payload to find rules that should be evaluated. There are currently no specific modules, although there are several search engine plugins. Related configuration is done with the basic detection module.

13 SO Rule Modules

SO rules are dynamic rules that require custom coding to perform detection not possible with the existing rule options. These rules typically do not have associated modules.

14 Logger Modules

All output of events and packets is done by Loggers.

14.1 alert_csv

What: output event in csv format

Type: logger

Usage: context

Configuration:

- bool **alert_csv.file** = false: output to alert_csv.txt instead of stdout
- multi **alert_csv.fields** = timestamp pkt_num proto pkt_gen pkt_len dir src_ap dst_ap rule action: selected fields will be output in given order left to right { action | class | b64_data | dir | dst_addr | dst_ap | dst_port | eth_dst | eth_len | eth_src | eth_type | gid | icmp_code | icmp_id | icmp_seq | icmp_type | iface | ip_id | ip_len | msg | mpls | pkt_gen | pkt_len | pkt_num | priority | proto | rev | rule | seconds | service | sid | src_addr | src_ap | src_port | target | tcp_ack | tcp_flags | tcp_len | tcp_seq | tcp_win | timestamp | tos | ttl | udp_len | vlan }
- int **alert_csv.limit** = 0: set maximum size in MB before rollover (0 is unlimited) { 0: }
- string **alert_csv.separator** = , : separate fields with this character sequence

14.2 alert_ex

What: output gid:sid:rev for alerts

Type: logger

Usage: context

Configuration:

- bool **alert_ex.upper** = false: true/false → convert to upper/lower case

14.3 alert_fast

What: output event with brief text format

Type: logger

Usage: context

Configuration:

- bool **alert_fast.file** = false: output to alert_fast.txt instead of stdout
- bool **alert_fast.packet** = false: output packet dump with alert
- int **alert_fast.limit** = 0: set maximum size in MB before rollover (0 is unlimited) { 0: }

14.4 alert_full

What: output event with full packet dump

Type: logger

Usage: context

Configuration:

- bool **alert_full.file** = false: output to alert_full.txt instead of stdout
- int **alert_full.limit** = 0: set maximum size in MB before rollover (0 is unlimited) { 0: }

14.5 alert_json

What: output event in json format

Type: logger

Usage: context

Configuration:

- bool **alert_json.file** = false: output to alert_json.txt instead of stdout
- multi **alert_json.fields** = timestamp pkt_num proto pkt_gen pkt_len dir src_ap dst_ap rule action: selected fields will be output in given order left to right { action | class | b64_data | dir | dst_addr | dst_ap | dst_port | eth_dst | eth_len | eth_src | eth_type | gid | icmp_code | icmp_id | icmp_seq | icmp_type | iface | ip_id | ip_len | msg | mpls | pkt_gen | pkt_len | pkt_num | priority | proto | rev | rule | seconds | service | sid | src_addr | src_ap | src_port | target | tcp_ack | tcp_flags | tcp_len | tcp_seq | tcp_win | timestamp | tos | ttl | udp_len | vlan }
- int **alert_json.limit** = 0: set maximum size in MB before rollover (0 is unlimited) { 0: }
- string **alert_json.separator** = , : separate fields with this character sequence

14.6 alert_sfsocket

What: output event over socket

Type: logger

Usage: context

Configuration:

- string **alert_sfsocket.file**: name of unix socket file
- int **alert_sfsocket.rules[].gid** = 1: rule generator ID { 1: }
- int **alert_sfsocket.rules[].sid** = 1: rule signature ID { 1: }

14.7 alert_syslog

What: output event to syslog

Type: logger

Usage: context

Configuration:

- enum **alert_syslog.facility** = auth: part of priority applied to each message { auth | authpriv | daemon | user | local0 | local1 | local2 | local3 | local4 | local5 | local6 | local7 }
- enum **alert_syslog.level** = info: part of priority applied to each message { emerg | alert | crit | err | warning | notice | info | debug }
- multi **alert_syslog.options**: used to open the syslog connection { cons | ndelay | perror | pid }

14.8 alert_unixsock

What: output event over unix socket

Type: logger

Usage: context

14.9 log_codecs

What: log protocols in packet by layer

Type: logger

Usage: context

Configuration:

- bool **log_codecs.file** = false: output to log_codecs.txt instead of stdout
- bool **log_codecs.msg** = false: include alert msg

14.10 log_hext

What: output payload suitable for daq hext

Type: logger

Usage: context

Configuration:

- bool **log_hext.file** = false: output to log_hext.txt instead of stdout
- bool **log_hext.raw** = false: output all full packets if true, else just TCP payload
- int **log_hext.limit** = 0: set maximum size in MB before rollover (0 is unlimited) { 0: }
- int **log_hext.width** = 20: set line width (0 is unlimited) { 0: }

14.11 log_pcap

What: log packet in pcap format

Type: logger

Usage: context

Configuration:

- int **log_pcap.limit** = 0: set maximum size in MB before rollover (0 is unlimited) { 0: }

14.12 unified2

What: output event and packet in unified2 format file

Type: logger

Usage: context

Configuration:

- bool **unified2.legacy_events** = false: generate Snort 2.X style events for barnyard2 compatibility
- int **unified2.limit** = 0: set maximum size in MB before rollover (0 is unlimited) { 0: }
- bool **unified2.nostamp** = true: append file creation time to name (in Unix Epoch format)

15 DAQ Configuration and Modules

The Data AcQuisition library (DAQ), provides pluggable packet I/O. LibDAQ replaces direct calls to libraries like libpcap 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 module and mode when invoking Snort to perform pcap readback or inline operation, etc. The DAQ library may be useful for other packet processing applications and the modular nature allows you to build new modules for other platforms.

The DAQ library is provided as a separate package on the official Snort download site (https://snort.org/downloads) and contains a number of DAQ modules including PCAP, AFPacket, NFQ, IPFQ, Netmap, and Dump implementations. Snort 3 itself contains a few new DAQ modules mostly used for testing as described below. Additionally, DAQ modules developed by third parties to facilitate the usage of their own hardware and software platforms exist.

15.1 Building the DAQ Library and Its Bundled DAQ Modules

Refer to the README in the LibDAQ source tarball for instructions on how to build the library and modules as well as details on configuring and using the bundled DAQ modules.

A copy of the README from LibDAQ has been included in the Reference section of this manual for convenience. For the most up-to-date information, please refer to the version that came with your installation's source code.

15.2 Configuration

As with a number of features in Snort 3, the LibDAQ and DAQ module configuration may be controlled using either the command line options or direct Snort module configuration.

DAQ modules may be statically built into Snort, but the more common case is to use DAQ modules that have been built as dynamically loadable objects. Because of this, the first thing to take care of is informing Snort of any locations it should search for dynamic DAQ modules. From the command line, this can be done with one or more invocations of the --daq-dir option,

which takes a path to search as its argument. All arguments will be collected into a list of locations to be searched. In the Lua configuration, the *module_dirs* property of the *daq* Snort module is a list of paths for the same purpose.

Next, one must select which DAQ module they wish to use by name. This is done using the --daq option from the command line or the *module* property of the *daq* Snort module. To get a list of the available modules, run Snort with the --daq-list option making sure to specify any DAQ module search directories beforehand. If no DAQ module is specified, Snort will default to attempting to find and use the *pcap* DAQ module.

Some DAQ modules can be further directly configured using DAQ module variables. All DAQ module variables come in the form of either just a key or a key and a value separated by an equals sign. For example, *debug* or *fanout_type=hash*. The command line option for specifying these is --daq-var and the configuration file equivalent is the *variables* property of the *daq* Snort module.

The LibDAQ concept of operational mode (passive, inline, or file readback) is not directly configurable but instead inferred from other Snort configuration. The DAQ module acquisition timeout is always configured to 1 second and the packet capture length (snaplen) is configured by the -s command line option and defaults to 1514 bytes.

Finally, and most importantly, is the input specification for the DAQ module. In readback mode, this is simply the file to be read back and analyzed. For live traffic processing, this is the name of the interface or other necessary input specification as required by the DAQ module to understand what to operate upon. From the command line, the -r option is used to specify a file to be read back and the -i option is used to indicate a live interface input specification. Both are covered by the *input_spec* property of the *daq* Snort module.

15.2.1 Command Line Example

```
    snort --daq-dir /usr/local/lib/daq --daq-dir /opt/lib/daq --daq afpacket
--daq-var debug --daq-var fanout_type=hash -i eth1:eth2
```

15.2.2 Configuration File Example

The following is the equivalent of the above command line DAQ configuration in Lua form:

```
daq =
{
    module_dirs =
    {
        '/usr/local/lib/daq',
        '/opt/lib/daq'
    },
    module = 'afpacket',
    input_spec = 'eth1:eth2',
    variables =
    {
        'debug',
        'fanout_type=hash'
    }
}
```

15.2.3 Interaction With Multiple Packet Threads

DAQ configuration can become much more complicated as additional packet threads are introduced. To allow for more flexibility in configuring DAQ module instances, each packet thread can be configured with its own input specification and/or DAQ module variables, which creates two classes of each: instance-specific and global. Global DAQ module variables are those defined before any -i option on the command line or in the top-level *variables* property demonstrated in the previous section. The global input specification is defined by either the first -i option on the command line (which doubles as the input specification for instance 0) or the top-level *input_spec* in the i'daq' Snort module. Instance-specific input specifiers are configured on the command line by

giving multiple -i options. In the same way, instance-specific DAQ module variables on the command line are declared normally but follow and apply only to the instance operating on the last -i option. When configuring through Lua, the *instances* property of the *daq* Snort module is a list of tables, each defining instance-specific configuration for a given instance ID.

Each packet thread will create an instance of the chosen DAQ module using the global interface specification and global set of DAQ module variables **unless** they were overridden with instance-specific values. When DAQ module instances are configured, any global DAQ modules will be set and then any instance-specific DAQ variables. This means that an instance will "inherit" the global DAQ modules and can override those by specifying them again with different values or add to them by specifying new variables entirely.

Here is the configuration for a hypothetical AFPacket DAQ module that has been modified to loadbalance based on DAQ variables (lb_total is the total number of instances to loadbalance across and is set globally, and lb_id is the instance's loadbalancing ID within that total and is set per-instance) across 4 packet processing threads within Snort:

```
daq =
{
    module_dirs =
    {
        '/usr/local/sf/lib/daq'
    },
    module = 'afpacket',
    input_spec = 'eth1',
    variables =
    {
        'lb_total=4'
    },
    instances =
    {
        {
            id = 0,
            variables =
            {
                'lb_id=1',
            }
        },
        {
            id = 1,
            variables =
            {
                'lb_id=2',
            }
        },
        {
            id = 2,
            variables =
            {
                'lb_id=3',
            }
        },
        {
            id = 3,
            variables =
            {
                'lb_id=4',
            }
        },
    }
}
```

The equivalent command line invocation would look like this (made uglier by the lack of needing a different input specification for each thread):

```
    snort --daq-dir /usr/local/sf/lib/daq --daq afpacket --daq-var lb_total=4 -i
eth1 --daq-var lb_id=1 -i eth1 --daq-var lb_id=2 -i eth1 --daq-var lb_id=3 -i
eth1 --daq-var lb_id=4 -z 4
```

For any particularly complicated setup, it is recommended that one configure via a Lua configuration file rather than using the command line options.

15.3 DAQ Modules Included With Snort 3

15.3.1 Socket Module

The socket module provides provides a stream socket server that will accept up to 2 simultaneous connections and bridge them together while also passing data to Snort for inspection. The first connection accepted is considered the client and the second connection accepted is considered the server. If there is only one connection, stream data can't be forwarded but it is still inspected.

Each read from a socket of up to snaplen bytes is passed as a packet to Snort along with a DAQ_SktHdr_t pointer in DAQ_PktHdr_t→priv DAQ_SktHdr_t conveys IP4 address, ports, protocol, and direction. Socket packets can be configured to be TCP or UDP. The socket DAQ can be operated in inline mode and is able to block packets.

The socket DAQ uses DLT_SOCKET and requires that Snort load the socket codec which is included in the extra package.

To use the socket DAQ, start Snort like this:

```
./snort --plugin-path /path/to/lib/snort_extra \
    --daq socket [--daq-var port=<port>] [--daq-var proto=<proto>] [-Q]

<port> ::= 1..65535; default is 8000
<proto> ::= tcp | udp
```

- This module only supports ip4 traffic.
- This module is only supported by Snort 3. It is not compatible with Snort 2.
- This module is primarily for development and test.

15.3.2 File Module

The file module provides the ability to process files directly w/o having to extract them from pcaps. Use the file module with Snort's stream_file to get file type identification and signature services. The usual IPS detection and logging etc. is available too.

You can process all the files in a directory recursively using 8 threads with these Snort options:

```
--pcap-dir path -z 8
```

- This module is only supported by Snort 3. It is not compatible with Snort 2.
- This module is primarily for development and test.

15.3.3 Hext Module

The hext module generates packets suitable for processing by Snort from hex/plain text. Raw packets include full headers and are processed normally. Otherwise the packets contain only payload and are accompanied with flow information (4-tuple) suitable for processing by stream_user.

The first character of the line determines it's purpose:

```
'$' command
'#' comment
'"' quoted string packet data
'x' hex packet data
' ' empty line separates packets
```

The available commands are:

```
$client <ip4> <port>
$server <ip4> <port>

$packet -> client
$packet -> server

$packet <addr> <port> -> <addr> <port>

$sof <i32:ingressZone> <i32:egressZone> <i32:ingressIntf> <i32:egressIntf> <s: ↩
    srcIp> <i16:srcPort> <s:destIp> <i16:dstPort> <u32:opaque> <u64:initiatorPkts>  ↩
    <u64:responderPkts> <u64:initiatorPktsDropped> <u64:responderPktsDropped> <u64: ↩
    initiatorBytesDropped> <u64:responderBytesDropped> <u8:isQosAppliedOnSrcIntf> < ↩
    timeval:sof_timestamp> <timeval:eof_timestamp> <u16:vlan> <u16:address_space_id ↩
    > <u8:protocol>
$eof <i32:ingressZone> <i32:egressZone> <i32:ingressIntf> <i32:egressIntf> <s: ↩
    srcIp> <i16:srcPort> <s:destIp> <i16:dstPort> <u32:opaque> <u64:initiatorPkts>  ↩
    <u64:responderPkts> <u64:initiatorPktsDropped> <u64:responderPktsDropped> <u64: ↩
    initiatorBytesDropped> <u64:responderBytesDropped> <u8:isQosAppliedOnSrcIntf> < ↩
    timeval:sof_timestamp> <timeval:eof_timestamp> <u16:vlan> <u16:address_space_id ↩
    > <u8:protocol>
```

Client and server are determined as follows. $packet → client indicates to the client (from server) and $packet → server indicates a packet to the server (from client). $packet followed by a 4-tuple uses the heuristic that the client is the side with the greater port number.

The default client and server are 192.168.1.1 12345 and 10.1.2.3 80 respectively. $packet commands with a 4-tuple do not change client and server set with the other $packet commands.

$packet commands should be followed by packet data, which may contain any combination of hex and strings. Data for a packet ends with the next command or a blank line. Data after a blank line will start another packet with the same tuple as the prior one.

$sof and $eof commands generate Start of Flow and End of Flow metapackets respectively. They are followed by a definition of a Flow_Stats_t data structure which will be fed into Snort via the metadata callback.

Strings may contain the following escape sequences:

```
\r = 0x0D = carriage return
\n = 0x0A = new line
\t = 0x09 = tab
\\ = 0x5C = \
```

Format your input carefully; there is minimal error checking and little tolerance for arbitrary whitespace. You can use Snort's -L hext option to generate hext input from a pcap.

- This module only supports ip4 traffic.
- This module is only supported by Snort 3. It is not compatible with Snort 2.
- This module is primarily for development and test.

The hext DAQ also supports a raw mode which is activated by setting the data link type. For example, you can input full ethernet packets with --daq-var dlt=1 (Data link types are defined in the DAQ include sfbpf_dlt.h.) Combine that with the hext logger in raw mode for a quick (and dirty) way to edit pcaps. With --lua "log_hext = { raw = true }", the hext logger will dump the full packet in a way that can be read by the hext DAQ in raw mode. Here is an example:

```
# 3 [96]

x02 09 08 07 06 05 02 01 02 03 04 05 08 00 45 00 00 52 00 03  # ..............E..R ↩
    ..
x00 00 40 06 5C 90 0A 01 02 03 0A 09 08 07 BD EC 00 50 00 00  # ..@.\............P ↩
    ..
x00 02 00 00 00 02 50 10 20 00 8A E1 00 00 47 45 54 20 2F 74  # ......P.  .....GET ↩
     /t
x72 69 67 67 65 72 2F 31 20 48 54 54 50 2F 31 2E 31 0D 0A 48  # rigger/1 HTTP ↩
    /1.1..H
x6F 73 74 3A 20 6C 6F 63 61 6C 68 6F 73 74 0D 0A              # ost: localhost..
```

A comment indicating packet number and size precedes each packet dump. Note that the commands are not applicable in raw mode and have no effect.

16 Snort 3 vs Snort 2

Snort 3 differs from Snort 2 in the following ways:

- command line and conf file syntax made more uniform
- removed unused and deprecated features
- remove as many barriers to successful run as possible (e.g.: no upper bounds on memcaps)
- assume the simplest mode of operation (e.g.: never assume input from or output to some hardcoded filename)
- all Snort 2 config options are grouped into Snort 3 modules

16.1 Features New to Snort 3

Some things Snort++ can do today that Snort can not do:

- regex fast patterns, not just literals
- FlatBuffers and JSON perf monitor logs
- LuaJIT scriptable rule options and loggers
- pub/sub inspection events (currently used by sip and http_inspect to appid)
- JIT buffer stuffers (notably with new http_inspect)
- C-style comments in rules
- #begin ... #end comment blocks in rules
- rule remarks (comment is part of rule, not just in it)

- process raw files (eg read a PDF and do file processing)
- process raw payload (eg bridge 2 sockets and do inspection)
- fast pattern offload to separate thread (experimental)
- track all memory allocated
- add or override any config item on command line
- set CPU affinity
- pause and resume commands

16.2 Features Improved over Snort 2

Some things Snort++ can do today that Snort can not do as well:

- Hyperscan search engine plugin (Intel provides patch for Snort 2)
- fast pattern sensitive data (Snort 2 requires a slow, extra search)
- multiple packet threads with one config (Snort 2 requires multiple processes)
- wizard automatically detects service for first flow (Snort 2 appid detects for next flow)
- nested policy binding (Snort 2 has just one level)
- decode arbitrary layers (Snort 2 supports only 2 IP layers)
- process PDU buffers (Snort 2 only processes packets)
- fully stateful http_inspect with 97 builtin alerts (Snort 2 is only partly stateful with 33 builtin alerts)
- output all semantic errors before quitting (Snort 2 stops at first one)
- alert file rules (Snort 2 must use multiple rules)
- alert service rules, eg alert http (Snort 2 must use metadata:service)
- automatic fast_pattern only (Snort 2 requires explicit fast_pattern:only)
- elided rule headers omit nets and/or ports (Snort 2 requires explicit *any*)
- dump builtin rule stubs (Snort 2 can only dump SO stubs)
- rule sticky buffers (Snort 2 buffers must be repeated)
- http_header:name supported to restrict to single field (Snort 2 searches all headers)
- fully equivalent SO rules (Snort 2 has some limitations with SO processing)
- text-based SO rule implementation (Snort 2 requires tedious, nested C structs)
- extensible module-based tracing (Snort 2 has a fixed set of flags)
- over 200 plugins, no need to change core source code (Snort 2 only supports preprocessors and outputs)
- use consistent conf syntax (Snort 2 defines lists different ways in different places, etc.)
- use consistent rule syntax (Snort 2 has semicolon separated suboptions, etc.)
- arbitrary whitespace and comments in conf and rules (Snort 2 requires newline escapes)
- properly parse rules (Snort 2 can actually completely ignore stuff)
- optional, expanded warnings output, can be fatal (Snort 2 warnings limited and are not optional or fatal)

- define and use arbitrary variables and functions in config with Lua (Snort 2 has variables just for rule headers)
- text-based command line shell (Snort 2 has binary control socket)
- generate text and HTML user guide in addition to PDF (Snort 2 just has PDF and Talos provides HTML)
- generate developer's guide (Snort 2's is manually written)
- extensive command line help, eg every config item, rule option, and peg count (Snort 2 only has command line args)
- cmake builds (Snort 2 only does automake)
- read rules from separate file or stdin (Snort 2 requires rules directly in or included in conf)
- simple, clean, uniform startup and shutdown output (Snort 2 is heavy and inconsistent)
- port_scan is fully configurable (Snort 2 hard codes most of the configuration)
- port_scan can block scans (Snort 2 can only detect scans)
- sigquit will cause a --dirty-pig style exit (Snort 2 handles sigquit the same as sigterm and sigint)
- detection trace (Snort 2 has more limited buffer dumping)
- updated unified2 events with MPLS, VLAN, and IP6 (Snort 2 requires configuration and extra data)
- significantly more unit tests, including --catch and make check (Snort 2 has very few unit tests)
- better modularity 346K/1534 = 226 lines/file, max=2700 (Snort 2 has 440K/1021 = 431 lines/file, max=13K)

16.3 Build Options

- configure --with-lib{pcap,pcre}-* → --with-{pcap,pcre}-*
- control socket, cs_dir, and users were deleted
- POLICY_BY_ID_ONLY code was deleted
- hardened --enable-inline-init-failopen / INLINE_FAILOPEN

16.4 Command Line

- --pause loads config and waits for resume before processing packets
- --require-rule-sid is hardened
- --shell enables interactive Lua shell
- -T is assumed if no input given
- added --help-config prefix to dump all matching settings
- added --script-path
- added -L none|dump|pcap
- added -z <#> and --max-packet-threads <#>
- delete --enable-mpls-multicast, --enable-mpls-overlapping-ip, --max-mpls-labelchain-len, --mpls-payload-type
- deleted --pid-path and --no-interface-pidfile
- deleting command line options which will be available with --lua or some such including: -I, -h, -F, -p, --disable-inline-init-failopen

- hardened -n < 0
- removed --search-method
- replaced "unknown args are bpf" with --bpf
- replaced --dynamic-*-lib[-dir] with --plugin-path (with : separators)
- removed -b, -N, -Z and, --perfmon-file options

16.5 Conf File

- Snort 3 has a default unicode.map
- Snort 3 will not enforce an upper bound on memcaps and the like within 64 bits
- Snort 3 will supply a default *_global config if not specified (Snort 2 would fatal; e.g. http_inspect_server w/o http_inspect_global)
- address list syntax changes: [[and]] must be [[and]] to avoid Lua string parsing errors (unless in quoted string)
- because the Lua conf is live code, we lose file:line locations in app error messages (syntax errors from Lua have file:line)
- changed search-method names for consistency
- delete config include_vlan_in_alerts (not used in code)
- delete config so_rule_memcap (not used in code)
- deleted --disable-attribute-table-reload-thread
- deleted config decode_*_{alerts,drops} (use rules only)
- deleted config dump-dynamic-rules-path
- deleted config ipv6_frag (not actually used)
- deleted config threshold and ips rule threshold (→ event_filter)
- eliminated ac-split; must use ac-full-q split-any-any
- frag3 → defrag, arpspoof → arp_spoof, sfportscan → port_scan, perfmonitor → perf_monitor, bo → back_orifice
- limits like "1234K" are now "limit = 1234, units = *K*"
- lua field names are (lower) case sensitive; snort.conf largely wasn't
- module filenames are not configurable: always <log-dir>/<module-name><suffix> (suffix is determined by module)
- no positional parameters; all name = value
- perf_monitor configuration was simplified
- portscan.detect_ack_scans deleted (exact same as include_midstream)
- removed various run modes - now just one
- frag3 default policy is Linux not bsd
- lowmem* search methods are now in snort_examples
- deleted unused http_inspect stateful mode
- deleted stateless inspection from ftp and telnet
- deleted http and ftp alert options (now strictly rule based)
- preprocessor disabled settings deleted since no longer relevant
- sessions are always created; snort config stateful checks eliminated
- stream5_tcp: prune_log_max deleted; to be replaced with histogram
- stream5_tcp: max_active_responses, min_response_seconds moved to active.max_responses, min_interval

16.6 Rules

- all rules must have a sid
- sid == 0 not allowed
- deleted activate / dynamic rules
- deleted unused rule_state.action
- deleted metadata engine shared
- deleted metadata: rule-flushing (with PDU flushing rule flushing can cause missed attacks, the opposite of its intent)
- changed metadata:service one[, service two]; to service:one[, two];
- soid is now a non-metadata option
- metadata is now truly metadata with no impact on detection (Snort doesn't care about metadata internal structure / syntax)
- deleted fast_pattern:only; use fast_pattern, nocase (option is not added to detection tree if not required)
- changed fast_pattern:<offset>,<length> to fast_pattern,fast_pattern_offset <offset>,fast_pattern_length <length>
- fast pattern sensitive data with sd_pattern using hyperscan
- hyperscan regex fast patterns with regex:"<regex>", fast_pattern;
- no ; separated content suboptions
- offset, depth, distance, and within must use a space separator not colon (e.g. offset:5; becomes offset 5;)
- content suboptions http_* are now full options
- added sticky buffers: buffer selector options must precede contents and remain in effect until changed
- the following pcre options have been deleted: use sticky buffers instead B, U, P, H, M, C, I, D, K, S, Y
- deleted uricontent option; use sticky buffer uricontent:"foo" -→ http_uri; content:"foo"
- deleted urilen raw and norm; must use http_raw_uri and http_uri instead
- deleted unused http_encode option
- urilen replaced with generic bufferlen which applies to current sticky buffer
- added optional selector to http_header, e.g. http_header:User-Agent;
- the all new http_inspect has new buffers and rule options
- added alert file and alert service rules (service in body not required if there is only one and it is in header; alert service / file rules disable fast pattern searching of raw packets)
- rule option sequence: <stub> soid <hidden>
- arbitrary whitespace and multiline rules w/o \n
- #begin … #end comments to easily comment out multiple lines
- add rule remarks option with rem:"arbitrary comment"
- nets and/or ports may be omitted from rule headers (matches any)
- parse all rules and output all errors before quitting
- read rules from conf, separate rules file, or stdin
- The symbol =< in a byte test is recognized as a syntax error. The correct symbol is <=.

16.7 Output

- alert_fast includes packet data by default
- all text mode outputs default to stdout
- changed default logging mode to -L none
- deleted layer2resets and flexresp2_*
- deleted log_ascii
- general output guideline: don't print zero counts
- Snort 3 queues decoder and inspector events to the main event queue before ips policy is selected; since some events may not be enabled, the queue needs to be sized larger than with Snort 2 which used an intermediate queue for decoder events.
- deleted the intermediate http and ftp_telnet event queues
- alert_unified2 and log_unified2 have been deleted

16.8 Sensitive Data

The Snort 2.X SDF Preprocessor is gone, replaced by ips option `sd_pattern`. The sd_pattern rule option is synonymous with the sd_pattern option used for gid:138 rules, but has a different syntax. A major difference in syntax is the use of Hyperscan pattern matching library which provides a regex language similar to PCRE.

To facilitate continued performance, sd_pattern rule option is implemented with Hyperscan pattern matching library. The rule option is now also utilized as a "fast pattern" in the Snort engine which provides a significant performance improvement over the separate detection step of earlier implementations.

The preprocessor alert SDF_COMBO_ALERT (139:1) has been removed and has no replacement in Snort 3.X. This is because the rule offered no additional value over gid:138 rules and was difficult to interpret the result of.

For more information, See Features > Sensitive Data Filtering for details.

16.9 Features Not Yet Supported by Snort 3

- Support in http_inspect for Original Client IP is limited to the X-Forwarded-For and True-Client-IP headers in that order. It is not possible to configure additional custom headers to search for Original Client IP.
- The -n option does not work properly when perf_monitor is configured. The number of packets processed from the pcap is likely to be more than the number specified with the -n option.
- When a file is transferred via SMB2 it may be allowed even though according to file policy it should be blocked. This occurs when the create and read requests are sent together and then the read and create responses are sent together. Blocking is done correctly if the create and read requests are sent separately or if the file is large enough to require two read responses.
- This user manual is incomplete and does not fully cover many Snort 2.X features that are also supported by Snort 3.

17 Snort2Lua

One of the major differences between Snort 2 and Snort 3 is the configuration. Snort 2 configuration files are written in Snort-specific syntax while Snort 3 configuration files are written in Lua. Snort2Lua is a program specifically designed to convert valid Snort 2 configuration files into Lua files that Snort 3 can understand.

Snort2Lua reads your legacy Snort conf file(s) and generates Snort 3 Lua and rules files. When running this program, the only mandatory option is to provide Snort2Lua with a Snort 2 configuration file. The default output file file is snort.lua, the default error file will be snort.rej, and the default rule file is the output file (default is snort.lua). When Snort2Lua finishes running,

the resulting configuration file can be successfully run as the Snort3.0 configuration file. The sole exception to this rule is when Snort2Lua cannot find an included file. If that occurs, the file will still be included in the output file and you will need to manually adjust or comment the file name. Additionally, if the exit code is not zero, some of the information may not be successfully converted. Check the error file for all of the conversion problems.

Those errors can occur for a multitude of reasons and are not necessarily bad. Snort2Lua expects a valid Snort 2 configuration. Therefore, if the configuration is invalid or has questionable syntax, Snort2Lua may fail to parse the configuration file or create an invalid Snort 3 configuration file.

There are a also few peculiarities of Snort2Lua that may be confusing to a first time user:

- Aside from an initial configuration file (which is specified from the command line or as the file in 'config binding'), every file that is included into Snort 3 must be either a Lua file or a rule file; the file cannot contain both rules and Lua syntax. Therefore, when parsing a file specified with the 'include' command, Snort2Lua will output both a Lua file and a rule file.
- Any line that is a comment in a configuration file will be added in to a comments section at the bottom of the main configuration file.
- Rules that contain unsupported options will be converted to the best of Snort2Lua's capability and then printed as a comment in the rule file.
- Files with a *.rules* suffix are assumed to be Talos 2.X rules files and converted line-by-line. In this case, lines starting with *alert* are converted as usual but lines starting with *# alert* are assumed to be commented out rules which are converted to 3.0 format and remain comments in the output file. All other comments are passed through directly. There is no support for other commented rule actions since these do not appear in Talos rules files.

17.1 Snort2Lua Command Line

By default, Snort2Lua will attempt to parse every 'include' file and every 'binding' file. There is an option to change this functionality.

When specifying a rule file with one of the command line options, Snort2Lua will output all of the converted rules to that specified rule file. This is especially useful when you are only interesting in converting rules since there is no Lua syntax in rule files. There is also an option that tells Snort2Lua to output every rule for a given configuration into a single rule file. Similarly, there is an option pull all of the Lua syntax from every 'include' file into the output file.

There are currently three output modes: default, quiet, and differences. As expected, quiet mode produces a Snort configuration. All errors (aside from Fatal Snort2Lua errors), differences, and comments will omitted from the final output file. Default mode will print everything. That mean you will be able to see exactly what changes have occurred between Snort 2 and Snort 3 in addition to the new syntax, the original file's comments, and all errors that have occurred. Finally, differences mode will not actually output a valid Snort 3 configuration. Instead, you can see the exact options from the input configuration that have changed.

17.1.1 Usage: snort2lua [OPTIONS]... -c <snort_conf> ...

Converts the Snort configuration file specified by the -c or --conf-file options into a Snort++ configuration file

Options:

- **-?** show usage
- **-h** this overview of snort2lua
- **-a** default option. print all data
- **-c <snort_conf>** The Snort <snort_conf> file to convert
- **-d** print the differences, and only the differences, between the Snort and Snort++ configurations to the <out_file>
- **-e <error_file>** output all errors to <error_file>

- **-i** if <snort_conf> file contains any <include_file> or <policy_file> (i.e. *include path/to/conf/other_conf*), do NOT parse those files
- **-m** add a remark to the end of every converted rule
- **-o <out_file>** output the new Snort++ lua configuration to <out_file>
- **-q** quiet mode. Only output valid configuration information to the <out_file>
- **-r <rule_file>** output any converted rule to <rule_file>
- **-s** when parsing <include_file>, write <include_file>'s rules to <rule_file>. Meaningless if *-i* provided
- **-t** when parsing <include_file>, write <include_file>'s information, excluding rules, to <out_file>. Meaningless if *-i* provided
- **-V** Print the current Snort2Lua version
- **--bind-wizard** Add default wizard to bindings
- **--conf-file** Same as *-c*. A Snort <snort_conf> file which will be converted
- **--dont-parse-includes** Same as *-p*. if <snort_conf> file contains any <include_file> or <policy_file> (i.e. *include path/to/conf/other_conf*), do NOT parse those files
- **--error-file=<error_file>** Same as *-e*. output all errors to <error_file>
- **--help** Same as *-h*. this overview of snort2lua
- **--ips-policy-pattern** Convert config bindings matching this path to ips policy bindings
- **--markup** print help in asciidoc compatible format
- **--output-file=<out_file>** Same as *-o*. output the new Snort++ lua configuration to <out_file>
- **--print-all** Same as *-a*. default option. print all data
- **--print-binding-order** Print sorting priority used when generating binder table
- **--print-differences** Same as *-d*. output the differences, and only the differences, between the Snort and Snort++ configurations to the <out_file>
- **--quiet** Same as *-q*. quiet mode. Only output valid configuration information to the <out_file>
- **--remark** same as *-m*. add a remark to the end of every converted rule
- **--rule-file=<rule_file>** Same as *-r*. output any converted rule to <rule_file>
- **--single-conf-file** Same as *-t*. when parsing <include_file>, write <include_file>'s information, excluding rules, to <out_file>
- **--single-rule-file** Same as *-s*. when parsing <include_file>, write <include_file>'s rules to <rule_file>.
- **--version** Same as *-V*. Print the current Snort2Lua version

Required option:

- A Snort configuration file to convert. Set with either *-c* or *--conf-file*

Default values:

- <out_file> = snort.lua
- <rule_file> = <out_file> = snort.lua. Rules are written to the *local_rules* variable in the <out_file>
- <error_file> = snort.rej. This file will not be created in quiet mode.

17.2 Known Problems

- Any Snort 2 'string' which is dependent on a variable will no longer have that variable in the Lua string.
- Snort2Lua currently does not handle variables well. First, that means variables will not always be parsed correctly. Second, sometimes a variables value will be output in the lua file rather than a variable For instance, if Snort2Lua attempted to convert the line *include $RULE_PATH/example.rule*, the output may output *include /etc/rules/example.rule* instead.
- When Snort2Lua parses a 'binding' configuration file, the rules and configuration will automatically be combined into the same file. Also, the new files name will automatically become the old file's name with a .lua extension. There is currently no way to specify or change that files name.
- If a rule's action is a custom ruletype, that rule action will be silently converted to the rultype's *type*. No warnings or errors are currently emitted. Additionally, the custom ruletypes outputs will be silently discarded.
- If the original configuration contains a binding that points to another file and the binding file contains an error, Snort2Lua will output the number of rejects for the binding file in addition to the number of rejects in the main file. The two numbers will eventually be combined into one output.

17.3 Usage

Snort2Lua is included in the Snort 3 distribution. The Snort2Lua source code is located in the tools/snort2lua directory. The program is automatically built and installed.

Translating your configuration

To run Snort2Lua, the only requirement is a file containing Snort 2 syntax. Assuming your configuration file is named snort.conf, run the command

```
snort2lua -c snort.conf
```

Snort2Lua will output a file named snort.lua. Assuming your snort.conf file is a valid Snort 2 configuration file, than the resulting snort.lua file will always be a valid Snort 3 configuration file; any errors that occur are because Snort 3 currently does not support all of the Snort 2 options.

Every keyword from the Snort configuration can be found in the output file. If the option or keyword has changed, then a comment containing both the option or keyword's old name and new name will be present in the output file.

Translating a rule file

Snort2Lua can also accommodate translating individual rule files. Assuming the Snort 2 rule file is named snort.rules and you want the new rule file to be name updated.rules, run the command

```
snort2lua -c snort.rules -r updated.rules
```

Snort2Lua will output a file named updated.rules. That file, updated.rules, will always be a valid Snort 3 rule file. Any rule that contains unsupported options will be a comment in the output file.

Understanding the Output

Although Snort2Lua outputs very little to the console, there are several things that occur when Snort2Lua runs. This is a list of Snort2Lua outputs.

The console. Every line that Snort2Lua is unable to translate from the Snort 2.X format to the Snort 3 format is considered an error. Upon exiting, Snort2Lua will print the number of errors that occurred. Snort2Lua will also print the name of the error file.

The output file. As previously mentioned, Snort2Lua will create a Lua file with valid Snort 3 syntax. The default Lua file is named snort.lua. This file is the equivalent of your main Snort 2 configuration file.

The rule file. By default, all rules will be printed to the Lua file. However, if a rule file is specified on the command line, any rules found in the Snort 2 configuration will be written to the rule file instead

The error file. By default, the error file is snort.rej. It will only be created if errors exist. Every error referenced on the command line can be found in this file. There are two reasons an error can occur.

- The Snort 2 configuration file has invalid syntax. If Snort 2 cannot parse the configuration file, neither can Snort2Lua. In the example below, Snort2Lua could not convert the line *config bad_option*. Since that is not valid Snort 2 syntax, this is a syntax error.
- The Snort 2 configuration file contains preprocessors and rule options that are not supported in Snort 3. If Snort 2 can parse a line that Snort2Lua cannot parse, than Snort 3 does not support something in the line. As Snort 3 begins supporting these preprocessors and rule options, Snort2Lua will also begin translating these lines. One example of such an error is dcerpc2.

Additional .lua and .rules files. Every time Snort2Lua parses the include or binding keyword, the program will attempt to parse the file referenced by the keyword. Snort2Lua will then create one or two new files. The new files will have a .lua or .rules extension appended to the original filename.

18 Extending Snort

18.1 Plugins

Plugins have an associated API defined for each type, all of which share a common *header*, called the BaseApi. A dynamic library makes its plugins available by exporting the snort_plugins symbol, which is a null terminated array of BaseApi pointers.

The BaseApi includes type, name, API version, plugin version, and function pointers for constructing and destructing a Module. The specific API add various other data and functions for their given roles.

18.2 Modules

If we are defining a new Inspector called, say, gadget, it might be configured in snort.lua like this:

```
gadget =
{
    brain = true,
    claw = 3
}
```

When the gadget table is processed, Snort will look for a module called gadget. If that Module has an associated API, it will be used to configure a new instance of the plugin. In this case, a GadgetModule would be instantiated, brain and claw would be set, and the Module instance would be passed to the GadgetInspector constructor.

Module has three key virtual methods:

- **begin()** - called when Snort starts processing the associated Lua table. This is a good place to allocate any required data and set defaults.
- **set()** - called to set each parameter after validation.
- **end()** - called when Snort finishes processing the associated Lua table. This is where additional integrity checks of related parameters should be done.

The configured Module is passed to the plugin constructor which pulls the configuration data from the Module. For non-trivial configurations, the working paradigm is that Module hands a pointer to the configured data to the plugin instance which takes ownership.

Note that there is at most one instance of a given Module, even if multiple plugin instances are created which use that Module. (Multiple instances require Snort binding configuration.)

18.3 Inspectors

There are several types of inspector, which determines which inspectors are executed when:

- IT_BINDER - determines which inspectors apply to given flows
- IT_WIZARD - determines which service inspector to use if none explicitly bound
- IT_PACKET - used to process all packets before session and service processing (e.g. normalize)
- IT_NETWORK - processes packets w/o service (e.g. arp_spoof, back_orifice)
- IT_STREAM - for flow tracking, ip defrag, and tcp reassembly
- IT_SERVICE - for http, ftp, telnet, etc.
- IT_PROBE - process all packets after all the above (e.g. perf_monitor, port_scan)

18.4 Codecs

The Snort Codecs decipher raw packets. These Codecs are now completely pluggable; almost every Snort Codec can be built dynamically and replaced with an alternative, customized Codec. The pluggable nature has also made it easier to build new Codecs for protocols without having to touch the Snort code base.

The first step in creating a Codec is defining its class and protocol. Every Codec must inherit from the Snort Codec class defined in "framework/codec.h". The following is an example Codec named "example" and has an associated struct that is 14 bytes long.

```
#include <cstdint>
#include <arpa/inet.h>
#include "framework/codec.h"
#include "main/snort_types.h"

#define EX_NAME "example"
#define EX_HELP "example codec help string"

struct Example
{
    uint8_t dst[6];
    uint8_t src[6];
    uint16_t ethertype;

    static inline uint8_t size()
    { return 14; }
}

class ExCodec : public Codec
{
public:
    ExCodec() : Codec(EX_NAME) { }
    ~ExCodec() { }

    bool decode(const RawData&, CodecData&, DecodeData&) override;
    void get_protocol_ids(std::vector<uint16_t>&) override;
};
```

After defining ExCodec, the next step is adding the Codec's decode functionality. The function below does this by implementing a valid decode function. The first parameter, which is the RawData struct, provides both a pointer to the raw data that has come from a wire and the length of that raw data. The function takes this information and validates that there are enough bytes for this protocol. If the raw data's length is less than 14 bytes, the function returns false and Snort discards the packet; the packet is neither inspected nor processed. If the length is greater than 14 bytes, the function populates two fields in the CodecData struct, next_prot_id and lyr_len. The lyr_len field tells Snort the number of bytes that this layer contains. The next_prot_id field provides Snort the value of the next EtherType or IP protocol number.

```
bool ExCodec::decode(const RawData& raw, CodecData& codec, DecodeData&)
{
   if ( raw.len < Example::size() )
       return false;

    const Example* const ex = reinterpret_cast<const Example*>(raw.data);
    codec.next_prot_id = ntohs(ex->ethertype);
    codec.lyr_len = ex->size();
    return true;
}
```

For instance, assume this decode function receives the following raw data with a validated length of 32 bytes:

```
00 11 22 33 44 55 66 77    88 99 aa bb 08 00 45 00
00 38 00 01 00 00 40 06    5c ac 0a 01 02 03 0a 09
```

The Example struct's EtherType field is the 13 and 14 bytes. Therefore, this function tells Snort that the next protocol has an EtherType of 0x0800. Additionally, since the lyr_len is set to 14, Snort knows that the next protocol begins 14 bytes after the beginning of this protocol. The Codec with EtherType 0x0800, which happens to be the IPv4 Codec, will receive the following data with a validated length of 18 (== 32 – 14):

```
45 00 00 38 00 01 00 00    40 06 5c ac 0a 01 02 03
0a 09
```

How does Snort know that the IPv4 Codec has an EtherType of 0x0800? The Codec class has a second virtual function named get_protocol_ids(). When implementing the function, a Codec can register for any number of values between 0x0000 - 0xFFFF. Then, if the next_proto_id is set to a value for which this Codec has registered, this Codec's decode function will be called. As a general note, the protocol ids between [0, 0x00FF] are IP protocol numbers, [0x0100, 0x05FF] are custom types, and [0x0600, 0xFFFF] are EtherTypes.

For example, in the get_protocol_ids function below, the ExCodec registers for the protocols numbers 17, 787, and 2054. 17 happens to be the protocol number for UDP while 2054 is ARP's EtherType. Therefore, this Codec will now attempt to decode UDP and ARP data. Additionally, if any Codec sets the next_protocol_id to 787, ExCodec's decode function will be called. Some custom protocols are already defined in the file "protocols/protocol_ids.h"

```
void ExCodec::get_protocol_ids(std::vector<uint16_t>&v)
{
    v.push_back(0x0011); // == 17  == UDP
    v.push_back(0x1313); // == 787  == custom
    v.push_back(0x0806); // == 2054  == ARP
}
```

To register a Codec for Data Link Type's rather than protocols, the function get_data_link_type() can be similarly implemented.

The final step to creating a pluggable Codec is the snort_plugins array. This array is important because when Snort loads a dynamic library, the program only find plugins that are inside the snort_plugins array. In other words, if a plugin has not been added to the snort_plugins array, that plugin will not be loaded into Snort.

Although the details will not be covered in this post, the following code snippet is a basic CodecApi that Snort can load. This snippet can be copied and used with only three minor changes. First, in the function ctor, ExCodec should be replaced with the name of the Codec that is being built. Second, EX_NAME must match the Codec's name or Snort will be unable to load this Codec. Third, EX_HELP should be replaced with the general description of this Codec. Once this code snippet has been added, ExCodec is ready to be compiled and plugged into Snort.

```
static Codec* ctor(Module*)
{ return new ExCodec; }

static void dtor(Codec *cd)
{ delete cd; }

static const CodecApi ex_api =
{
    {
        PT_CODEC,
        EX_NAME,
        EX_HELP,
        CDAPI_PLUGIN_V0,
        0,
        nullptr,
        nullptr,
    },
    nullptr, // pointer to a function called during Snort's startup.
    nullptr, // pointer to a function called during Snort's exit.
    nullptr, // pointer to a function called during thread's startup.
    nullptr, // pointer to a function called during thread's destruction.
    ctor, // pointer to the codec constructor.
    dtor, // pointer to the codec destructor.
};

SO_PUBLIC const BaseApi* snort_plugins[] =
{
    &ex_api.base,
    nullptr
};
```

Two example Codecs are available in the extra directory on git and the extra tarball on the Snort page. One of those examples is the Token Ring Codec while the other example is the PIM Codec.

As a final note, there are four more virtual functions that a Codec should implement: encode, format, update, and log. If the functions are not implemented Snort will not throw any errors. However, Snort may also be unable to accomplish some of its basic functionality.

- encode is called whenever Snort actively responds and needs to builds a packet, i.e. whenever a rule using an IPS ACTION like react, reject, or rewrite is triggered. This function is used to build the response packet protocol by protocol.
- format is called when Snort is rebuilding a packet. For instance, every time Snort reassembles a TCP stream or IP fragment, format is called. Generally, this function either swaps any source and destination fields in the protocol or does nothing.
- update is similar to format in that it is called when Snort is reassembling a packet. Unlike format, this function only sets length fields.
- log is called when either the log_codecs logger or a custom logger that calls PacketManager::log_protocols is used when running Snort.

18.5 IPS Actions

Action plugins specify a builtin action in the API which is used to determine verdict. (Conversely, builtin actions don't have an associated plugin function.)

18.6 Developers Guide

Run doc/dev_guide.sh to generate /tmp/dev_guide.html, an annotated guide to the source tree.

18.7 Piglet Test Harness

In order to assist with plugin development, an experimental mode called "piglet" mode is provided. With piglet mode, you can call individual methods for a specific plugin. The piglet tests are specified as Lua scripts. Each piglet test script defines a test for a specific plugin.

Here is a minimal example of a piglet test script for the IPv4 Codec plugin:

```
plugin =
{
    type = "piglet",
    name = "codec::ipv4",
    use_defaults = true,
    test = function()
        local daq_header = DAQHeader.new()
        local raw_buffer = RawBuffer.new("some data")
        local codec_data = CodecData.new()
        local decode_data = DecodeData.new()

        return Codec.decode(
            daq_header,
            raw_buffer,
            codec_data,
            decode_data
        )
    end
}
```

To run snort in piglet mode, first build snort with the ENABLE_PIGLET option turned on (pass the flag -DENABLE_PIGLET:BOOL=ON in cmake).

Then, run the following command:

```
snort --script-path $test_scripts --piglet
```

(where $test_scripts is the directory containing your piglet tests).

The test runner will generate a check-like output, indicating the the results of each test script.

18.8 Piglet Lua API

This section documents the API that piglet exposes to Lua. Refer to the piglet directory in the source tree for examples of usage.

Note: Because of the differences between the Lua and C++ data model and type system, not all parameters map directly to the parameters of the underlying C\++ member functions. Every effort has been made to keep the mappings consist, but there are still some differences. They are documented below.

18.8.1 Plugin Instances

For each test, piglet instantiates plugin specified in the `name` field of the `plugin` table. The virtual methods of the instance are exposed in a table unique to each plugin type. The name of the table is the CamelCase name of the plugin type.

For example, codec plugins have a virtual method called `decode`. This method is called like this:

`Codec.decode(...)`

Codec

- `Codec.get_data_link_type() → { int, int, ... }`
- `Codec.get_protocol_ids() → { int, int, ... }`
- `Codec.decode(DAQHeader, RawBuffer, CodecData, DecodeData) → bool`
- `Codec.log(RawBuffer, uint[lyr_len])`
- `Codec.encode(RawBuffer, EncState, Buffer) → bool`
- `Codec.update(uint[flags_hi], uint[flags_lo], RawBuffer, uint[lyr_len] → int`
- `Codec.format(bool[reverse], RawBuffer, DecodeData)`

Differences:

- In `Codec.update()`, the `(uint64_t) flags` parameter has been split into `flags_hi` and `flags_lo`

Inspector

- `Inspector.configure()`
- `Inspector.tinit()`
- `Inspector.tterm()`
- `Inspector.likes(Packet)`
- `Inspector.eval(Packet)`
- `Inspector.clear(Packet)`
- `Inspector.get_buf_from_key(string[key], Packet, RawBuffer) → bool`
- `Inspector.get_buf_from_id(uint[id], Packet, RawBuffer) → bool`
- `Inspector.get_buf_from_type(uint[type], Packet, RawBuffer) → bool`
- `Inspector.get_splitter(bool[to_server]) → StreamSplitter`

Differences: * In `Inspector.configure()`, the `SnortConfig*` parameter is passed implicitly. * the overloaded `get_buf()` member function has been split into three separate methods.

IpsOption

- `IpsOption.hash() → int`
- `IpsOption.is_relative() → bool`
- `IpsOption.fp_research() → bool`
- `IpsOption.get_cursor_type() → int`
- `IpsOption.eval(Cursor, Packet) → int`
- `IpsOption.action(Packet)`

IpsAction

- `IpsAction.exec(Packet)`

Logger

- `Logger.open()`
- `Logger.close()`
- `Logger.reset()`
- `Logger.alert(Packet, string[message], Event)`
- `Logger.log(Packet, string[message], Event)`

SearchEngine

Currently, SearchEngine does not expose any methods.

SoRule

Currently, SoRule does not expose any methods.

Interface Objects

Many of the plugins take C++ classes and structs as arguments. These objects are exposed to the Lua API as Lua userdata. Exposed objects are instantiated by calling the `new` method from each object's method table.

For example, the DecodeData object can be instantiated and exposed to Lua like this:

```
local decode_data = DecodeData.new(...)
```

Each object also exposes useful methods for getting and setting member variables, and calling the C++ methods contained in the the object. These methods can be accessed using the `:` accessor syntax:

```
decode_data:set({ sp = 80, dp = 3500 })
```

Since this is just syntactic sugar for passing the object as the first parameter of the function `DecodeData.set`, an equivalent form is:

```
decode_data.set(decode_data, { sp = 80, dp = 3500 })
```

or even:

```
DecodeData.set(decode_data, { sp = 80, dp = 3500 })
```

Buffer

- `Buffer.new(string[data]) → Buffer`
- `Buffer.new(uint[length]) → Buffer`
- `Buffer.new(RawBuffer) → Buffer`
- `Buffer:allocate(uint[length]) → bool`
- `Buffer:clear()`

CodecData

- `CodecData.new() → CodecData`
- `CodecData.new(uint[next_prot_id]) → CodecData`

- `CodecData.new(fields) → CodecData`
- `CodecData:get() → fields`
- `CodecData:set(fields)`

`fields` is a table with the following contents:

- `next_prot_id`
- `lyr_len`
- `invalid_bytes`
- `proto_bits`
- `codec_flags`
- `ip_layer_cnt`
- `ip6_extension_count`
- `curr_ip6_extension`
- `ip6_csum_proto`

Cursor

- `Cursor.new() → Cursor`
- `Cursor.new(Packet) → Cursor`
- `Cursor.new(string[data]) → Cursor`
- `Cursor.new(RawBuffer) → Cursor`
- `Cursor:reset()`
- `Cursor:reset(Packet)`
- `Cursor:reset(string[data])`
- `Cursor:reset(RawBuffer)`

DAQHeader

- `DAQHeader.new() → DAQHeader`
- `DAQHeader.new(fields) → DAQHeader`
- `DAQHeader:get() → fields`
- `DAQHeader:set(fields)`

`fields` is a table with the following contents:

- `caplen`
- `pktlen`
- `ingress_index`
- `egress_index`
- `ingress_group`

- egress_group
- flags
- opaque

DecodeData

- DecodeData.new() → DecodeData
- DecodeData.new(fields) → DecodeData
- DecodeData:reset()
- DecodeData:get() → fields
- DecodeData:set(fields)
- DecodeData:set_ipv4_hdr(RawBuffer, uint[offset])

fields is a table with the following contents:

- sp
- dp
- decode_flags
- type

EncState

- EncState.new() → EncState
- EncState.new(uint[flags_lo]) → EncState
- EncState.new(uint[flags_lo], uint[flags_hi]) → EncState
- EncState.new(uint[flags_lo], uint[flags_hi], uint[next_proto]) → EncState
- EncState.new(uint[flags_lo], uint[flags_hi], uint[next_proto], uint[ttl]) → EncState
- EncState.new(uint[flags_lo], uint[flags_hi], uint[next_proto], uint[ttl], uint[dsize]) → EncState

Event

- Event.new() → Event
- Event.new(fields) → Event
- Event:get() → fields
- Event:set(fields)

fields is a table with the following contents:

- event_id
- event_reference
- sig_info

- generator
- id
- rev
- class_id
- priority
- text_rule
- num_services

Flow

- Flow.new() → Flow
- Flow:reset()

Packet

- Packet.new() → Packet
- Packet.new(string[data]) → Packet
- Packet.new(uint[size]) → Packet
- Packet.new(fields) → Packet
- Packet.new(RawBuffer) → Packet
- Packet.new(DAQHeader) → Packet
- Packet:set_decode_data(DecodeData)
- Packet:set_data(uint[offset], uint[length])
- Packet:set_flow(Flow)
- Packet:get() → fields
- Packet:set()
- Packet:set(string[data])
- Packet:set(uint[size])
- Packet:set(fields)
- Packet:set(RawBuffer)
- Packet:set(DAQHeader)

fields is a table with the following contents:

- packet_flags
- xtradata_mask
- proto_bits
- application_protocol_ordinal
- alt_dsize
- num_layers

- iplist_id
- user_policy_id
- ps_proto

Note: Packet.new() and Packet:set() accept multiple arguments of the types described above in any order

RawBuffer

- RawBuffer.new() → RawBuffer
- RawBuffer.new(uint[size]) → RawBuffer
- RawBuffer.new(string[data]) → RawBuffer
- RawBuffer:size() → int
- RawBuffer:resize(uint[size])
- RawBuffer:write(string[data])
- RawBuffer:write(string[data], uint[size])
- RawBuffer:read() → string
- RawBuffer:read(uint[end]) → string
- RawBuffer:read(uint[start], uint[end]) → string

Note: calling RawBuffer.new() with no arguments returns a RawBuffer of size 0

StreamSplitter

- StreamSplitter:scan(Flow, RawBuffer) → int, int
- StreamSplitter:scan(Flow, RawBuffer, uint[len]) → int, int
- StreamSplitter:scan(Flow, RawBuffer, uint[len], uint[flags]) → int, int
- StreamSplitter:reassemble(Flow, uint[total], uint[offset], RawBuffer) → int, RawBuffer
- StreamSplitter:reassemble(Flow, uint[total], uint[offset], RawBuffer, uint[len]) → int, RawBuffer
- StreamSplitter:reassemble(Flow, uint[total], uint[offset], RawBuffer, uint[len], uint[flags]) → int, RawBuffer
- StreamSplitter:finish(Flow) → bool

Note: StreamSplitter does not have a new() method, it must be created by an inspector via Inspector.get_splitter()

19 Coding Style

All new code should try to follow these style guidelines. These are not yet firm so feedback is welcome to get something we can live with.

19.1 General

- Generally try to follow http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml, but there are some differences documented here.
- Each source directory should have a dev_notes.txt file summarizing the key points and design decisions for the code in that directory. These are built into the developers guide.
- Makefile.am and CMakeLists.txt should have the same files listed in alpha order. This makes it easier to maintain both build systems.
- All new code must come with unit tests providing 95% coverage or better.
- Generally, Catch is preferred for tests in the source file and CppUTest is preferred for test executables in a test subdirectory.

19.2 C++ Specific

- Do not use exceptions. Exception-safe code is non-trivial and we have ported legacy code that makes use of exceptions unwise. There are a few exceptions to this rule for the memory manager, shell, etc. Other code should handle errors as errors.
- Do not use dynamic_cast or RTTI. Although compilers are getting better all the time, there is a time and space cost to this that is easily avoided.
- Use smart pointers judiciously as they aren't free. If you would have to roll your own, then use a smart pointer. If you just need a dtor to delete something, write the dtor.
- Prefer *and* over && and *or* over || for new source files.
- Use nullptr instead of NULL.
- Use new, delete, and their [] counterparts instead of malloc and free except where realloc must be used. But try not to use realloc. New and delete can't return nullptr so no need to check. And Snort's memory manager will ensure that we live within our memory budget.
- Use references in lieu of pointers wherever possible.
- Use the order public, protected, private top to bottom in a class declaration.
- Keep inline functions in a class declaration very brief, preferably just one line. If you need a more complex inline function, move the definition below the class declaration.
- The goal is to have highly readable class declarations. The user shouldn't have to sift through implementation details to see what is available to the client.
- Any using statements in source files should be added only after all includes have been declared.

19.3 Naming

- Use camel case for namespaces, classes, and types like WhizBangPdfChecker.
- Use lower case identifiers with underscore separators, e.g. some_function() and my_var.
- Do not start or end variable names with an underscore. This has a good chance of conflicting with macro and/or system definitions.
- Use lower case filenames with underscores.

19.4 Comments

- Write comments sparingly with a mind towards future proofing. Often the comments can be obviated with better code. Clear code is better than a comment.
- Heed Tim Ottinger's Rules on Comments (https://disqus.com/by/tim_ottinger/):
  1. Comments should only say what the code is incapable of saying.
  2. Comments that repeat (or pre-state) what the code is doing must be removed.
  3. If the code CAN say what the comment is saying, it must be changed at least until rule #2 is in force.
- Function comment blocks are generally just noise that quickly becomes obsolete. If you absolutely must comment on parameters, put each on a separate line along with the comment. That way changing the signature may prompt a change to the comments too.
- Use FIXIT (not FIXTHIS or TODO or whatever) to mark things left for a day or even just a minute. That way we can find them easily and won't lose track of them.
- Presently using FIXIT-X where X = A | W | P | H | M | L, indicating analysis, warning, perf, high, med, or low priority. Place A and W comments on the exact warning line so we can match up comments and build output. Supporting comments can be added above.
- Put the copyright(s) and license in a comment block at the top of each source file (.h and .cc). Don't bother with trivial scripts and make foo. Some interesting Lua code should get a comment block too. Copy and paste exactly from src/main.h (don't reformat).
- Put author, description, etc. in separate comment(s) following the license. Do not put such comments in the middle of the license foo. Be sure to put the author line ahead of the header guard to exclude them from the developers guide. Use the following format, and include a mention to the original author if this is derived work:

```
// ips_dnp3_obj.cc author Maya Dagon <mdagon@cisco.com>
// based on work by Ryan Jordan
```

- Each header should have a comment immediately after the header guard to give an overview of the file so the reader knows what's going on.
- Use the following comment on switch cases that intentionally fall through to the next case to suppress compiler warning on known valid cases:

```
// fallthrough
```

19.5 Logging

- Messages intended for the user should not look like debug messages. Eg, the function name should not be included. It is generally unhelpful to include pointers.
- Most debug messages should just be deleted.
- Don't bang your error messages (no !). The user feels bad enough about the problem already w/o you shouting at him.

19.6 Types

- Use logical types to make the code clearer and to help the compiler catch problems. typedef uint16_t Port; bool foo(Port) is way better than int foo(int port).
- Use forward declarations (e.g. struct SnortConfig;) instead of void*.
- Try not to use extern data unless absolutely necessary and then put the extern in an appropriate header. Exceptions for things used in exactly one place like BaseApi pointers.

- Use const liberally. In most cases, const char* s = "foo" should be const char* const s = "foo". The former goes in the initialized data section and the latter in read only data section.
- But use const char s[] = "foo" instead of const char* s = "foo" when possible. The latter form allocates a pointer variable and the data while the former allocates only the data.
- Use static wherever possible to minimize public symbols and eliminate unneeded relocations.
- Declare functions virtual only in the parent class introducing the function (not in a derived class that is overriding the function). This makes it clear which class introduces the function.
- Declare functions as override if they are intended to override a function. This makes it possible to find derived implementations that didn't get updated and therefore won't get called due a change in the parent signature.
- Use bool functions instead of int unless there is truly a need for multiple error returns. The C-style use of zero for success and -1 for error is less readable and often leads to messy code that either ignores the various errors anyway or needlessly and ineffectively tries to do something about them. Generally that code is not updated if new errors are added.

19.7 Macros (aka defines)

- In many cases, even in C++, use #define name "value" instead of a const char* const name = "value" because it will eliminate a symbol from the binary.
- Use inline functions instead of macros where possible (pretty much all cases except where stringification is necessary). Functions offer better typing, avoid re-expansions, and a debugger can break there.
- All macros except simple const values should be wrapped in () and all args should be wrapped in () too to avoid surprises upon expansion. Example:

```
#define SEQ_LT(a,b)  ((int)((a) - (b)) <  0)
```

- Multiline macros should be blocked (i.e. inside { }) to avoid if-else type surprises.

19.8 Formatting

- Try to keep all source files under 2500 lines. 3000 is the max allowed. If you need more lines, chances are that the code needs to be refactored.
- Indent 4 space chars . . . no tabs!
- If you need to indent many times, something could be rewritten or restructured to make it clearer. Fewer indents is generally easier to write, easier to read, and overall better code.
- Braces go on the line immediately following a new scope (function signature, if, else, loop, switch, etc.
- Use consistent spacing and line breaks. Always indent 4 spaces from the breaking line. Keep lines less than 100 chars; it greatly helps readability.

```
No:
    calling_a_func_with_a_long_name(arg1,
                                    arg2,
                                    arg3);


Yes:
    calling_a_func_with_a_long_name(
        arg1, arg2, arg3);
```

- Put function signature on one line, except when breaking for the arg list:

```
No:
    inline
    bool foo()
    { // ...

Yes:
    inline bool foo()
    { // ...
```

- Put conditional code on the line following the if so it is easy to break on the conditional block:

```
No:
    if ( test ) foo();

Yes:
    if ( test )
        foo();
```

19.9 Headers

- Don't hesitate to create a new header if it is needed. Don't lump unrelated stuff into an header because it is convenient.
- Write header guards like this (leading underscores are reserved for system stuff). In my_header.h:

```
#ifndef MY_HEADER_H
#define MY_HEADER_H
// ...
#endif
```

- Includes from a different directory should specify parent directory. This makes it clear exactly what is included and avoids the primordial soup that results from using -I this -I that -I the_other_thing ... .

```
// given:
src/foo/foo.cc
src/bar/bar.cc
src/bar/baz.cc

// in baz.cc
#include "bar.h"

// in foo.cc
#include "bar/bar.h"
```

- Includes within installed headers should specify parent directory.
- Just because it is a #define doesn't mean it goes in a header. Everything should be scoped as tightly as possible. Shared implementation declarations should go in a separate header from the interface. And so on.
- All .cc files should include config.h with the standard block shown below immediately following the initial comment blocks and before anything else. This presents a consistent view of all included header files as well as access to any other configure-time definitions. No .h files should include config.h unless they are guaranteed to be local header files (never installed).

```
#ifdef HAVE_CONFIG_H
#include "config.h"
#endif
```

- A .cc should include its own .h before any others aside from the aforementioned config.h (including system headers). This ensures that the header stands on its own and can be used by clients without include prerequisites and the developer will be the first to find a dependency issue.
- Split headers included from the local directory into a final block of headers. For a .cc file, the final order of sets of header includes should look like this:
  1. config.h
  2. its own .h file
  3. system headers (.h/.hpp/.hxx)
  4. C++ standard library headers (no file extension)
  5. Snort headers external to the local directory (path-prefixed)
  6. Snort headers in the local directory
- Include required headers, all required headers, and nothing but required headers. Don't just clone a bunch of headers because it is convenient.
- Keep includes in alphabetical order. This makes it easier to maintain, avoid duplicates, etc.
- Do not put using statements in headers unless they are tightly scoped.

19.10 Warnings

- With g++, use at least these compiler flags:

```
-Wall -Wextra -pedantic -Wformat -Wformat-security
-Wunused-but-set-variable -Wno-deprecated-declarations
-fsanitize=address -fno-omit-frame-pointer
```

- With clang, use at least these compiler flags:

```
-Wall -Wextra -pedantic -Wformat -Wformat-security
-Wno-deprecated-declarations
-fsanitize=address -fno-omit-frame-pointer
```

- Two macros (PADDING_GUARD_BEGIN and PADDING_GUARD_END) are provided by utils/cpp_macros.h. These should be used to surround any structure used as a hash key with a raw comparator or that would otherwise suffer from unintentional padding. A compiler warning will be generated if any structure definition is automatically padded between the macro invocations.
- Then Fix All Warnings and Aborts. None Allowed.

19.11 Uncrustify

Currently using uncrustify from at https://github.com/bengardner/uncrustify to reformat legacy code and anything that happens to need a makeover at some point.

The working config is crusty.cfg in the top level directory. It does well but will munge some things. Specially formatted INDENT-OFF comments were added in 2 places to avoid a real mess.

You can use uncrustify something like this:

```
uncrustify -c crusty.cfg --replace file.cc
```

20 Reference

20.1 Build Options

The options listed below must be explicitly enabled so they are built into the Snort binary. For a full list of build options, run ./configure --help.

- **--enable-shell**: enable building local and remote command line shell support.
- **--enable-tsc-clock**: use the TSC register on x86 systems for improved performance of latency and profiler features.

These options are built only if the required libraries and headers are present. There is no need to explicitly enable.

- **flatbuffers**: for an alternative perf_monitor logging format.
- **hyperscan** >= 4.4.0: for the regex and sd_pattern rule options and the hyperscan search engine.
- **iconv**: for converting UTF16-LE filenames to UTF8 (usually included in glibc)
- **lzma**: for decompression of SWF and PDF files.
- **safec**: for additional runtime error checking of some memory copy operations.

If you need to use headers and/or libraries in non-standard locations, you can use these options:

- **--with-pkg-includes**: specify the directory containing the package headers.
- **--with-pkg-libraries**: specify the directory containing the package libraries.

These can be used for pcap, luajit, pcre, dnet, daq, lzma, openssl, flatbuffers, iconv, and hyperscan packages. For more information on these libraries see the Getting Started section of the manual.

20.2 Environment Variables

- **HOSTTYPE**: optional string that is output with the version at end of line.
- **LUA_PATH**: you must export as follows so LuaJIT can find required files.

```
LUA_PATH=$install_dir/include/snort/lua/\?.lua\;\;
```

- **SNORT_IGNORE**: the list of symbols Snort should ignore when parsing the Lua conf. Unknown symbols not in SNORT_IGNORE will cause warnings with --warn-unknown or fatals with --warn-unknown --pedantic.
- **SNORT_LUA_PATH**: an optional path where Snort can find supplemental conf files such as classification.lua.
- **SNORT_PROMPT**: the character sequence that is printed at startup, shutdown, and in the shell. The default is the mini-pig: o")~ .
- **SNORT_PLUGIN_PATH**: an optional path where Snort can find supplemental shared libraries. This is only used when Snort is building manuals. Modules in supplemental shared libraries will be added to the manuals.

20.3 Command Line Options

- **-?** <option prefix> output matching command line option quick help (same as --help-options) (optional)
- **-A** <mode> set alert mode: none, cmg, or alert_*
- **-B** <mask> obfuscated IP addresses in alerts and packet dumps using CIDR mask
- **-C** print out payloads with character data only (no hex)
- **-c** <conf> use this configuration
- **-D** run Snort in background (daemon) mode
- **-d** dump the Application Layer
- **-e** display the second layer header info
- **-f** turn off fflush() calls after binary log writes
- **-G** <0xid> (same as --logid) (0:65535)
- **-g** <gname> run snort gid as <gname> group (or gid) after initialization
- **-H** make hash tables deterministic
- **-i** <iface>… list of interfaces
- **-j** <port> to listen for Telnet connections
- **-k** <mode> checksum mode; default is all (all|noip|notcp|noudp|noicmp|none)
- **-L** <mode> logging mode (none, dump, pcap, or log_*)
- **-l** <logdir> log to this directory instead of current directory
- **-M** log messages to syslog (not alerts)
- **-m** <umask> set umask = <umask> (0:)
- **-n** <count> stop after count packets (0:)
- **-O** obfuscate the logged IP addresses
- **-Q** enable inline mode operation
- **-q** quiet mode - Don't show banner and status report
- **-R** <rules> include this rules file in the default policy
- **-r** <pcap>… (same as --pcap-list)
- **-S** <x=v> set config variable x equal to value v
- **-s** <snap> (same as --snaplen); default is 1514 (68:65535)
- **-T** test and report on the current Snort configuration
- **-t** <dir> chroots process to <dir> after initialization
- **-U** use UTC for timestamps
- **-u** <uname> run snort as <uname> or <uid> after initialization
- **-V** (same as --version)
- **-v** be verbose
- **-W** lists available interfaces

- **-X** dump the raw packet data starting at the link layer
- **-x** same as --pedantic
- **-y** include year in timestamp in the alert and log files
- **-z** <count> maximum number of packet threads (same as --max-packet-threads); 0 gets the number of CPU cores reported by the system; default is 1 (0:)
- **--alert-before-pass** process alert, drop, sdrop, or reject before pass; default is pass before alert, drop,. . .
- **--bpf** <filter options> are standard BPF options, as seen in TCPDump
- **--c2x** output hex for given char (see also --x2c)
- **--control-socket** <file> to create unix socket
- **--create-pidfile** create PID file, even when not in Daemon mode
- **--daq** <type> select packet acquisition module (default is pcap)
- **--daq-dir** <dir> tell snort where to find desired DAQ
- **--daq-list** list packet acquisition modules available in optional dir, default is static modules only
- **--daq-var** <name=value> specify extra DAQ configuration variable
- **--dirty-pig** don't flush packets on shutdown
- **--dump-builtin-rules** [<module prefix>] output stub rules for selected modules (optional)
- **--dump-dynamic-rules** output stub rules for all loaded rules libraries
- **--dump-defaults** [<module prefix>] output module defaults in Lua format (optional)
- **--dump-version** output the version, the whole version, and only the version
- **--enable-inline-test** enable Inline-Test Mode Operation
- **--gen-msg-map** dump builtin rules in gen-msg.map format for use by other tools
- **--help** list command line options
- **--help-commands** [<module prefix>] output matching commands (optional)
- **--help-config** [<module prefix>] output matching config options (optional)
- **--help-counts** [<module prefix>] output matching peg counts (optional)
- **--help-module** <module> output description of given module
- **--help-modules** list all available modules with brief help
- **--help-options** [<option prefix>] output matching command line option quick help (same as -?) (optional)
- **--help-plugins** list all available plugins with brief help
- **--help-signals** dump available control signals
- **--id-offset** offset to add to instance IDs when logging to files (0:65535)
- **--id-subdir** create/use instance subdirectories in logdir instead of instance filename prefix
- **--id-zero** use id prefix / subdirectory even with one packet thread
- **--list-buffers** output available inspection buffers
- **--list-builtin** [<module prefix>] output matching builtin rules (optional)

- **--list-gids** [<module prefix>] output matching generators (optional)
- **--list-modules** [<module type>] list all known modules of given type (optional)
- **--list-plugins** list all known plugins
- **--lua** <chunk> extend/override conf with chunk; may be repeated
- **--logid** <0xid> log Identifier to uniquely id events for multiple snorts (same as -G) (0:65535)
- **--markup** output help in asciidoc compatible format
- **--max-packet-threads** <count> configure maximum number of packet threads (same as -z) (0:)
- **--mem-check** like -T but also compile search engines
- **--nostamps** don't include timestamps in log file names
- **--nolock-pidfile** do not try to lock Snort PID file
- **--pause** wait for resume/quit command before processing packets/terminating
- **--parsing-follows-files** parse relative paths from the perspective of the current configuration file
- **--pcap-file** <file> file that contains a list of pcaps to read - read mode is implied
- **--pcap-list** <list> a space separated list of pcaps to read - read mode is implied
- **--pcap-dir** <dir> a directory to recurse to look for pcaps - read mode is implied
- **--pcap-filter** <filter> filter to apply when getting pcaps from file or directory
- **--pcap-loop** <count> read all pcaps <count> times; 0 will read until Snort is terminated (-1:)
- **--pcap-no-filter** reset to use no filter when getting pcaps from file or directory
- **--pcap-reload** if reading multiple pcaps, reload snort config between pcaps
- **--pcap-show** print a line saying what pcap is currently being read
- **--pedantic** warnings are fatal
- **--plugin-path** <path> where to find plugins
- **--process-all-events** process all action groups
- **--rule** <rules> to be added to configuration; may be repeated
- **--rule-to-hex** output so rule header to stdout for text rule on stdin
- **--rule-to-text** output plain so rule header to stdout for text rule on stdin (16)
- **--run-prefix** <pfx> prepend this to each output file
- **--script-path** <path> to a luajit script or directory containing luajit scripts
- **--shell** enable the interactive command line
- **--piglet** enable piglet test harness mode
- **--show-plugins** list module and plugin versions
- **--skip** <n> skip 1st n packets (0:)
- **--snaplen** <snap> set snaplen of packet (same as -s) (68:65535)
- **--stdin-rules** read rules from stdin until EOF or a line starting with END is read
- **--talos** enable Talos inline rule test mode (same as --tweaks talos -Q -q)

- **--treat-drop-as-alert** converts drop, sdrop, and reject rules into alert rules during startup
- **--treat-drop-as-ignore** use drop, sdrop, and reject rules to ignore session traffic when not inline
- **--tweaks** tune configuration
- **--catch-test** comma separated list of cat unit test tags or *all*
- **--version** show version number (same as -V)
- **--warn-all** enable all warnings
- **--warn-conf** warn about configuration issues
- **--warn-daq** warn about DAQ issues, usually related to mode
- **--warn-flowbits** warn about flowbits that are checked but not set and vice-versa
- **--warn-hosts** warn about host table issues
- **--warn-plugins** warn about issues that prevent plugins from loading
- **--warn-rules** warn about duplicate rules and rule parsing issues
- **--warn-scripts** warn about issues discovered while processing Lua scripts
- **--warn-symbols** warn about unknown symbols in your Lua config
- **--warn-vars** warn about variable definition and usage issues
- **--x2c** output ASCII char for given hex (see also --c2x)
- **--x2s** output ASCII string for given byte code (see also --x2c)
- **--trace** turn on main loop debug trace

20.4 Configuration

- interval **ack.~range**: check if TCP ack value is *value* | *min*<>*max* | <*max* | >*min* { 0: }
- int **active.attempts** = 0: number of TCP packets sent per response (with varying sequence numbers) { 0:20 }
- string **active.device**: use *ip* for network layer responses or *eth0* etc for link layer
- string **active.dst_mac**: use format *01:23:45:67:89:ab*
- int **active.max_responses** = 0: maximum number of responses { 0: }
- int **active.min_interval** = 255: minimum number of seconds between responses { 1:255 }
- multi **alert_csv.fields** = timestamp pkt_num proto pkt_gen pkt_len dir src_ap dst_ap rule action: selected fields will be output in given order left to right { action | class | b64_data | dir | dst_addr | dst_ap | dst_port | eth_dst | eth_len | eth_src | eth_type | gid | icmp_code | icmp_id | icmp_seq | icmp_type | iface | ip_id | ip_len | msg | mpls | pkt_gen | pkt_len | pkt_num | priority | proto | rev | rule | seconds | service | sid | src_addr | src_ap | src_port | target | tcp_ack | tcp_flags | tcp_len | tcp_seq | tcp_win | timestamp | tos | ttl | udp_len | vlan }
- bool **alert_csv.file** = false: output to alert_csv.txt instead of stdout
- int **alert_csv.limit** = 0: set maximum size in MB before rollover (0 is unlimited) { 0: }
- string **alert_csv.separator** = , : separate fields with this character sequence
- bool **alert_ex.upper** = false: true/false → convert to upper/lower case
- bool **alert_fast.file** = false: output to alert_fast.txt instead of stdout

- int **alert_fast.limit** = 0: set maximum size in MB before rollover (0 is unlimited) { 0: }
- bool **alert_fast.packet** = false: output packet dump with alert
- bool **alert_full.file** = false: output to alert_full.txt instead of stdout
- int **alert_full.limit** = 0: set maximum size in MB before rollover (0 is unlimited) { 0: }
- multi **alert_json.fields** = timestamp pkt_num proto pkt_gen pkt_len dir src_ap dst_ap rule action: selected fields will be output in given order left to right { action | class | b64_data | dir | dst_addr | dst_ap | dst_port | eth_dst | eth_len | eth_src | eth_type | gid | icmp_code | icmp_id | icmp_seq | icmp_type | iface | ip_id | ip_len | msg | mpls | pkt_gen | pkt_len | pkt_num | priority | proto | rev | rule | seconds | service | sid | src_addr | src_ap | src_port | target | tcp_ack | tcp_flags | tcp_len | tcp_seq | tcp_win | timestamp | tos | ttl | udp_len | vlan }
- bool **alert_json.file** = false: output to alert_json.txt instead of stdout
- int **alert_json.limit** = 0: set maximum size in MB before rollover (0 is unlimited) { 0: }
- string **alert_json.separator** = , : separate fields with this character sequence
- bool **alerts.alert_with_interface_name** = false: include interface in alert info (fast, full, or syslog only)
- bool **alerts.default_rule_state** = true: enable or disable ips rules
- int **alerts.detection_filter_memcap** = 1048576: set available bytes of memory for detection_filters { 0: }
- int **alerts.event_filter_memcap** = 1048576: set available bytes of memory for event_filters { 0: }
- string **alert_sfsocket.file**: name of unix socket file
- int **alert_sfsocket.rules[].gid** = 1: rule generator ID { 1: }
- int **alert_sfsocket.rules[].sid** = 1: rule signature ID { 1: }
- bool **alerts.log_references** = false: include rule references in alert info (full only)
- string **alerts.order** = pass drop alert log: change the order of rule action application
- int **alerts.rate_filter_memcap** = 1048576: set available bytes of memory for rate_filters { 0: }
- string **alerts.reference_net**: set the CIDR for homenet (for use with -l or -B, does NOT change $HOME_NET in IDS mode)
- bool **alerts.stateful** = false: don't alert w/o established session (note: rule action still taken)
- string **alerts.tunnel_verdicts**: let DAQ handle non-allow verdicts for gtp|teredo|6in4|4in6|4in4|6in6|gre|mpls traffic
- enum **alert_syslog.facility** = auth: part of priority applied to each message { auth | authpriv | daemon | user | local0 | local1 | local2 | local3 | local4 | local5 | local6 | local7 }
- enum **alert_syslog.level** = info: part of priority applied to each message { emerg | alert | crit | err | warning | notice | info | debug }
- multi **alert_syslog.options**: used to open the syslog connection { cons | ndelay | perror | pid }
- string **appid.app_detector_dir**: directory to load appid detectors from
- int **appid.app_stats_period** = 300: time period for collecting and logging appid statistics { 0: }
- int **appid.app_stats_rollover_size** = 20971520: max file size for appid stats before rolling over the log file { 0: }
- int **appid.app_stats_rollover_time** = 86400: max time period for collection appid stats before rolling over the log file { 0: }
- bool **appid.debug** = false: enable appid debug logging
- bool **appid.dump_ports** = false: enable dump of appid port information
- int **appid.first_decrypted_packet_debug** = 0: the first packet of an already decrypted SSL flow (debug single session only) { 0: }

- int **appid.instance_id** = 0: instance id - ignored { 0: }
- bool **appid.log_all_sessions** = false: enable logging of all appid sessions
- bool **appid.log_stats** = false: enable logging of appid statistics
- int **appid.memcap** = 0: disregard - not implemented { 0: }
- string **appids.~**: comma separated list of application names
- string **appid.tp_appid_config**: path to third party appid configuration file
- string **appid.tp_appid_path**: path to third party appid dynamic library
- int **appid.trace**: mask for enabling debug traces in module
- ip4 **arp_spoof.hosts[].ip**: host ip address
- mac **arp_spoof.hosts[].mac**: host mac address
- int **asn1.absolute_offset**: absolute offset from the beginning of the packet { 0: }
- implied **asn1.bitstring_overflow**: detects invalid bitstring encodings that are known to be remotely exploitable
- implied **asn1.double_overflow**: detects a double ASCII encoding that is larger than a standard buffer
- int **asn1.oversize_length**: compares ASN.1 type lengths with the supplied argument { 0: }
- implied **asn1.print**: dump decode data to console; always true
- int **asn1.relative_offset**: relative offset from the cursor
- int **attribute_table.max_hosts** = 1024: maximum number of hosts in attribute table { 32:207551 }
- int **attribute_table.max_metadata_services** = 8: maximum number of services in rule metadata { 1:256 }
- int **attribute_table.max_services_per_host** = 8: maximum number of services per host entry in attribute table { 1:65535 }
- int **base64_decode.bytes**: number of base64 encoded bytes to decode { 1: }
- int **base64_decode.offset** = 0: bytes past start of buffer to start decoding { 0: }
- implied **base64_decode.relative**: apply offset to cursor instead of start of buffer
- enum **binder[].use.action** = inspect: what to do with matching traffic { reset | block | allow | inspect }
- string **binder[].use.file**: use configuration in given file
- string **binder[].use.inspection_policy**: use inspection policy from given file
- string **binder[].use.ips_policy**: use ips policy from given file
- string **binder[].use.name**: symbol name (defaults to type)
- string **binder[].use.network_policy**: use network policy from given file
- string **binder[].use.service**: override automatic service identification
- string **binder[].use.type**: select module for binding
- addr_list **binder[].when.dst_nets**: list of destination networks
- bit_list **binder[].when.dst_ports**: list of destination ports { 65535 }
- int **binder[].when.dst_zone**: destination zone { 0:2147483647 }
- bit_list **binder[].when.ifaces**: list of interface indices { 255 }
- int **binder[].when.ips_policy_id** = 0: unique ID for selection of this config by external logic { 0: }

- addr_list **binder[].when.nets**: list of networks
- bit_list **binder[].when.ports**: list of ports { 65535 }
- enum **binder[].when.proto**: protocol { any | ip | icmp | tcp | udp | user | file }
- enum **binder[].when.role** = any: use the given configuration on one or any end of a session { client | server | any }
- string **binder[].when.service**: override default configuration
- addr_list **binder[].when.src_nets**: list of source networks
- bit_list **binder[].when.src_ports**: list of source ports { 65535 }
- int **binder[].when.src_zone**: source zone { 0:2147483647 }
- bit_list **binder[].when.vlans**: list of VLAN IDs { 4095 }
- interval **bufferlen.~range**: check that length of current buffer is in given range { 0:65535 }
- int **byte_extract.align** = 0: round the number of converted bytes up to the next 2- or 4-byte boundary { 0:4 }
- implied **byte_extract.big**: big endian
- int **byte_extract.bitmask**: applies as an AND to the extracted value before storage in *name* { 0x1:0xFFFFFFFF }
- int **byte_extract.~count**: number of bytes to pick up from the buffer { 1:10 }
- implied **byte_extract.dce**: dcerpc2 determines endianness
- implied **byte_extract.dec**: convert from decimal string
- implied **byte_extract.hex**: convert from hex string
- implied **byte_extract.little**: little endian
- int **byte_extract.multiplier** = 1: scale extracted value by given amount { 1:65535 }
- string **byte_extract.~name**: name of the variable that will be used in other rule options
- implied **byte_extract.oct**: convert from octal string
- int **byte_extract.~offset**: number of bytes into the buffer to start processing { -65535:65535 }
- implied **byte_extract.relative**: offset from cursor instead of start of buffer
- implied **byte_extract.string**: convert from string
- int **byte_jump.align** = 0: round the number of converted bytes up to the next 2- or 4-byte boundary { 0:4 }
- implied **byte_jump.big**: big endian
- int **byte_jump.bitmask**: applies as an AND prior to evaluation { 0x1:0xFFFFFFFF }
- int **byte_jump.~count**: number of bytes to pick up from the buffer { 0:10 }
- implied **byte_jump.dce**: dcerpc2 determines endianness
- implied **byte_jump.dec**: convert from decimal string
- implied **byte_jump.from_beginning**: jump from start of buffer instead of cursor
- implied **byte_jump.from_end**: jump backward from end of buffer
- implied **byte_jump.hex**: convert from hex string
- implied **byte_jump.little**: little endian
- int **byte_jump.multiplier** = 1: scale extracted value by given amount { 1:65535 }

- implied **byte_jump.oct**: convert from octal string
- string **byte_jump.~offset**: variable name or number of bytes into the buffer to start processing
- string **byte_jump.post_offset**: skip forward or backward (positive or negative value) by variable name or number of bytes after the other jump options have been applied
- implied **byte_jump.relative**: offset from cursor instead of start of buffer
- implied **byte_jump.string**: convert from string
- int **byte_math.bitmask**: applies as bitwise AND to the extracted value before storage in *name* { 0x1:0xFFFFFFFF }
- int **byte_math.bytes**: number of bytes to pick up from the buffer { 1:10 }
- implied **byte_math.dce**: dcerpc2 determines endianness
- enum **byte_math.endian**: specify big/little endian { big|little }
- string **byte_math.offset**: number of bytes into the buffer to start processing
- enum **byte_math.oper**: mathematical operation to perform { +|-|*|/|<<|>> }
- implied **byte_math.relative**: offset from cursor instead of start of buffer
- string **byte_math.result**: name of the variable to store the result
- string **byte_math.rvalue**: value to use mathematical operation against
- enum **byte_math.string**: convert extracted string to dec/hex/oct { hex|dec|oct }
- implied **byte_test.big**: big endian
- int **byte_test.bitmask**: applies as an AND prior to evaluation { 0x1:0xFFFFFFFF }
- string **byte_test.~compare**: variable name or value to test the converted result against
- int **byte_test.~count**: number of bytes to pick up from the buffer { 1:10 }
- implied **byte_test.dce**: dcerpc2 determines endianness
- implied **byte_test.dec**: convert from decimal string
- implied **byte_test.hex**: convert from hex string
- implied **byte_test.little**: little endian
- implied **byte_test.oct**: convert from octal string
- string **byte_test.~offset**: variable name or number of bytes into the payload to start processing
- string **byte_test.~operator**: operation to perform to test the value
- implied **byte_test.relative**: offset from cursor instead of start of buffer
- implied **byte_test.string**: convert from string
- string **classifications[].name**: name used with classtype rule option
- int **classifications[].priority** = 1: default priority for class { 0: }
- string **classifications[].text**: description of class
- string **classtype.~**: classification for this rule
- string **content.~data**: data to match
- string **content.depth**: var or maximum number of bytes to search from beginning of buffer

- string **content.distance**: var or number of bytes from cursor to start search
- int **content.fast_pattern_length**: maximum number of characters from this content the fast pattern matcher should use { 1: }
- int **content.fast_pattern_offset** = 0: number of leading characters of this content the fast pattern matcher should exclude { 0: }
- implied **content.fast_pattern**: use this content in the fast pattern matcher instead of the content selected by default
- implied **content.nocase**: case insensitive match
- string **content.offset**: var or number of bytes from start of buffer to start search
- string **content.within**: var or maximum number of bytes to search from cursor
- implied **cvs.invalid-entry**: looks for an invalid Entry string
- string **daq.input_spec**: input specification
- int **daq.instances[].id**: instance ID (required) { 0: }
- string **daq.instances[].input_spec**: input specification
- string **daq.instances[].variables[].str**: string parameter
- string **daq.module**: DAQ module to use
- string **daq.module_dirs[].str**: string parameter
- bool **daq.no_promisc** = false: whether to put DAQ device into promiscuous mode
- int **daq.snaplen**: set snap length (same as -s) { 0:65535 }
- string **daq.variables[].str**: string parameter
- select **data_log.key** = http_request_header_event : name of the event to log { http_request_header_event | http_response_header_event }
- int **data_log.limit** = 0: set maximum size in MB before rollover (0 is unlimited) { 0: }
- implied **dce_iface.any_frag**: match on any fragment
- string **dce_iface.uuid**: match given dcerpc uuid
- interval **dce_iface.version**: interface version { 0: }
- string **dce_opnum.~**: match given dcerpc operation number, range or list
- bool **dce_smb.disable_defrag** = false: Disable DCE/RPC defragmentation
- int **dce_smb.max_frag_len** = 65535: Maximum fragment size for defragmentation { 1514:65535 }
- enum **dce_smb.policy** = WinXP: Target based policy to use { Win2000 | WinXP | WinVista | Win2003 | Win2008 | Win7 | Samba | Samba-3.0.37 | Samba-3.0.22 | Samba-3.0.20 }
- int **dce_smb.reassemble_threshold** = 0: Minimum bytes received before performing reassembly { 0:65535 }
- int **dce_smb.smb_file_depth** = 16384: SMB file depth for file data { -1: }
- enum **dce_smb.smb_file_inspection** = off: SMB file inspection { off | on | only }
- enum **dce_smb.smb_fingerprint_policy** = none: Target based SMB policy to use { none | client | server | both }
- string **dce_smb.smb_invalid_shares**: SMB shares to alert on
- bool **dce_smb.smb_legacy_mode** = false: inspect only SMBv1
- int **dce_smb.smb_max_chain** = 3: SMB max chain size { 0:255 }

- int **dce_smb.smb_max_compound** = 3: SMB max compound size { 0:255 }
- int **dce_smb.trace**: mask for enabling debug traces in module
- multi **dce_smb.valid_smb_versions** = all: Valid SMB versions { v1 | v2 | all }
- bool **dce_tcp.disable_defrag** = false: Disable DCE/RPC defragmentation
- int **dce_tcp.max_frag_len** = 65535: Maximum fragment size for defragmentation { 1514:65535 }
- enum **dce_tcp.policy** = WinXP: Target based policy to use { Win2000 | WinXP | WinVista | Win2003 | Win2008 | Win7 | Samba | Samba-3.0.37 | Samba-3.0.22 | Samba-3.0.20 }
- int **dce_tcp.reassemble_threshold** = 0: Minimum bytes received before performing reassembly { 0:65535 }
- bool **dce_udp.disable_defrag** = false: Disable DCE/RPC defragmentation
- int **dce_udp.max_frag_len** = 65535: Maximum fragment size for defragmentation { 1514:65535 }
- int **dce_udp.trace**: mask for enabling debug traces in module
- int **decode.trace**: mask for enabling debug traces in module
- int **detection.asn1** = 256: maximum decode nodes { 1: }
- bool **detection.enable_address_anomaly_checks** = false: enable check and alerting of address anomalies
- int **detection_filter.count**: hits in interval before allowing the rule to fire { 1: }
- int **detection_filter.seconds**: length of interval to count hits { 1: }
- enum **detection_filter.track**: track hits by source or destination IP address { by_src | by_dst }
- int **detection.offload_limit** = 99999: minimum sizeof PDU to offload fast pattern search (defaults to disabled) { 0: }
- int **detection.offload_threads** = 0: maximum number of simultaneous offloads (defaults to disabled) { 0: }
- bool **detection.pcre_enable** = true: disable pcre pattern matching
- int **detection.pcre_match_limit** = 1500: limit pcre backtracking, -1 = max, 0 = off { -1:1000000 }
- int **detection.pcre_match_limit_recursion** = 1500: limit pcre stack consumption, -1 = max, 0 = off { -1:10000 }
- int **detection.trace**: mask for enabling debug traces in module
- bool **dnp3.check_crc** = false: validate checksums in DNP3 link layer frames
- string **dnp3_func.~**: match DNP3 function code or name
- string **dnp3_ind.~**: match given DNP3 indicator flags
- int **dnp3_obj.group** = 0: match given DNP3 object header group { 0:255 }
- int **dnp3_obj.var** = 0: match given DNP3 object header var { 0:255 }
- string **domain_filter.file**: file with list of domains identifying hosts to be filtered
- string **domain_filter.hosts**: list of domains identifying hosts to be filtered
- int **dpx.max** = 0: maximum payload before alert { 0:65535 }
- port **dpx.port**: port to check
- interval **dsize.~range**: check if packet payload size is in the given range { 0:65535 }
- bool **esp.decode_esp** = false: enable for inspection of esp traffic that has authentication but not encryption
- int **event_filter[].count** = 0: number of events in interval before tripping; -1 to disable { -1: }

- int **event_filter[].gid** = 1: rule generator ID { 0: }
- string **event_filter[].ip**: restrict filter to these addresses according to track
- int **event_filter[].seconds** = 0: count interval { 0: }
- int **event_filter[].sid** = 1: rule signature ID { 0: }
- enum **event_filter[].track**: filter only matching source or destination addresses { by_src | by_dst }
- enum **event_filter[].type**: 1st count events | every count events | once after count events { limit | threshold | both }
- int **event_queue.log** = 3: maximum events to log { 1: }
- int **event_queue.max_queue** = 8: maximum events to queue { 1: }
- enum **event_queue.order_events** = content_length: criteria for ordering incoming events { priority|content_length }
- bool **event_queue.process_all_events** = false: process just first action group or all action groups
- string **file_connector.connector**: connector name
- enum **file_connector.direction**: usage { receive | transmit | duplex }
- enum **file_connector.format**: file format { binary | text }
- string **file_connector.name**: channel name
- int **file_id.block_timeout** = 86400: stop blocking after this many seconds { 0: }
- bool **file_id.block_timeout_lookup** = false: block if lookup times out
- int **file_id.capture_block_size** = 32768: file capture block size in bytes { 8: }
- int **file_id.capture_max_size** = 1048576: stop file capture beyond this point { 0: }
- int **file_id.capture_memcap** = 100: memcap for file capture in megabytes { 0: }
- int **file_id.capture_min_size** = 0: stop file capture if file size less than this { 0: }
- bool **file_id.enable_capture** = false: enable file capture
- bool **file_id.enable_signature** = true: enable signature calculation
- bool **file_id.enable_type** = true: enable type ID
- bool **file_id.file_policy[].use.enable_file_capture** = false: true/false → enable/disable file capture
- bool **file_id.file_policy[].use.enable_file_signature** = false: true/false → enable/disable file signature
- bool **file_id.file_policy[].use.enable_file_type** = false: true/false → enable/disable file type identification
- enum **file_id.file_policy[].use.verdict** = unknown: what to do with matching traffic { unknown | log | stop | block | reset }
- int **file_id.file_policy[].when.file_type_id** = 0: unique ID for file type in file magic rule { 0: }
- string **file_id.file_policy[].when.sha256**: SHA 256
- string **file_id.file_rules[].category**: file type category
- string **file_id.file_rules[].group**: comma separated list of groups associated with file type
- int **file_id.file_rules[].id** = 0: file type id { 0: }
- string **file_id.file_rules[].magic[].content**: file magic content
- int **file_id.file_rules[].magic[].offset** = 0: file magic offset { 0: }
- string **file_id.file_rules[].msg**: information about the file type

- int **file_id.file_rules[].rev** = 0: rule revision { 0: }
- string **file_id.file_rules[].type**: file type name
- string **file_id.file_rules[].version**: file type version
- int **file_id.lookup_timeout** = 2: give up on lookup after this many seconds { 0: }
- int **file_id.max_files_cached** = 65536: maximal number of files cached in memory { 8: }
- int **file_id.show_data_depth** = 100: print this many octets { 0: }
- int **file_id.signature_depth** = 10485760: stop signature at this point { 0: }
- bool **file_id.trace_signature** = false: enable runtime dump of signature info
- bool **file_id.trace_stream** = false: enable runtime dump of file data
- bool **file_id.trace_type** = false: enable runtime dump of type info
- int **file_id.type_depth** = 1460: stop type ID at this point { 0: }
- int **file_id.verdict_delay** = 0: number of queries to return final verdict { 0: }
- bool **file_log.log_pkt_time** = true: log the packet time when event generated
- bool **file_log.log_sys_time** = false: log the system time when event generated
- string **file_type.~**: list of file type IDs to match
- string **flags.~mask_flags**: these flags are don't cares
- string **flags.~test_flags**: these flags are tested
- string **flowbits.~arg1**: bits or group
- string **flowbits.~arg2**: group if arg1 is bits
- string **flowbits.~command**: set|reset|isset|etc.
- implied **flow.established**: match only during data transfer phase
- implied **flow.from_client**: same as to_server
- implied **flow.from_server**: same as to_client
- implied **flow.no_frag**: match on raw packets only
- implied **flow.no_stream**: match on raw packets only
- implied **flow.not_established**: match only outside data transfer phase
- implied **flow.only_frag**: match on defragmented packets only
- implied **flow.only_stream**: match on reassembled packets only
- implied **flow.stateless**: match regardless of stream state
- implied **flow.to_client**: match on server responses
- implied **flow.to_server**: match on client requests
- string **fragbits.~flags**: these flags are tested
- interval **fragoffset.~range**: check if ip fragment offset is in given range { 0:8192 }
- bool **ftp_client.bounce** = false: check for bounces
- addr **ftp_client.bounce_to[].address** = 1.0.0.0/32: allowed IP address in CIDR format

- port **ftp_client.bounce_to[].last_port**: optional allowed range from port to last_port inclusive { 0: }
- port **ftp_client.bounce_to[].port** = 20: allowed port { 1: }
- bool **ftp_client.ignore_telnet_erase_cmds** = false: ignore erase character and erase line commands when normalizing
- int **ftp_client.max_resp_len** = -1: maximum FTP response accepted by client { -1: }
- bool **ftp_client.telnet_cmds** = false: detect Telnet escape sequences on FTP control channel
- bool **ftp_server.check_encrypted** = false: check for end of encryption
- string **ftp_server.chk_str_fmt**: check the formatting of the given commands
- string **ftp_server.cmd_validity[].command**: command string
- string **ftp_server.cmd_validity[].format**: format specification
- int **ftp_server.cmd_validity[].length** = 0: specify non-default maximum for command { 0: }
- string **ftp_server.data_chan_cmds**: check the formatting of the given commands
- string **ftp_server.data_rest_cmds**: check the formatting of the given commands
- string **ftp_server.data_xfer_cmds**: check the formatting of the given commands
- int **ftp_server.def_max_param_len** = 100: default maximum length of commands handled by server; 0 is unlimited { 1: }
- string **ftp_server.directory_cmds[].dir_cmd**: directory command
- int **ftp_server.directory_cmds[].rsp_code** = 200: expected successful response code for command { 200: }
- string **ftp_server.encr_cmds**: check the formatting of the given commands
- bool **ftp_server.encrypted_traffic** = false: check for encrypted Telnet and FTP
- string **ftp_server.file_get_cmds**: check the formatting of the given commands
- string **ftp_server.file_put_cmds**: check the formatting of the given commands
- string **ftp_server.ftp_cmds**: specify additional commands supported by server beyond RFC 959
- bool **ftp_server.ignore_data_chan** = false: do not inspect FTP data channels
- bool **ftp_server.ignore_telnet_erase_cmds** = false: ignore erase character and erase line commands when normalizing
- string **ftp_server.login_cmds**: check the formatting of the given commands
- bool **ftp_server.print_cmds** = false: print command configurations on start up
- bool **ftp_server.telnet_cmds** = false: detect Telnet escape sequences of FTP control channel
- int **gid.~**: generator id { 1: }
- string **gtp_info.~**: info element to match
- int **gtp_inspect[].infos[].length** = 0: information element type code { 0:255 }
- string **gtp_inspect[].infos[].name**: information element name
- int **gtp_inspect[].infos[].type** = 0: information element type code { 0:255 }
- string **gtp_inspect[].messages[].name**: message name
- int **gtp_inspect[].messages[].type** = 0: message type code { 0:255 }
- int **gtp_inspect.trace**: mask for enabling debug traces in module
- int **gtp_inspect[].version** = 2: GTP version { 0:2 }

- string **gtp_type.~**: list of types to match
- int **gtp_version.~**: version to match { 0:2 }
- bool **high_availability.daq_channel** = false: enable use of daq data plane channel
- bool **high_availability.enable** = false: enable high availability
- real **high_availability.min_age** = 1.0: minimum session life before HA updates { 0.0:100.0 }
- real **high_availability.min_sync** = 1.0: minimum interval between HA updates { 0.0:100.0 }
- bit_list **high_availability.ports**: side channel message port list { 65535 }
- int **host_cache[].size**: size of host cache
- enum **hosts[].frag_policy**: defragmentation policy { first | linux | bsd | bsd_right | last | windows | solaris }
- addr **hosts[].ip** = 0.0.0.0/32: hosts address / CIDR
- string **hosts[].services[].name**: service identifier
- port **hosts[].services[].port**: port number
- enum **hosts[].services[].proto** = tcp: IP protocol { tcp | udp }
- enum **hosts[].tcp_policy**: TCP reassembly policy { first | last | linux | old_linux | bsd | macos | solaris | irix | hpux11 | hpux10 | windows | win_2003 | vista | proxy }
- enum **host_tracker[].frag_policy**: defragmentation policy { first | linux | bsd | bsd_right | last | windows | solaris }
- addr **host_tracker[].IP** = 0.0.0.0/32: hosts address / cidr
- string **host_tracker[].services[].name**: service identifier
- port **host_tracker[].services[].port**: port number
- enum **host_tracker[].services[].proto** = tcp: IP protocol { tcp | udp }
- enum **host_tracker[].tcp_policy**: TCP reassembly policy { first | last | linux | old_linux | bsd | macos | solaris | irix | hpux11 | hpux10 | windows | win_2003 | vista | proxy }
- implied **http_cookie.request**: match against the cookie from the request message even when examining the response
- implied **http_cookie.with_body**: parts of this rule examine HTTP message body
- implied **http_cookie.with_trailer**: parts of this rule examine HTTP message trailers
- string **http_header.field**: restrict to given header. Header name is case insensitive.
- implied **http_header.request**: match against the headers from the request message even when examining the response
- implied **http_header.with_body**: parts of this rule examine HTTP message body
- implied **http_header.with_trailer**: parts of this rule examine HTTP message trailers
- bool **http_inspect.backslash_to_slash** = false: replace \ with / when normalizing URIs
- bit_list **http_inspect.bad_characters**: alert when any of specified bytes are present in URI after percent decoding { 255 }
- bool **http_inspect.decompress_pdf** = false: decompress pdf files in response bodies
- bool **http_inspect.decompress_swf** = false: decompress swf files in response bodies
- string **http_inspect.ignore_unreserved**: do not alert when the specified unreserved characters are percent-encoded in a URI.Unreserved characters are 0-9, a-z, A-Z, period, underscore, tilde, and minus. { (optional) }
- bool **http_inspect.iis_double_decode** = false: perform double decoding of percent encodings to normalize characters

- int **http_inspect.iis_unicode_code_page** = 1252: code page to use from the IIS unicode map file { 0:65535 }
- bool **http_inspect.iis_unicode** = false: use IIS unicode code point mapping to normalize characters
- string **http_inspect.iis_unicode_map_file**: file containing code points for IIS unicode. { (optional) }
- int **http_inspect.max_javascript_whitespaces** = 200: maximum consecutive whitespaces allowed within the Javascript obfuscated data { 1:65535 }
- bool **http_inspect.normalize_javascript** = false: normalize javascript in response bodies
- bool **http_inspect.normalize_utf** = true: normalize charset utf encodings in response bodies
- int **http_inspect.oversize_dir_length** = 300: maximum length for URL directory { 1:65535 }
- bool **http_inspect.percent_u** = false: normalize %uNNNN and %UNNNN encodings
- bool **http_inspect.plus_to_space** = true: replace + with <sp> when normalizing URIs
- int **http_inspect.print_amount** = 1200: number of characters to print from a Field { 1:1000000 }
- bool **http_inspect.print_hex** = false: nonprinting characters printed in [HH] format instead of using an asterisk
- int **http_inspect.request_depth** = -1: maximum request message body bytes to examine (-1 no limit) { -1: }
- int **http_inspect.response_depth** = -1: maximum response message body bytes to examine (-1 no limit) { -1: }
- bool **http_inspect.show_pegs** = true: display peg counts with test output
- bool **http_inspect.show_scan** = false: display scanned segments
- bool **http_inspect.simplify_path** = true: reduce URI directory path to simplest form
- bool **http_inspect.test_input** = false: read HTTP messages from text file
- bool **http_inspect.test_output** = false: print out HTTP section data
- bool **http_inspect.unzip** = true: decompress gzip and deflate message bodies
- bool **http_inspect.utf8_bare_byte** = false: when doing UTF-8 character normalization include bytes that were not percent encoded
- bool **http_inspect.utf8** = true: normalize 2-byte and 3-byte UTF-8 characters to a single byte
- implied **http_method.with_body**: parts of this rule examine HTTP message body
- implied **http_method.with_trailer**: parts of this rule examine HTTP message trailers
- implied **http_raw_cookie.request**: match against the cookie from the request message even when examining the response
- implied **http_raw_cookie.with_body**: parts of this rule examine HTTP message body
- implied **http_raw_cookie.with_trailer**: parts of this rule examine HTTP message trailers
- implied **http_raw_header.request**: match against the headers from the request message even when examining the response
- implied **http_raw_header.with_body**: parts of this rule examine HTTP message body
- implied **http_raw_header.with_trailer**: parts of this rule examine HTTP message trailers
- implied **http_raw_request.with_body**: parts of this rule examine HTTP message body
- implied **http_raw_request.with_trailer**: parts of this rule examine HTTP message trailers
- implied **http_raw_status.with_body**: parts of this rule examine HTTP message body
- implied **http_raw_status.with_trailer**: parts of this rule examine HTTP message trailers
- implied **http_raw_trailer.request**: match against the trailers from the request message even when examining the response

- implied **http_raw_trailer.with_body**: parts of this rule examine HTTP response message body (must be combined with request)
- implied **http_raw_trailer.with_header**: parts of this rule examine HTTP response message headers (must be combined with request)
- implied **http_raw_uri.fragment**: match against fragment section of URI only
- implied **http_raw_uri.host**: match against host section of URI only
- implied **http_raw_uri.path**: match against path section of URI only
- implied **http_raw_uri.port**: match against port section of URI only
- implied **http_raw_uri.query**: match against query section of URI only
- implied **http_raw_uri.scheme**: match against scheme section of URI only
- implied **http_raw_uri.with_body**: parts of this rule examine HTTP message body
- implied **http_raw_uri.with_trailer**: parts of this rule examine HTTP message trailers
- implied **http_stat_code.with_body**: parts of this rule examine HTTP message body
- implied **http_stat_code.with_trailer**: parts of this rule examine HTTP message trailers
- implied **http_stat_msg.with_body**: parts of this rule examine HTTP message body
- implied **http_stat_msg.with_trailer**: parts of this rule examine HTTP message trailers
- string **http_trailer.field**: restrict to given trailer
- implied **http_trailer.request**: match against the trailers from the request message even when examining the response
- implied **http_trailer.with_body**: parts of this rule examine HTTP message body (must be combined with request)
- implied **http_trailer.with_header**: parts of this rule examine HTTP response message headers (must be combined with request)
- implied **http_true_ip.with_body**: parts of this rule examine HTTP message body
- implied **http_true_ip.with_trailer**: parts of this rule examine HTTP message trailers
- implied **http_uri.fragment**: match against fragment section of URI only
- implied **http_uri.host**: match against host section of URI only
- implied **http_uri.path**: match against path section of URI only
- implied **http_uri.port**: match against port section of URI only
- implied **http_uri.query**: match against query section of URI only
- implied **http_uri.scheme**: match against scheme section of URI only
- implied **http_uri.with_body**: parts of this rule examine HTTP message body
- implied **http_uri.with_trailer**: parts of this rule examine HTTP message trailers
- implied **http_version.request**: match against the version from the request message even when examining the response
- implied **http_version.with_body**: parts of this rule examine HTTP message body
- implied **http_version.with_trailer**: parts of this rule examine HTTP message trailers
- interval **icmp_id.~range**: check if ICMP ID is in given range { 0:65535 }
- interval **icmp_seq.~range**: check if ICMP sequence number is in given range { 0:65535 }

- interval **icode.~range**: check if ICMP code is in given range is { 0:255 }
- interval **id.~range**: check if the IP ID is in the given range { 0: }
- int **imap.b64_decode_depth** = 1460: base64 decoding depth (-1 no limit) { -1:65535 }
- int **imap.bitenc_decode_depth** = 1460: non-Encoded MIME attachment extraction depth (-1 no limit) { -1:65535 }
- int **imap.qp_decode_depth** = 1460: quoted Printable decoding depth (-1 no limit) { -1:65535 }
- int **imap.uu_decode_depth** = 1460: Unix-to-Unix decoding depth (-1 no limit) { -1:65535 }
- int **inspection.id** = 0: correlate policy and events with other items in configuration { 0:65535 }
- enum **inspection.mode** = inline-test: set policy mode { inline | inline-test }
- string **inspection.uuid**: correlate events by uuid
- select **ipopts.~opt**: output format { rr|eol|nop|ts|sec|esec|lsrr|lsrre|ssrr|satid|any }
- string **ip_proto.~proto**: [!|>|<] name or number
- bool **ips.enable_builtin_rules** = false: enable events from builtin rules w/o stubs
- int **ips.id** = 0: correlate unified2 events with configuration { 0:65535 }
- string **ips.include**: legacy snort rules and includes
- enum **ips.mode**: set policy mode { tap | inline | inline-test }
- string **ips.rules**: snort rules and includes
- string **ips.uuid** = 00000000-0000-0000-0000-000000000000: IPS policy uuid
- string **isdataat.~length**: num | !num
- implied **isdataat.relative**: offset from cursor instead of start of buffer
- interval **itype.~range**: check if ICMP type is in given range { 0:255 }
- enum **latency.packet.action** = none: event action if packet times out and is fastpathed { none | alert | log | alert_and_log }
- bool **latency.packet.fastpath** = false: fastpath expensive packets (max_time exceeded)
- int **latency.packet.max_time** = 500: set timeout for packet latency thresholding (usec) { 0: }
- enum **latency.rule.action** = none: event action for rule latency enable and suspend events { none | alert | log | alert_and_log }
- int **latency.rule.max_suspend_time** = 30000: set max time for suspending a rule (ms, 0 means permanently disable rule) { 0: }
- int **latency.rule.max_time** = 500: set timeout for rule evaluation (usec) { 0: }
- bool **latency.rule.suspend** = false: temporarily suspend expensive rules
- int **latency.rule.suspend_threshold** = 5: set threshold for number of timeouts before suspending a rule { 1: }
- bool **log_codecs.file** = false: output to log_codecs.txt instead of stdout
- bool **log_codecs.msg** = false: include alert msg
- bool **log_hext.file** = false: output to log_hext.txt instead of stdout
- int **log_hext.limit** = 0: set maximum size in MB before rollover (0 is unlimited) { 0: }
- bool **log_hext.raw** = false: output all full packets if true, else just TCP payload
- int **log_hext.width** = 20: set line width (0 is unlimited) { 0: }

- int **log_pcap.limit** = 0: set maximum size in MB before rollover (0 is unlimited) { 0: }
- string **md5.~hash**: data to match
- int **md5.length**: number of octets in plain text { 1:65535 }
- string **md5.offset**: var or number of bytes from start of buffer to start search
- implied **md5.relative** = false: offset from cursor instead of start of buffer
- int **memory.cap** = 0: set the per-packet-thread cap on memory (bytes, 0 to disable) { 0: }
- bool **memory.soft** = false: always succeed in allocating memory, even if above the cap
- int **memory.threshold** = 0: set the per-packet-thread threshold for preemptive cleanup actions (percent, 0 to disable) { 0: }
- string **metadata.***: comma-separated list of arbitrary name value pairs
- string **modbus_func.~**: function code to match
- int **modbus_unit.~**: Modbus unit ID { 0:255 }
- bool **mpls.enable_mpls_multicast** = false: enables support for MPLS multicast
- bool **mpls.enable_mpls_overlapping_ip** = false: enable if private network addresses overlap and must be differentiated by MPLS label(s)
- int **mpls.max_mpls_stack_depth** = -1: set MPLS stack depth { -1: }
- enum **mpls.mpls_payload_type** = ip4: set encapsulated payload type { eth | ip4 | ip6 }
- string **msg.~**: message describing rule
- interval **mss.~range**: check if TCP MSS is in given range { 0:65535 }
- multi **network.checksum_drop** = none: drop if checksum is bad { all | ip | noip | tcp | notcp | udp | noudp | icmp | noicmp | none }
- multi **network.checksum_eval** = none: checksums to verify { all | ip | noip | tcp | notcp | udp | noudp | icmp | noicmp | none }
- bool **network.decode_drops** = false: enable dropping of packets by the decoder
- int **network.id** = 0: correlate unified2 events with configuration { 0:65535 }
- int **network.layers** = 40: the maximum number of protocols that Snort can correctly decode { 3:255 }
- int **network.max_ip6_extensions** = 0: the maximum number of IP6 options Snort will process for a given IPv6 layer before raising 116:456 (0 = unlimited) { 0:255 }
- int **network.max_ip_layers** = 0: the maximum number of IP layers Snort will process for a given packet before raising 116:293 (0 = unlimited) { 0:255 }
- int **network.min_ttl** = 1: alert / normalize packets with lower TTL / hop limit (you must enable rules and / or normalization also) { 1:255 }
- int **network.new_ttl** = 1: use this value for responses and when normalizing { 1:255 }
- bool **normalizer.icmp4** = false: clear reserved flag
- bool **normalizer.icmp6** = false: clear reserved flag
- bool **normalizer.ip4.base** = true: clear options
- bool **normalizer.ip4.df** = false: clear don't frag flag
- bool **normalizer.ip4.rf** = false: clear reserved flag
- bool **normalizer.ip4.tos** = false: clear tos / differentiated services byte

- bool **normalizer.ip4.trim** = false: truncate excess payload beyond datagram length
- bool **normalizer.ip6** = false: clear reserved flag
- string **normalizer.tcp.allow_codes**: don't clear given option codes
- multi **normalizer.tcp.allow_names**: don't clear given option names { sack | echo | partial_order | conn_count | alt_checksum | md5 }
- bool **normalizer.tcp.base** = true: clear reserved bits and option padding and fix urgent pointer / flags issues
- bool **normalizer.tcp.block** = true: allow packet drops during TCP normalization
- select **normalizer.tcp.ecn** = off: clear ecn for all packets | sessions w/o ecn setup { off | packet | stream }
- bool **normalizer.tcp.ips** = false: ensure consistency in retransmitted data
- bool **normalizer.tcp.opts** = true: clear all options except mss, wscale, timestamp, and any explicitly allowed
- bool **normalizer.tcp.pad** = true: clear any option padding bytes
- bool **normalizer.tcp.req_pay** = true: clear the urgent pointer and the urgent flag if there is no payload
- bool **normalizer.tcp.req_urg** = true: clear the urgent pointer if the urgent flag is not set
- bool **normalizer.tcp.req_urp** = true: clear the urgent flag if the urgent pointer is not set
- bool **normalizer.tcp.rsv** = true: clear the reserved bits in the TCP header
- bool **normalizer.tcp.trim** = false: enable all of the TCP trim options
- bool **normalizer.tcp.trim_mss** = false: trim data to MSS
- bool **normalizer.tcp.trim_rst** = false: remove any data from RST packet
- bool **normalizer.tcp.trim_syn** = false: remove data on SYN
- bool **normalizer.tcp.trim_win** = false: trim data to window
- bool **normalizer.tcp.urp** = true: adjust urgent pointer if beyond segment length
- bool **output.dump_chars_only** = false: turns on character dumps (same as -C)
- bool **output.dump_payload** = false: dumps application layer (same as -d)
- bool **output.dump_payload_verbose** = false: dumps raw packet starting at link layer (same as -X)
- int **output.event_trace.max_data** = 0: maximum amount of packet data to capture { 0:65535 }
- string **output.logdir** = .: where to put log files (same as -l)
- bool **output.obfuscate** = false: obfuscate the logged IP addresses (same as -O)
- bool **output.obfuscate_pii** = false: mask all but the last 4 characters of credit card and social security numbers
- bool **output.quiet** = false: suppress non-fatal information (still show alerts, same as -q)
- bool **output.show_year** = false: include year in timestamp in the alert and log files (same as -y)
- int **output.tagged_packet_limit** = 256: maximum number of packets tagged for non-packet metrics { 0: }
- bool **output.verbose** = false: be verbose (same as -v)
- bool **output.wide_hex_dump** = true: output 20 bytes per lines instead of 16 when dumping buffers
- bool **packet_capture.enable** = false: initially enable packet dumping
- string **packet_capture.filter**: bpf filter to use for packet dump

- bool **packets.address_space_agnostic** = false: determines whether DAQ address space info is used to track fragments and connections
- string **packets.bpf_file**: file with BPF to select traffic for Snort
- int **packets.limit** = 0: maximum number of packets to process before stopping (0 is unlimited) { 0: }
- int **packets.skip** = 0: number of packets to skip before before processing { 0: }
- bool **packets.vlan_agnostic** = false: determines whether VLAN info is used to track fragments and connections
- bool **packet_tracer.enable** = false: enable summary output of state that determined packet verdict
- enum **packet_tracer.output** = console: select where to send packet trace { console | file }
- string **pcre.~re**: Snort regular expression
- bool **perf_monitor.base** = true: enable base statistics { nullptr }
- bool **perf_monitor.cpu** = false: enable cpu statistics { nullptr }
- bool **perf_monitor.flow** = false: enable traffic statistics
- bool **perf_monitor.flow_ip** = false: enable statistics on host pairs
- int **perf_monitor.flow_ip_memcap** = 52428800: maximum memory in bytes for flow tracking { 8200: }
- int **perf_monitor.flow_ports** = 1023: maximum ports to track { 0:65535 }
- enum **perf_monitor.format** = csv: output format for stats { csv | text | json | flatbuffers }
- int **perf_monitor.max_file_size** = 1073741824: files will be rolled over if they exceed this size { 4096: }
- string **perf_monitor.modules[].name**: name of the module
- string **perf_monitor.modules[].pegs**: list of statistics to track or empty for all counters
- enum **perf_monitor.output** = file: output location for stats { file | console }
- int **perf_monitor.packets** = 10000: minimum packets to report { 0: }
- int **perf_monitor.seconds** = 60: report interval { 1: }
- bool **perf_monitor.summary** = false: output summary at shutdown
- interval **pkt_num.~range**: check if packet number is in given range { 1: }
- int **pop.b64_decode_depth** = 1460: base64 decoding depth (-1 no limit) { -1:65535 }
- int **pop.bitenc_decode_depth** = 1460: Non-Encoded MIME attachment extraction depth (-1 no limit) { -1:65535 }
- int **pop.qp_decode_depth** = 1460: Quoted Printable decoding depth (-1 no limit) { -1:65535 }
- int **pop.uu_decode_depth** = 1460: Unix-to-Unix decoding depth (-1 no limit) { -1:65535 }
- bool **port_scan.alert_all** = false: alert on all events over threshold within window if true; else alert on first only
- int **port_scan.icmp_sweep.nets** = 25: number of times address changed from prior attempt { 0: }
- int **port_scan.icmp_sweep.ports** = 25: number of times port (or proto) changed from prior attempt { 0: }
- int **port_scan.icmp_sweep.rejects** = 15: scan attempts with negative response { 0: }
- int **port_scan.icmp_sweep.scans** = 100: scan attempts { 0: }
- int **port_scan.icmp_window** = 0: detection interval for all ICMP scans { 0: }
- string **port_scan.ignore_scanned**: list of CIDRs with optional ports to ignore if the destination of scan alerts

- string **port_scan.ignore_scanners**: list of CIDRs with optional ports to ignore if the source of scan alerts
- bool **port_scan.include_midstream** = false: list of CIDRs with optional ports
- int **port_scan.ip_decoy.nets** = 25: number of times address changed from prior attempt { 0: }
- int **port_scan.ip_decoy.ports** = 25: number of times port (or proto) changed from prior attempt { 0: }
- int **port_scan.ip_decoy.rejects** = 15: scan attempts with negative response { 0: }
- int **port_scan.ip_decoy.scans** = 100: scan attempts { 0: }
- int **port_scan.ip_dist.nets** = 25: number of times address changed from prior attempt { 0: }
- int **port_scan.ip_dist.ports** = 25: number of times port (or proto) changed from prior attempt { 0: }
- int **port_scan.ip_dist.rejects** = 15: scan attempts with negative response { 0: }
- int **port_scan.ip_dist.scans** = 100: scan attempts { 0: }
- int **port_scan.ip_proto.nets** = 25: number of times address changed from prior attempt { 0: }
- int **port_scan.ip_proto.ports** = 25: number of times port (or proto) changed from prior attempt { 0: }
- int **port_scan.ip_proto.rejects** = 15: scan attempts with negative response { 0: }
- int **port_scan.ip_proto.scans** = 100: scan attempts { 0: }
- int **port_scan.ip_sweep.nets** = 25: number of times address changed from prior attempt { 0: }
- int **port_scan.ip_sweep.ports** = 25: number of times port (or proto) changed from prior attempt { 0: }
- int **port_scan.ip_sweep.rejects** = 15: scan attempts with negative response { 0: }
- int **port_scan.ip_sweep.scans** = 100: scan attempts { 0: }
- int **port_scan.ip_window** = 0: detection interval for all IP scans { 0: }
- int **port_scan.memcap** = 1048576: maximum tracker memory in bytes { 1: }
- multi **port_scan.protos** = all: choose the protocols to monitor { tcp | udp | icmp | ip | all }
- multi **port_scan.scan_types** = all: choose type of scans to look for { portscan | portsweep | decoy_portscan | distributed_portscan | all }
- int **port_scan.tcp_decoy.nets** = 25: number of times address changed from prior attempt { 0: }
- int **port_scan.tcp_decoy.ports** = 25: number of times port (or proto) changed from prior attempt { 0: }
- int **port_scan.tcp_decoy.rejects** = 15: scan attempts with negative response { 0: }
- int **port_scan.tcp_decoy.scans** = 100: scan attempts { 0: }
- int **port_scan.tcp_dist.nets** = 25: number of times address changed from prior attempt { 0: }
- int **port_scan.tcp_dist.ports** = 25: number of times port (or proto) changed from prior attempt { 0: }
- int **port_scan.tcp_dist.rejects** = 15: scan attempts with negative response { 0: }
- int **port_scan.tcp_dist.scans** = 100: scan attempts { 0: }
- int **port_scan.tcp_ports.nets** = 25: number of times address changed from prior attempt { 0: }
- int **port_scan.tcp_ports.ports** = 25: number of times port (or proto) changed from prior attempt { 0: }
- int **port_scan.tcp_ports.rejects** = 15: scan attempts with negative response { 0: }
- int **port_scan.tcp_ports.scans** = 100: scan attempts { 0: }

- int **port_scan.tcp_sweep.nets** = 25: number of times address changed from prior attempt { 0: }
- int **port_scan.tcp_sweep.ports** = 25: number of times port (or proto) changed from prior attempt { 0: }
- int **port_scan.tcp_sweep.rejects** = 15: scan attempts with negative response { 0: }
- int **port_scan.tcp_sweep.scans** = 100: scan attempts { 0: }
- int **port_scan.tcp_window** = 0: detection interval for all TCP scans { 0: }
- int **port_scan.udp_decoy.nets** = 25: number of times address changed from prior attempt { 0: }
- int **port_scan.udp_decoy.ports** = 25: number of times port (or proto) changed from prior attempt { 0: }
- int **port_scan.udp_decoy.rejects** = 15: scan attempts with negative response { 0: }
- int **port_scan.udp_decoy.scans** = 100: scan attempts { 0: }
- int **port_scan.udp_dist.nets** = 25: number of times address changed from prior attempt { 0: }
- int **port_scan.udp_dist.ports** = 25: number of times port (or proto) changed from prior attempt { 0: }
- int **port_scan.udp_dist.rejects** = 15: scan attempts with negative response { 0: }
- int **port_scan.udp_dist.scans** = 100: scan attempts { 0: }
- int **port_scan.udp_ports.nets** = 25: number of times address changed from prior attempt { 0: }
- int **port_scan.udp_ports.ports** = 25: number of times port (or proto) changed from prior attempt { 0: }
- int **port_scan.udp_ports.rejects** = 15: scan attempts with negative response { 0: }
- int **port_scan.udp_ports.scans** = 100: scan attempts { 0: }
- int **port_scan.udp_sweep.nets** = 25: number of times address changed from prior attempt { 0: }
- int **port_scan.udp_sweep.ports** = 25: number of times port (or proto) changed from prior attempt { 0: }
- int **port_scan.udp_sweep.rejects** = 15: scan attempts with negative response { 0: }
- int **port_scan.udp_sweep.scans** = 100: scan attempts { 0: }
- int **port_scan.udp_window** = 0: detection interval for all UDP scans { 0: }
- string **port_scan.watch_ip**: list of CIDRs with optional ports to watch
- int **priority.~**: relative severity level; 1 is highest priority { 1: }
- string **process.chroot**: set chroot directory (same as -t)
- bool **process.daemon** = false: fork as a daemon (same as -D)
- bool **process.dirty_pig** = false: shutdown without internal cleanup
- string **process.set_gid**: set group ID (same as -g)
- string **process.set_uid**: set user ID (same as -u)
- string **process.threads[].cpuset**: pin the associated thread to this cpuset
- int **process.threads[].thread** = 0: set cpu affinity for the <cur_thread_num> thread that runs { 0: }
- string **process.umask**: set process umask (same as -m)
- bool **process.utc** = false: use UTC instead of local time for timestamps
- int **profiler.memory.count** = 0: limit results to count items per level (0 = no limit) { 0: }
- int **profiler.memory.max_depth** = -1: limit depth to max_depth (-1 = no limit) { -1: }

- bool **profiler.memory.show** = true: show module memory profile stats
- enum **profiler.memory.sort** = total_used: sort by given field { none | allocations | total_used | avg_allocation }
- int **profiler.modules.count** = 0: limit results to count items per level (0 = no limit) { 0: }
- int **profiler.modules.max_depth** = -1: limit depth to max_depth (-1 = no limit) { -1: }
- bool **profiler.modules.show** = true: show module time profile stats
- enum **profiler.modules.sort** = total_time: sort by given field { none | checks | avg_check | total_time }
- int **profiler.rules.count** = 0: print results to given level (0 = all) { 0: }
- bool **profiler.rules.show** = true: show rule time profile stats
- enum **profiler.rules.sort** = total_time: sort by given field { none | checks | avg_check | total_time | matches | no_matches | avg_match | avg_no_match }
- string **rate_filter[].apply_to**: restrict filter to these addresses according to track
- int **rate_filter[].count** = 1: number of events in interval before tripping { 0: }
- int **rate_filter[].gid** = 1: rule generator ID { 0: }
- enum **rate_filter[].new_action** = alert: take this action on future hits until timeout { log | pass | alert | drop | block | reset }
- int **rate_filter[].seconds** = 1: count interval { 0: }
- int **rate_filter[].sid** = 1: rule signature ID { 0: }
- int **rate_filter[].timeout** = 1: count interval { 0: }
- enum **rate_filter[].track** = by_src: filter only matching source or destination addresses { by_src | by_dst | by_rule }
- bool **react.msg** = false: use rule msg in response page instead of default message
- string **react.page**: file containing HTTP response (headers and body)
- string **reference.~id**: reference id
- string **reference.~scheme**: reference scheme
- string **references[].name**: name used with reference rule option
- string **references[].url**: where this reference is defined
- implied **regex.dotall**: matching a . will not exclude newlines
- implied **regex.fast_pattern**: use this content in the fast pattern matcher instead of the content selected by default
- implied **regex.multiline**: ^ and $ anchors match any newlines in data
- implied **regex.nocase**: case insensitive match
- string **regex.~re**: hyperscan regular expression
- implied **regex.relative**: start search from end of last match instead of start of buffer
- bool **reg_test.test_daq_retry** = true: test daq packet retry feature
- enum **reject.control**: send ICMP unreachable(s) { network|host|port|all }
- enum **reject.reset**: send TCP reset to one or both ends { source|dest|both }
- string **rem.~**: comment
- string **replace.~**: byte code to replace with

- string **reputation.blacklist**: blacklist file name with IP lists
- string **reputation.list_dir**: directory for IP lists and manifest file
- int **reputation.memcap** = 500: maximum total MB of memory allocated { 1:4095 }
- enum **reputation.nested_ip** = inner: IP to use when there is IP encapsulation { inner|outer|all }
- enum **reputation.priority** = whitelist: defines priority when there is a decision conflict during run-time { blacklist|whitelist }
- bool **reputation.scan_local** = false: inspect local address defined in RFC 1918
- string **reputation.whitelist**: whitelist file name with IP lists
- enum **reputation.white** = unblack: specify the meaning of whitelist { unblack|trust }
- int **rev.~**: revision { 1: }
- bool **rewrite.disable_replace** = false: disable replace of packet contents with rewrite rules
- int **rpc.~app**: application number
- string **rpc.~proc**: procedure number or * for any
- string **rpc.~ver**: version number or * for any
- bool **rule_state.enable** = true: enable or disable rule in all policies
- int **rule_state.gid** = 0: rule generator ID { 0: }
- int **rule_state.sid** = 0: rule signature ID { 0: }
- string **sd_pattern.~pattern**: The pattern to search for
- int **sd_pattern.threshold**: number of matches before alerting { 1 }
- int **search_engine.bleedover_port_limit** = 1024: maximum ports in rule before demotion to any-any port group { 1: }
- bool **search_engine.bleedover_warnings_enabled** = false: print warning if a rule is demoted to any-any port group
- bool **search_engine.debug** = false: print verbose fast pattern info
- bool **search_engine.debug_print_nocontent_rule_tests** = false: print rule group info during packet evaluation
- bool **search_engine.debug_print_rule_group_build_details** = false: print rule group info during compilation
- bool **search_engine.debug_print_rule_groups_compiled** = false: prints compiled rule group information
- bool **search_engine.debug_print_rule_groups_uncompiled** = false: prints uncompiled rule group information
- bool **search_engine.detect_raw_tcp** = false: detect on TCP payload before reassembly
- bool **search_engine.enable_single_rule_group** = false: put all rules into one group
- int **search_engine.max_pattern_len** = 0: truncate patterns when compiling into state machine (0 means no maximum) { 0: }
- int **search_engine.max_queue_events** = 5: maximum number of matching fast pattern states to queue per packet { 2:100 }
- dynamic **search_engine.search_method** = ac_bnfa: set fast pattern algorithm - choose available search engine { ac_banded | ac_bnfa | ac_full | ac_sparse | ac_sparse_bands | ac_std | hyperscan | lowmem }
- bool **search_engine.search_optimize** = true: tweak state machine construction for better performance
- bool **search_engine.show_fast_patterns** = false: print fast pattern info for each rule
- bool **search_engine.split_any_any** = true: evaluate any-any rules separately to save memory
- interval **seq.~range**: check if TCP sequence number is in given range { 0: }

- string **service.***: one or more comma-separated service names
- enum **session.~mode**: output format { printable|binary|all }
- string **sha256.~hash**: data to match
- int **sha256.length**: number of octets in plain text { 1:65535 }
- string **sha256.offset**: var or number of bytes from start of buffer to start search
- implied **sha256.relative** = false: offset from cursor instead of start of buffer
- string **sha512.~hash**: data to match
- int **sha512.length**: number of octets in plain text { 1:65535 }
- string **sha512.offset**: var or number of bytes from start of buffer to start search
- implied **sha512.relative** = false: offset from cursor instead of start of buffer
- string **side_channel.connector**: connector handle
- string **side_channel.connectors[].connector**: connector handle
- bit_list **side_channel.ports**: side channel message port list { 65535 }
- int **sid.~**: signature id { 1: }
- bool **sip.ignore_call_channel** = false: enables the support for ignoring audio/video data channel
- int **sip.max_call_id_len** = 256: maximum call id field size { 0:65535 }
- int **sip.max_contact_len** = 256: maximum contact field size { 0:65535 }
- int **sip.max_content_len** = 1024: maximum content length of the message body { 0:65535 }
- int **sip.max_dialogs** = 4: maximum number of dialogs within one stream session { 1:4194303 }
- int **sip.max_from_len** = 256: maximum from field size { 0:65535 }
- int **sip.max_requestName_len** = 20: maximum request name field size { 0:65535 }
- int **sip.max_to_len** = 256: maximum to field size { 0:65535 }
- int **sip.max_uri_len** = 256: maximum request uri field size { 0:65535 }
- int **sip.max_via_len** = 1024: maximum via field size { 0:65535 }
- string **sip_method.*method**: sip method
- string **sip.methods** = invite cancel ack bye register options: list of methods to check in SIP messages
- int **sip_stat_code.*code**: stat code { 1:999 }
- string **smtp.alt_max_command_line_len[].command**: command string
- int **smtp.alt_max_command_line_len[].length** = 0: specify non-default maximum for command { 0: }
- string **smtp.auth_cmds**: commands that initiate an authentication exchange
- int **smtp.b64_decode_depth** = 1460: depth used to decode the base64 encoded MIME attachments (-1 no limit) { -1:65535 }
- string **smtp.binary_data_cmds**: commands that initiate sending of data and use a length value after the command
- int **smtp.bitenc_decode_depth** = 1460: depth used to extract the non-encoded MIME attachments (-1 no limit) { -1:65535 }
- string **smtp.data_cmds**: commands that initiate sending of data with an end of data delimiter
- int **smtp.email_hdrs_log_depth** = 1464: depth for logging email headers { 0:20480 }

- bool **smtp.ignore_data** = false: ignore data section of mail
- bool **smtp.ignore_tls_data** = false: ignore TLS-encrypted data when processing rules
- string **smtp.invalid_cmds**: alert if this command is sent from client side
- bool **smtp.log_email_hdrs** = false: log the SMTP email headers extracted from SMTP data
- bool **smtp.log_filename** = false: log the MIME attachment filenames extracted from the Content-Disposition header within the MIME body
- bool **smtp.log_mailfrom** = false: log the sender's email address extracted from the MAIL FROM command
- bool **smtp.log_rcptto** = false: log the recipient's email address extracted from the RCPT TO command
- int **smtp.max_auth_command_line_len** = 1000: max auth command Line Length { 0:65535 }
- int **smtp.max_command_line_len** = 0: max Command Line Length { 0:65535 }
- int **smtp.max_header_line_len** = 0: max SMTP DATA header line { 0:65535 }
- int **smtp.max_response_line_len** = 0: max SMTP response line { 0:65535 }
- string **smtp.normalize_cmds**: list of commands to normalize
- enum **smtp.normalize** = none: turns on/off normalization { none | cmds | all }
- int **smtp.qp_decode_depth** = 1460: quoted-Printable decoding depth (-1 no limit) { -1:65535 }
- int **smtp.uu_decode_depth** = 1460: Unix-to-Unix decoding depth (-1 no limit) { -1:65535 }
- string **smtp.valid_cmds**: list of valid commands
- enum **smtp.xlink2state** = alert: enable/disable xlink2state alert { disable | alert | drop }
- implied **snort.--alert-before-pass**: process alert, drop, sdrop, or reject before pass; default is pass before alert, drop,. . .
- string **snort.-A**: <mode> set alert mode: none, cmg, or alert_*
- addr **snort.-B** = 255.255.255.255/32: <mask> obfuscated IP addresses in alerts and packet dumps using CIDR mask
- string **snort.--bpf**: <filter options> are standard BPF options, as seen in TCPDump
- string **snort.--c2x**: output hex for given char (see also --x2c)
- string **snort.--catch-test**: comma separated list of cat unit test tags or *all*
- string **snort.-c**: <conf> use this configuration
- string **snort.--control-socket**: <file> to create unix socket
- implied **snort.-C**: print out payloads with character data only (no hex)
- implied **snort.--create-pidfile**: create PID file, even when not in Daemon mode
- string **snort.--daq-dir**: <dir> tell snort where to find desired DAQ
- implied **snort.--daq-list**: list packet acquisition modules available in optional dir, default is static modules only
- string **snort.--daq**: <type> select packet acquisition module (default is pcap)
- string **snort.--daq-var**: <name=value> specify extra DAQ configuration variable
- implied **snort.-d**: dump the Application Layer
- implied **snort.--dirty-pig**: don't flush packets on shutdown
- implied **snort.-D**: run Snort in background (daemon) mode

- string **snort.--dump-builtin-rules**: [<module prefix>] output stub rules for selected modules { (optional) }
- string **snort.--dump-defaults**: [<module prefix>] output module defaults in Lua format { (optional) }
- implied **snort.--dump-dynamic-rules**: output stub rules for all loaded rules libraries
- implied **snort.--dump-version**: output the version, the whole version, and only the version
- implied **snort.-e**: display the second layer header info
- implied **snort.--enable-inline-test**: enable Inline-Test Mode Operation
- implied **snort.-f**: turn off fflush() calls after binary log writes
- int **snort.-G**: <0xid> (same as --logid) { 0:65535 }
- implied **snort.--gen-msg-map**: dump builtin rules in gen-msg.map format for use by other tools
- string **snort.-g**: <gname> run snort gid as <gname> group (or gid) after initialization
- string **snort.--help-commands**: [<module prefix>] output matching commands { (optional) }
- string **snort.--help-config**: [<module prefix>] output matching config options { (optional) }
- string **snort.--help-counts**: [<module prefix>] output matching peg counts { (optional) }
- implied **snort.--help**: list command line options
- string **snort.--help-module**: <module> output description of given module
- implied **snort.--help-modules**: list all available modules with brief help
- string **snort.--help-options**: [<option prefix>] output matching command line option quick help (same as -?) { (optional) }
- implied **snort.--help-plugins**: list all available plugins with brief help
- implied **snort.--help-signals**: dump available control signals
- implied **snort.-H**: make hash tables deterministic
- int **snort.--id-offset** = 0: offset to add to instance IDs when logging to files { 0:65535 }
- implied **snort.--id-subdir**: create/use instance subdirectories in logdir instead of instance filename prefix
- implied **snort.--id-zero**: use id prefix / subdirectory even with one packet thread
- string **snort.-i**: <iface>. . . list of interfaces
- port **snort.-j**: <port> to listen for Telnet connections
- enum **snort.-k** = all: <mode> checksum mode; default is all { all|noip|notcp|noudp|noicmp|none }
- implied **snort.--list-buffers**: output available inspection buffers
- string **snort.--list-builtin**: [<module prefix>] output matching builtin rules { (optional) }
- string **snort.--list-gids**: [<module prefix>] output matching generators { (optional) }
- string **snort.--list-modules**: [<module type>] list all known modules of given type { (optional) }
- implied **snort.--list-plugins**: list all known plugins
- string **snort.-l**: <logdir> log to this directory instead of current directory
- string **snort.-L**: <mode> logging mode (none, dump, pcap, or log_*)
- int **snort.--logid**: <0xid> log Identifier to uniquely id events for multiple snorts (same as -G) { 0:65535 }
- string **snort.--lua**: <chunk> extend/override conf with chunk; may be repeated

- implied **snort.--markup**: output help in asciidoc compatible format
- int **snort.--max-packet-threads** = 1: <count> configure maximum number of packet threads (same as -z) { 0: }
- implied **snort.--mem-check**: like -T but also compile search engines
- implied **snort.-M**: log messages to syslog (not alerts)
- int **snort.-m**: <umask> set umask = <umask> { 0: }
- int **snort.-n**: <count> stop after count packets { 0: }
- implied **snort.--nolock-pidfile**: do not try to lock Snort PID file
- implied **snort.--nostamps**: don't include timestamps in log file names
- implied **snort.-O**: obfuscate the logged IP addresses
- string **snort.-?**: <option prefix> output matching command line option quick help (same as --help-options) { (optional) }
- implied **snort.--parsing-follows-files**: parse relative paths from the perspective of the current configuration file
- implied **snort.--pause**: wait for resume/quit command before processing packets/terminating
- string **snort.--pcap-dir**: <dir> a directory to recurse to look for pcaps - read mode is implied
- string **snort.--pcap-file**: <file> file that contains a list of pcaps to read - read mode is implied
- string **snort.--pcap-filter**: <filter> filter to apply when getting pcaps from file or directory
- string **snort.--pcap-list**: <list> a space separated list of pcaps to read - read mode is implied
- int **snort.--pcap-loop**: <count> read all pcaps <count> times; 0 will read until Snort is terminated { -1: }
- implied **snort.--pcap-no-filter**: reset to use no filter when getting pcaps from file or directory
- implied **snort.--pcap-reload**: if reading multiple pcaps, reload snort config between pcaps
- implied **snort.--pcap-show**: print a line saying what pcap is currently being read
- implied **snort.--pedantic**: warnings are fatal
- implied **snort.--piglet**: enable piglet test harness mode
- string **snort.--plugin-path**: <path> where to find plugins
- implied **snort.--process-all-events**: process all action groups
- implied **snort.-Q**: enable inline mode operation
- implied **snort.-q**: quiet mode - Don't show banner and status report
- string **snort.-r**: <pcap>... (same as --pcap-list)
- string **snort.-R**: <rules> include this rules file in the default policy
- string **snort.--rule**: <rules> to be added to configuration; may be repeated
- implied **snort.--rule-to-hex**: output so rule header to stdout for text rule on stdin
- string **snort.--rule-to-text** = [SnortFoo]: output plain so rule header to stdout for text rule on stdin { 16 }
- string **snort.--run-prefix**: <pfx> prepend this to each output file
- int **snort.-s** = 1514: <snap> (same as --snaplen); default is 1514 { 68:65535 }
- string **snort.--script-path**: <path> to a luajit script or directory containing luajit scripts
- implied **snort.--shell**: enable the interactive command line

- implied **snort.--show-plugins**: list module and plugin versions
- int **snort.--skip**: <n> skip 1st n packets { 0: }
- int **snort.--snaplen** = 1514: <snap> set snaplen of packet (same as -s) { 68:65535 }
- implied **snort.--stdin-rules**: read rules from stdin until EOF or a line starting with END is read
- string **snort.-S**: <x=v> set config variable x equal to value v
- implied **snort.--talos**: enable Talos inline rule test mode (same as --tweaks talos -Q -q)
- string **snort.-t**: <dir> chroots process to <dir> after initialization
- int **snort.trace**: mask for enabling debug traces in module
- implied **snort.--trace**: turn on main loop debug trace
- implied **snort.--treat-drop-as-alert**: converts drop, sdrop, and reject rules into alert rules during startup
- implied **snort.--treat-drop-as-ignore**: use drop, sdrop, and reject rules to ignore session traffic when not inline
- implied **snort.-T**: test and report on the current Snort configuration
- string **snort.--tweaks**: tune configuration
- string **snort.-u**: <uname> run snort as <uname> or <uid> after initialization
- implied **snort.-U**: use UTC for timestamps
- implied **snort.-v**: be verbose
- implied **snort.--version**: show version number (same as -V)
- implied **snort.-V**: (same as --version)
- implied **snort.--warn-all**: enable all warnings
- implied **snort.--warn-conf**: warn about configuration issues
- implied **snort.--warn-daq**: warn about DAQ issues, usually related to mode
- implied **snort.--warn-flowbits**: warn about flowbits that are checked but not set and vice-versa
- implied **snort.--warn-hosts**: warn about host table issues
- implied **snort.--warn-plugins**: warn about issues that prevent plugins from loading
- implied **snort.--warn-rules**: warn about duplicate rules and rule parsing issues
- implied **snort.--warn-scripts**: warn about issues discovered while processing Lua scripts
- implied **snort.--warn-symbols**: warn about unknown symbols in your Lua config
- implied **snort.--warn-vars**: warn about variable definition and usage issues
- implied **snort.-W**: lists available interfaces
- int **snort.--x2c**: output ASCII char for given hex (see also --c2x)
- string **snort.--x2s**: output ASCII string for given byte code (see also --x2c)
- implied **snort.-X**: dump the raw packet data starting at the link layer
- implied **snort.-x**: same as --pedantic
- implied **snort.-y**: include year in timestamp in the alert and log files

- int **snort.-z** = 1: <count> maximum number of packet threads (same as --max-packet-threads); 0 gets the number of CPU cores reported by the system; default is 1 { 0: }
- string **so.~func**: name of eval function
- string **soid.~**: SO rule ID is unique key, eg <gid>_<sid>_<rev> like 3_45678_9
- int **ssh.max_client_bytes** = 19600: number of unanswered bytes before alerting on challenge-response overflow or CRC32 { 0:65535 }
- int **ssh.max_encrypted_packets** = 25: ignore session after this many encrypted packets { 0:65535 }
- int **ssh.max_server_version_len** = 80: limit before alerting on secure CRT server version string overflow { 0:255 }
- int **ssl.max_heartbeat_length** = 0: maximum length of heartbeat record allowed { 0:65535 }
- implied **ssl_state.client_hello**: check for client hello
- implied **ssl_state.!client_hello**: check for records that are not client hello
- implied **ssl_state.client_keyx**: check for client keyx
- implied **ssl_state.!client_keyx**: check for records that are not client keyx
- implied **ssl_state.!server_hello**: check for records that are not server hello
- implied **ssl_state.server_hello**: check for server hello
- implied **ssl_state.!server_keyx**: check for records that are not server keyx
- implied **ssl_state.server_keyx**: check for server keyx
- implied **ssl_state.!unknown**: check for records that are not unknown
- implied **ssl_state.unknown**: check for unknown record
- bool **ssl.trust_servers** = false: disables requirement that application (encrypted) data must be observed on both sides
- implied **ssl_version.!sslv2**: check for records that are not sslv2
- implied **ssl_version.sslv2**: check for sslv2
- implied **ssl_version.!sslv3**: check for records that are not sslv3
- implied **ssl_version.sslv3**: check for sslv3
- implied **ssl_version.!tls1.0**: check for records that are not tls1.0
- implied **ssl_version.tls1.0**: check for tls1.0
- implied **ssl_version.!tls1.1**: check for records that are not tls1.1
- implied **ssl_version.tls1.1**: check for tls1.1
- implied **ssl_version.!tls1.2**: check for records that are not tls1.2
- implied **ssl_version.tls1.2**: check for tls1.2
- int **stream.file_cache.idle_timeout** = 180: maximum inactive time before retiring session tracker { 1: }
- int **stream.file_cache.max_sessions** = 128: maximum simultaneous sessions tracked before pruning { 2: }
- int **stream.file_cache.pruning_timeout** = 30: minimum inactive time before being eligible for pruning { 1: }
- bool **stream_file.upload** = false: indicate file transfer direction
- int **stream.footprint** = 0: use zero for production, non-zero for testing at given size (for TCP and user) { 0: }
- int **stream.icmp_cache.idle_timeout** = 180: maximum inactive time before retiring session tracker { 1: }

- int **stream.icmp_cache.max_sessions** = 65536: maximum simultaneous sessions tracked before pruning { 2: }
- int **stream.icmp_cache.pruning_timeout** = 30: minimum inactive time before being eligible for pruning { 1: }
- int **stream_icmp.session_timeout** = 30: session tracking timeout { 1:86400 }
- int **stream.ip_cache.idle_timeout** = 180: maximum inactive time before retiring session tracker { 1: }
- int **stream.ip_cache.max_sessions** = 16384: maximum simultaneous sessions tracked before pruning { 2: }
- int **stream.ip_cache.pruning_timeout** = 30: minimum inactive time before being eligible for pruning { 1: }
- bool **stream.ip_frags_only** = false: don't process non-frag flows
- int **stream_ip.max_frags** = 8192: maximum number of simultaneous fragments being tracked { 1: }
- int **stream_ip.max_overlaps** = 0: maximum allowed overlaps per datagram; 0 is unlimited { 0: }
- int **stream_ip.min_frag_length** = 0: alert if fragment length is below this limit before or after trimming { 0: }
- int **stream_ip.min_ttl** = 1: discard fragments with TTL below the minimum { 1:255 }
- enum **stream_ip.policy** = linux: fragment reassembly policy { first | linux | bsd | bsd_right | last | windows | solaris }
- int **stream_ip.session_timeout** = 30: session tracking timeout { 1:86400 }
- int **stream_ip.trace**: mask for enabling debug traces in module
- enum **stream_reassemble.action**: stop or start stream reassembly { disable|enable }
- enum **stream_reassemble.direction**: action applies to the given direction(s) { client|server|both }
- implied **stream_reassemble.fastpath**: optionally whitelist the remainder of the session
- implied **stream_reassemble.noalert**: don't alert when rule matches
- enum **stream_size.~direction**: compare applies to the given direction(s) { either|to_server|to_client|both }
- interval **stream_size.~range**: check if the stream size is in the given range { 0: }
- int **stream.tcp_cache.idle_timeout** = 3600: maximum inactive time before retiring session tracker { 1: }
- int **stream.tcp_cache.max_sessions** = 262144: maximum simultaneous sessions tracked before pruning { 2: }
- int **stream.tcp_cache.pruning_timeout** = 30: minimum inactive time before being eligible for pruning { 1: }
- int **stream_tcp.flush_factor** = 0: flush upon seeing a drop in segment size after given number of non-decreasing segments { 0: }
- int **stream_tcp.max_pdu** = 16384: maximum reassembled PDU size { 1460:32768 }
- int **stream_tcp.max_window** = 0: maximum allowed TCP window { 0:1073725440 }
- int **stream_tcp.overlap_limit** = 0: maximum number of allowed overlapping segments per session { 0:255 }
- enum **stream_tcp.policy** = bsd: determines operating system characteristics like reassembly { first | last | linux | old_linux | bsd | macos | solaris | irix | hpux11 | hpux10 | windows | win_2003 | vista | proxy }
- int **stream_tcp.queue_limit.max_bytes** = 1048576: don't queue more than given bytes per session and direction { 0: }
- int **stream_tcp.queue_limit.max_segments** = 2621: don't queue more than given segments per session and direction { 0: }
- bool **stream_tcp.reassemble_async** = true: queue data for reassembly before traffic is seen in both directions
- int **stream_tcp.require_3whs** = -1: don't track midstream sessions after given seconds from start up; -1 tracks all { -1:86400 }
- int **stream_tcp.session_timeout** = 30: session tracking timeout { 1:86400 }

- bool **stream_tcp.show_rebuilt_packets** = false: enable cmg like output of reassembled packets
- int **stream_tcp.small_segments.count** = 0: limit number of small segments queued { 0:2048 }
- int **stream_tcp.small_segments.maximum_size** = 0: limit number of small segments queued { 0:2048 }
- int **stream.trace**: mask for enabling debug traces in module
- int **stream.udp_cache.idle_timeout** = 180: maximum inactive time before retiring session tracker { 1: }
- int **stream.udp_cache.max_sessions** = 131072: maximum simultaneous sessions tracked before pruning { 2: }
- int **stream.udp_cache.pruning_timeout** = 30: minimum inactive time before being eligible for pruning { 1: }
- int **stream_udp.session_timeout** = 30: session tracking timeout { 1:86400 }
- int **stream.user_cache.idle_timeout** = 180: maximum inactive time before retiring session tracker { 1: }
- int **stream.user_cache.max_sessions** = 1024: maximum simultaneous sessions tracked before pruning { 2: }
- int **stream.user_cache.pruning_timeout** = 30: minimum inactive time before being eligible for pruning { 1: }
- int **stream_user.session_timeout** = 30: session tracking timeout { 1:86400 }
- int **stream_user.trace**: mask for enabling debug traces in module
- int **suppress[].gid** = 0: rule generator ID { 0: }
- string **suppress[].ip**: restrict suppression to these addresses according to track
- int **suppress[].sid** = 0: rule signature ID { 0: }
- enum **suppress[].track**: suppress only matching source or destination addresses { by_src | by_dst }
- int **tag.bytes**: tag for this many bytes { 1: }
- enum **tag.~**: log all packets in session or all packets to or from host { session|host_src|host_dst }
- int **tag.packets**: tag this many packets { 1: }
- int **tag.seconds**: tag for this many seconds { 1: }
- enum **target.~**: indicate the target of the attack { src_ip | dst_ip }
- string **tcp_connector.address**: address
- port **tcp_connector.base_port**: base port number
- string **tcp_connector.connector**: connector name
- enum **tcp_connector.setup**: stream establishment { call | answer }
- int **telnet.ayt_attack_thresh** = -1: alert on this number of consecutive Telnet AYT commands { -1: }
- bool **telnet.check_encrypted** = false: check for end of encryption
- bool **telnet.encrypted_traffic** = false: check for encrypted Telnet and FTP
- bool **telnet.normalize** = false: eliminate escape sequences
- interval **tos.~range**: check if IP TOS is in given range { 0:255 }
- interval **ttl.~range**: check if IP TTL is in the given range { 0:255 }
- bool **udp.deep_teredo_inspection** = false: look for Teredo on all UDP ports (default is only 3544)
- bool **udp.enable_gtp** = false: decode GTP encapsulations
- bit_list **udp.gtp_ports** = 2152 3386: set GTP ports { 65535 }

- bool **unified2.legacy_events** = false: generate Snort 2.X style events for barnyard2 compatibility
- int **unified2.limit** = 0: set maximum size in MB before rollover (0 is unlimited) { 0: }
- bool **unified2.nostamp** = true: append file creation time to name (in Unix Epoch format)
- interval **urg.~range**: check if tcp urgent offset is in given range { 0:65535 }
- interval **window.~range**: check if TCP window size is in given range { 0:65535 }
- multi **wizard.curses**: enable service identification based on internal algorithm { dce_smb | dce_udp | dce_tcp }
- bool **wizard.hexes[].client_first** = true: which end initiates data transfer
- select **wizard.hexes[].proto** = tcp: protocol to scan { tcp | udp }
- string **wizard.hexes[].service**: name of service
- string **wizard.hexes[].to_client[].hex**: sequence of data with wild chars (?)
- string **wizard.hexes[].to_server[].hex**: sequence of data with wild chars (?)
- bool **wizard.spells[].client_first** = true: which end initiates data transfer
- select **wizard.spells[].proto** = tcp: protocol to scan { tcp | udp }
- string **wizard.spells[].service**: name of service
- string **wizard.spells[].to_client[].spell**: sequence of data with wild cards (*)
- string **wizard.spells[].to_server[].spell**: sequence of data with wild cards (*)
- interval **wscale.~range**: check if TCP window scale is in given range { 0:65535 }

20.5 Counts

- **appid.appid_unknown**: count of sessions where appid could not be determined (sum)
- **appid.ignored_packets**: count of packets ignored (sum)
- **appid.packets**: count of packets received (sum)
- **appid.processed_packets**: count of packets processed (sum)
- **appid.total_sessions**: count of sessions created (sum)
- **arp_spoof.packets**: total packets (sum)
- **back_orifice.packets**: total packets (sum)
- **binder.allows**: allow bindings (sum)
- **binder.blocks**: block bindings (sum)
- **binder.inspects**: inspect bindings (sum)
- **binder.packets**: initial bindings (sum)
- **binder.resets**: reset bindings (sum)
- **daq.allow**: total allow verdicts (sum)
- **daq.analyzed**: total packets analyzed from DAQ (sum)
- **daq.blacklist**: total blacklist verdicts (sum)
- **daq.block**: total block verdicts (sum)

- **daq.dropped**: packets dropped (sum)
- **daq.filtered**: packets filtered out (sum)
- **daq.idle**: attempts to acquire from DAQ without available packets (sum)
- **daq.ignore**: total ignore verdicts (sum)
- **daq.injected**: active responses or replacements (sum)
- **daq.internal_blacklist**: packets blacklisted internally due to lack of DAQ support (sum)
- **daq.internal_whitelist**: packets whitelisted internally due to lack of DAQ support (sum)
- **daq.outstanding**: packets unprocessed (sum)
- **daq.pcaps**: total files and interfaces processed (max)
- **daq.received**: total packets received from DAQ (sum)
- **daq.replace**: total replace verdicts (sum)
- **daq.retry**: total retry verdicts (sum)
- **daq.rx_bytes**: total bytes received (sum)
- **daq.skipped**: packets skipped at startup (sum)
- **daq.whitelist**: total whitelist verdicts (sum)
- **data_log.packets**: total packets (sum)
- **dce_http_proxy.http_proxy_session_failures**: failed http proxy sessions (sum)
- **dce_http_proxy.http_proxy_sessions**: successful http proxy sessions (sum)
- **dce_http_server.http_server_session_failures**: failed http server sessions (sum)
- **dce_http_server.http_server_sessions**: successful http server sessions (sum)
- **dce_smb.alter_context_responses**: total connection-oriented alter context responses (sum)
- **dce_smb.alter_contexts**: total connection-oriented alter contexts (sum)
- **dce_smb.auth3s**: total connection-oriented auth3s (sum)
- **dce_smb.bind_acks**: total connection-oriented binds acks (sum)
- **dce_smb.bind_naks**: total connection-oriented bind naks (sum)
- **dce_smb.binds**: total connection-oriented binds (sum)
- **dce_smb.cancels**: total connection-oriented cancels (sum)
- **dce_smb.client_frags_reassembled**: total connection-oriented client fragments reassembled (sum)
- **dce_smb.client_max_fragment_size**: connection-oriented client maximum fragment size (sum)
- **dce_smb.client_min_fragment_size**: connection-oriented client minimum fragment size (sum)
- **dce_smb.client_segs_reassembled**: total connection-oriented client segments reassembled (sum)
- **dce_smb.concurrent_sessions**: total concurrent sessions (now)
- **dce_smb.events**: total events (sum)
- **dce_smb.faults**: total connection-oriented faults (sum)
- **dce_smb.files_processed**: total smb files processed (sum)

- **dce_smb.ignored_bytes**: total ignored bytes (sum)
- **dce_smb.max_concurrent_sessions**: maximum concurrent sessions (max)
- **dce_smb.max_outstanding_requests**: total smb maximum outstanding requests (sum)
- **dce_smb.ms_rpc_http_pdus**: total connection-oriented MS requests to send RPC over HTTP (sum)
- **dce_smb.orphaned**: total connection-oriented orphaned (sum)
- **dce_smb.other_requests**: total connection-oriented other requests (sum)
- **dce_smb.other_responses**: total connection-oriented other responses (sum)
- **dce_smb.packets**: total smb packets (sum)
- **dce_smb.pdus**: total connection-oriented PDUs (sum)
- **dce_smb.rejects**: total connection-oriented rejects (sum)
- **dce_smb.request_fragments**: total connection-oriented request fragments (sum)
- **dce_smb.requests**: total connection-oriented requests (sum)
- **dce_smb.response_fragments**: total connection-oriented response fragments (sum)
- **dce_smb.responses**: total connection-oriented responses (sum)
- **dce_smb.server_frags_reassembled**: total connection-oriented server fragments reassembled (sum)
- **dce_smb.server_max_fragment_size**: connection-oriented server maximum fragment size (sum)
- **dce_smb.server_min_fragment_size**: connection-oriented server minimum fragment size (sum)
- **dce_smb.server_segs_reassembled**: total connection-oriented server segments reassembled (sum)
- **dce_smb.sessions**: total smb sessions (sum)
- **dce_smb.shutdowns**: total connection-oriented shutdowns (sum)
- **dce_smb.smb_client_segs_reassembled**: total smb client segments reassembled (sum)
- **dce_smb.smb_server_segs_reassembled**: total smb server segments reassembled (sum)
- **dce_smb.smbv2_close**: total number of SMBv2 close packets seen (sum)
- **dce_smb.smbv2_create**: total number of SMBv2 create packets seen (sum)
- **dce_smb.smbv2_read**: total number of SMBv2 read packets seen (sum)
- **dce_smb.smbv2_set_info**: total number of SMBv2 set info packets seen (sum)
- **dce_smb.smbv2_tree_connect**: total number of SMBv2 tree connect packets seen (sum)
- **dce_smb.smbv2_tree_disconnect**: total number of SMBv2 tree disconnect packets seen (sum)
- **dce_smb.smbv2_write**: total number of SMBv2 write packets seen (sum)
- **dce_tcp.alter_context_responses**: total connection-oriented alter context responses (sum)
- **dce_tcp.alter_contexts**: total connection-oriented alter contexts (sum)
- **dce_tcp.auth3s**: total connection-oriented auth3s (sum)
- **dce_tcp.bind_acks**: total connection-oriented binds acks (sum)
- **dce_tcp.bind_naks**: total connection-oriented bind naks (sum)
- **dce_tcp.binds**: total connection-oriented binds (sum)

- **dce_tcp.cancels**: total connection-oriented cancels (sum)
- **dce_tcp.client_frags_reassembled**: total connection-oriented client fragments reassembled (sum)
- **dce_tcp.client_max_fragment_size**: connection-oriented client maximum fragment size (sum)
- **dce_tcp.client_min_fragment_size**: connection-oriented client minimum fragment size (sum)
- **dce_tcp.client_segs_reassembled**: total connection-oriented client segments reassembled (sum)
- **dce_tcp.concurrent_sessions**: total concurrent sessions (now)
- **dce_tcp.events**: total events (sum)
- **dce_tcp.faults**: total connection-oriented faults (sum)
- **dce_tcp.max_concurrent_sessions**: maximum concurrent sessions (max)
- **dce_tcp.ms_rpc_http_pdus**: total connection-oriented MS requests to send RPC over HTTP (sum)
- **dce_tcp.orphaned**: total connection-oriented orphaned (sum)
- **dce_tcp.other_requests**: total connection-oriented other requests (sum)
- **dce_tcp.other_responses**: total connection-oriented other responses (sum)
- **dce_tcp.pdus**: total connection-oriented PDUs (sum)
- **dce_tcp.rejects**: total connection-oriented rejects (sum)
- **dce_tcp.request_fragments**: total connection-oriented request fragments (sum)
- **dce_tcp.requests**: total connection-oriented requests (sum)
- **dce_tcp.response_fragments**: total connection-oriented response fragments (sum)
- **dce_tcp.responses**: total connection-oriented responses (sum)
- **dce_tcp.server_frags_reassembled**: total connection-oriented server fragments reassembled (sum)
- **dce_tcp.server_max_fragment_size**: connection-oriented server maximum fragment size (sum)
- **dce_tcp.server_min_fragment_size**: connection-oriented server minimum fragment size (sum)
- **dce_tcp.server_segs_reassembled**: total connection-oriented server segments reassembled (sum)
- **dce_tcp.shutdowns**: total connection-oriented shutdowns (sum)
- **dce_tcp.tcp_packets**: total tcp packets (sum)
- **dce_tcp.tcp_sessions**: total tcp sessions (sum)
- **dce_udp.acks**: total connection-less acks (sum)
- **dce_udp.cancel_acks**: total connection-less cancel acks (sum)
- **dce_udp.cancels**: total connection-less cancels (sum)
- **dce_udp.client_facks**: total connection-less client facks (sum)
- **dce_udp.concurrent_sessions**: total concurrent sessions (now)
- **dce_udp.events**: total events (sum)
- **dce_udp.faults**: total connection-less faults (sum)
- **dce_udp.fragments**: total connection-less fragments (sum)
- **dce_udp.frags_reassembled**: total connection-less fragments reassembled (sum)

- **dce_udp.max_concurrent_sessions**: maximum concurrent sessions (max)
- **dce_udp.max_fragment_size**: connection-less maximum fragment size (sum)
- **dce_udp.max_seqnum**: max connection-less seqnum (sum)
- **dce_udp.no_calls**: total connection-less no calls (sum)
- **dce_udp.other_requests**: total connection-less other requests (sum)
- **dce_udp.other_responses**: total connection-less other responses (sum)
- **dce_udp.ping**: total connection-less ping (sum)
- **dce_udp.rejects**: total connection-less rejects (sum)
- **dce_udp.requests**: total connection-less requests (sum)
- **dce_udp.responses**: total connection-less responses (sum)
- **dce_udp.server_facks**: total connection-less server facks (sum)
- **dce_udp.udp_packets**: total udp packets (sum)
- **dce_udp.udp_sessions**: total udp sessions (sum)
- **dce_udp.working**: total connection-less working (sum)
- **detection.alert_limit**: events previously triggered on same PDU (sum)
- **detection.alerts**: alerts not including IP reputation (sum)
- **detection.alt_searches**: alt fast pattern searches in packet data (sum)
- **detection.analyzed**: packets sent to detection (sum)
- **detection.body_searches**: fast pattern searches in body buffer (sum)
- **detection.cooked_searches**: fast pattern searches in cooked packet data (sum)
- **detection.event_limit**: events filtered (sum)
- **detection.file_searches**: fast pattern searches in file buffer (sum)
- **detection.hard_evals**: non-fast pattern rule evaluations (sum)
- **detection.header_searches**: fast pattern searches in header buffer (sum)
- **detection.key_searches**: fast pattern searches in key buffer (sum)
- **detection.logged**: logged packets (sum)
- **detection.log_limit**: events queued but not logged (sum)
- **detection.match_limit**: fast pattern matches not processed (sum)
- **detection.offloads**: fast pattern searches that were offloaded (sum)
- **detection.passed**: passed packets (sum)
- **detection.pkt_searches**: fast pattern searches in packet data (sum)
- **detection.queue_limit**: events not queued because queue full (sum)
- **detection.raw_searches**: fast pattern searches in raw packet data (sum)
- **detection.total_alerts**: alerts including IP reputation (sum)
- **dnp3.concurrent_sessions**: total concurrent dnp3 sessions (now)

- **dnp3.dnp3_application_pdus**: total dnp3 application pdus (sum)
- **dnp3.dnp3_link_layer_frames**: total dnp3 link layer frames (sum)
- **dnp3.max_concurrent_sessions**: maximum concurrent dnp3 sessions (max)
- **dnp3.tcp_pdus**: total tcp pdus (sum)
- **dnp3.total_packets**: total packets (sum)
- **dnp3.udp_packets**: total udp packets (sum)
- **dns.concurrent_sessions**: total concurrent dns sessions (now)
- **dns.max_concurrent_sessions**: maximum concurrent dns sessions (max)
- **dns.packets**: total packets processed (sum)
- **dns.requests**: total dns requests (sum)
- **dns.responses**: total dns responses (sum)
- **domain_filter.checked**: domains checked (sum)
- **domain_filter.filtered**: domains filtered (sum)
- **dpx.packets**: total packets (sum)
- **file_connector.messages**: total messages (sum)
- **file_id.cache_failures**: number of file cache add failures (sum)
- **file_id.total_file_data**: number of file data bytes processed (sum)
- **file_id.total_files**: number of files processed (sum)
- **file_log.total_events**: total file events (sum)
- **ftp_data.packets**: total packets (sum)
- **ftp_server.concurrent_sessions**: total concurrent FTP sessions (now)
- **ftp_server.max_concurrent_sessions**: maximum concurrent FTP sessions (max)
- **ftp_server.total_packets**: total packets (sum)
- **gtp_inspect.concurrent_sessions**: total concurrent gtp sessions (now)
- **gtp_inspect.events**: requests (sum)
- **gtp_inspect.max_concurrent_sessions**: maximum concurrent gtp sessions (max)
- **gtp_inspect.sessions**: total sessions processed (sum)
- **gtp_inspect.unknown_infos**: unknown information elements (sum)
- **gtp_inspect.unknown_types**: unknown message types (sum)
- **high_availability.packets**: total packets (sum)
- **host_cache.lru_cache_adds**: lru cache added new entry (sum)
- **host_cache.lru_cache_clears**: lru cache clear API calls (sum)
- **host_cache.lru_cache_find_hits**: lru cache found entry in cache (sum)
- **host_cache.lru_cache_find_misses**: lru cache did not find entry in cache (sum)
- **host_cache.lru_cache_prunes**: lru cache pruned entry to make space for new entry (sum)

- **host_cache.lru_cache_removes**: lru cache found entry and removed it (sum)
- **host_cache.lru_cache_replaces**: lru cache replaced existing entry (sum)
- **host_tracker.service_adds**: host service adds (sum)
- **host_tracker.service_finds**: host service finds (sum)
- **host_tracker.service_removes**: host service removes (sum)
- **http2_inspect.concurrent_sessions**: total concurrent HTTP/2 sessions (now)
- **http2_inspect.flows**: HTTP connections inspected (sum)
- **http2_inspect.max_concurrent_sessions**: maximum concurrent HTTP/2 sessions (max)
- **http_inspect.chunked**: chunked message bodies (sum)
- **http_inspect.concurrent_sessions**: total concurrent http sessions (now)
- **http_inspect.connect_requests**: CONNECT requests inspected (sum)
- **http_inspect.delete_requests**: DELETE requests inspected (sum)
- **http_inspect.flows**: HTTP connections inspected (sum)
- **http_inspect.get_requests**: GET requests inspected (sum)
- **http_inspect.head_requests**: HEAD requests inspected (sum)
- **http_inspect.inspections**: total message sections inspected (sum)
- **http_inspect.max_concurrent_sessions**: maximum concurrent http sessions (max)
- **http_inspect.options_requests**: OPTIONS requests inspected (sum)
- **http_inspect.other_requests**: other request methods inspected (sum)
- **http_inspect.post_requests**: POST requests inspected (sum)
- **http_inspect.put_requests**: PUT requests inspected (sum)
- **http_inspect.reassembles**: TCP segments combined into HTTP messages (sum)
- **http_inspect.request_bodies**: POST, PUT, and other requests with message bodies (sum)
- **http_inspect.requests**: HTTP request messages inspected (sum)
- **http_inspect.responses**: HTTP response messages inspected (sum)
- **http_inspect.scans**: TCP segments scanned looking for HTTP messages (sum)
- **http_inspect.trace_requests**: TRACE requests inspected (sum)
- **http_inspect.uri_coding**: URIs with character coding problems (sum)
- **http_inspect.uri_normalizations**: URIs needing to be normalization (sum)
- **http_inspect.uri_path**: URIs with path problems (sum)
- **icmp4.bad_checksum**: non-zero icmp checksums (sum)
- **icmp6.bad_icmp6_checksum**: nonzero icmp6 checksums (sum)
- **imap.b64_attachments**: total base64 attachments decoded (sum)
- **imap.b64_decoded_bytes**: total base64 decoded bytes (sum)
- **imap.concurrent_sessions**: total concurrent imap sessions (now)

- **imap.max_concurrent_sessions**: maximum concurrent imap sessions (max)
- **imap.non_encoded_attachments**: total non-encoded attachments extracted (sum)
- **imap.non_encoded_bytes**: total non-encoded extracted bytes (sum)
- **imap.packets**: total packets processed (sum)
- **imap.qp_attachments**: total quoted-printable attachments decoded (sum)
- **imap.qp_decoded_bytes**: total quoted-printable decoded bytes (sum)
- **imap.sessions**: total imap sessions (sum)
- **imap.uu_attachments**: total uu attachments decoded (sum)
- **imap.uu_decoded_bytes**: total uu decoded bytes (sum)
- **ipv4.bad_checksum**: nonzero ip checksums (sum)
- **latency.max_usecs**: maximum usecs elapsed (sum)
- **latency.packet_timeouts**: packets that timed out (sum)
- **latency.rule_eval_timeouts**: rule evals that timed out (sum)
- **latency.rule_tree_enables**: rule tree re-enables (sum)
- **latency.total_packets**: total packets monitored (sum)
- **latency.total_rule_evals**: total rule evals monitored (sum)
- **latency.total_usecs**: total usecs elapsed (sum)
- **modbus.concurrent_sessions**: total concurrent modbus sessions (now)
- **modbus.frames**: total Modbus messages (sum)
- **modbus.max_concurrent_sessions**: maximum concurrent modbus sessions (max)
- **modbus.sessions**: total sessions processed (sum)
- **mpls.total_bytes**: total mpls labeled bytes processed (sum)
- **mpls.total_packets**: total mpls labeled packets processed (sum)
- **normalizer.icmp4_echo**: icmp4 ping normalizations (sum)
- **normalizer.icmp6_echo**: icmp6 echo normalizations (sum)
- **normalizer.ip4_df**: don't frag bit normalizations (sum)
- **normalizer.ip4_opts**: ip4 options cleared (sum)
- **normalizer.ip4_rf**: reserved flag bit clears (sum)
- **normalizer.ip4_tos**: type of service normalizations (sum)
- **normalizer.ip4_trim**: eth packets trimmed to datagram size (sum)
- **normalizer.ip4_ttl**: time-to-live normalizations (sum)
- **normalizer.ip6_hops**: ip6 hop limit normalizations (sum)
- **normalizer.ip6_options**: ip6 options cleared (sum)
- **normalizer.tcp_block**: blocked segments (sum)
- **normalizer.tcp_ecn_pkt**: packets with ECN bits cleared (sum)

- **normalizer.tcp_ecn_session**: ECN bits cleared (sum)
- **normalizer.tcp_ips_data**: normalized segments (sum)
- **normalizer.tcp_nonce**: packets with nonce bit cleared (sum)
- **normalizer.tcp_options**: packets with options cleared (sum)
- **normalizer.tcp_padding**: packets with padding cleared (sum)
- **normalizer.tcp_req_pay**: cleared urgent pointer and urgent flag when there is no payload (sum)
- **normalizer.tcp_req_urg**: cleared urgent pointer when urgent flag is not set (sum)
- **normalizer.tcp_req_urp**: cleared the urgent flag if the urgent pointer is not set (sum)
- **normalizer.tcp_reserved**: packets with reserved bits cleared (sum)
- **normalizer.tcp_syn_options**: SYN only options cleared from non-SYN packets (sum)
- **normalizer.tcp_trim_mss**: data trimmed to MSS (sum)
- **normalizer.tcp_trim_rst**: RST packets with data trimmed (sum)
- **normalizer.tcp_trim_syn**: tcp segments trimmed on SYN (sum)
- **normalizer.tcp_trim_win**: data trimmed to window (sum)
- **normalizer.tcp_ts_ecr**: timestamp cleared on non-ACKs (sum)
- **normalizer.tcp_ts_nop**: timestamp options cleared (sum)
- **normalizer.tcp_urgent_ptr**: packets without data with urgent pointer cleared (sum)
- **normalizer.test_icmp4_echo**: test icmp4 ping normalizations (sum)
- **normalizer.test_icmp6_echo**: test icmp6 echo normalizations (sum)
- **normalizer.test_ip4_df**: test don't frag bit normalizations (sum)
- **normalizer.test_ip4_opts**: test ip4 options cleared (sum)
- **normalizer.test_ip4_rf**: test reserved flag bit clears (sum)
- **normalizer.test_ip4_tos**: test type of service normalizations (sum)
- **normalizer.test_ip4_trim**: test eth packets trimmed to datagram size (sum)
- **normalizer.test_ip4_ttl**: test time-to-live normalizations (sum)
- **normalizer.test_ip6_hops**: test ip6 hop limit normalizations (sum)
- **normalizer.test_ip6_options**: test ip6 options cleared (sum)
- **normalizer.test_tcp_block**: test blocked segments (sum)
- **normalizer.test_tcp_ecn_pkt**: test packets with ECN bits cleared (sum)
- **normalizer.test_tcp_ecn_session**: test ECN bits cleared (sum)
- **normalizer.test_tcp_ips_data**: test normalized segments (sum)
- **normalizer.test_tcp_nonce**: test packets with nonce bit cleared (sum)
- **normalizer.test_tcp_options**: test packets with options cleared (sum)
- **normalizer.test_tcp_padding**: test packets with padding cleared (sum)
- **normalizer.test_tcp_req_pay**: test cleared urgent pointer and urgent flag when there is no payload (sum)

- **normalizer.test_tcp_req_urg**: test cleared urgent pointer when urgent flag is not set (sum)
- **normalizer.test_tcp_req_urp**: test cleared the urgent flag if the urgent pointer is not set (sum)
- **normalizer.test_tcp_reserved**: test packets with reserved bits cleared (sum)
- **normalizer.test_tcp_syn_options**: test SYN only options cleared from non-SYN packets (sum)
- **normalizer.test_tcp_trim_mss**: test data trimmed to MSS (sum)
- **normalizer.test_tcp_trim_rst**: test RST packets with data trimmed (sum)
- **normalizer.test_tcp_trim_syn**: test tcp segments trimmed on SYN (sum)
- **normalizer.test_tcp_trim_win**: test data trimmed to window (sum)
- **normalizer.test_tcp_ts_ecr**: test timestamp cleared on non-ACKs (sum)
- **normalizer.test_tcp_ts_nop**: test timestamp options cleared (sum)
- **normalizer.test_tcp_urgent_ptr**: test packets without data with urgent pointer cleared (sum)
- **packet_capture.captured**: packets matching dumped after matching filter (sum)
- **packet_capture.processed**: packets processed against filter (sum)
- **perf_monitor.packets**: total packets (sum)
- **pop.b64_attachments**: total base64 attachments decoded (sum)
- **pop.b64_decoded_bytes**: total base64 decoded bytes (sum)
- **pop.concurrent_sessions**: total concurrent pop sessions (now)
- **pop.max_concurrent_sessions**: maximum concurrent pop sessions (max)
- **pop.non_encoded_attachments**: total non-encoded attachments extracted (sum)
- **pop.non_encoded_bytes**: total non-encoded extracted bytes (sum)
- **pop.packets**: total packets processed (sum)
- **pop.qp_attachments**: total quoted-printable attachments decoded (sum)
- **pop.qp_decoded_bytes**: total quoted-printable decoded bytes (sum)
- **pop.sessions**: total pop sessions (sum)
- **pop.uu_attachments**: total uu attachments decoded (sum)
- **pop.uu_decoded_bytes**: total uu decoded bytes (sum)
- **port_scan.packets**: total packets (sum)
- **reg_test.packets**: total packets (sum)
- **reg_test.retry_packets**: total retried packets received (sum)
- **reg_test.retry_requests**: total retry packets requested (sum)
- **reputation.blacklisted**: number of packets blacklisted (sum)
- **reputation.memory_allocated**: total memory allocated (sum)
- **reputation.monitored**: number of packets monitored (sum)
- **reputation.packets**: total packets processed (sum)
- **reputation.whitelisted**: number of packets whitelisted (sum)

- **rpc_decode.concurrent_sessions**: total concurrent rpc sessions (now)
- **rpc_decode.max_concurrent_sessions**: maximum concurrent rpc sessions (max)
- **rpc_decode.total_packets**: total packets (sum)
- **sd_pattern.below_threshold**: sd_pattern matched but missed threshold (sum)
- **sd_pattern.pattern_not_found**: sd_pattern did not not match (sum)
- **sd_pattern.terminated**: hyperscan terminated (sum)
- **search_engine.max_queued**: maximum fast pattern matches queued for further evaluation (sum)
- **search_engine.non_qualified_events**: total non-qualified events (sum)
- **search_engine.qualified_events**: total qualified events (sum)
- **search_engine.searched_bytes**: total bytes searched (sum)
- **search_engine.total_flushed**: fast pattern matches discarded due to overflow (sum)
- **search_engine.total_inserts**: total fast pattern hits (sum)
- **search_engine.total_unique**: total unique fast pattern hits (sum)
- **side_channel.packets**: total packets (sum)
- **sip.ack**: ack (sum)
- **sip.bye**: bye (sum)
- **sip.cancel**: cancel (sum)
- **sip.code_1xx**: 1xx (sum)
- **sip.code_2xx**: 2xx (sum)
- **sip.code_3xx**: 3xx (sum)
- **sip.code_4xx**: 4xx (sum)
- **sip.code_5xx**: 5xx (sum)
- **sip.code_6xx**: 6xx (sum)
- **sip.code_7xx**: 7xx (sum)
- **sip.code_8xx**: 8xx (sum)
- **sip.code_9xx**: 9xx (sum)
- **sip.concurrent_sessions**: total concurrent SIP sessions (now)
- **sip.dialogs**: total dialogs (sum)
- **sip.events**: events generated (sum)
- **sip.ignored_channels**: total channels ignored (sum)
- **sip.ignored_sessions**: total sessions ignored (sum)
- **sip.info**: info (sum)
- **sip.invite**: invite (sum)
- **sip.join**: join (sum)
- **sip.max_concurrent_sessions**: maximum concurrent SIP sessions (max)

- **sip.message**: message (sum)
- **sip.notify**: notify (sum)
- **sip.options**: options (sum)
- **sip.packets**: total packets (sum)
- **sip.prack**: prack (sum)
- **sip.refer**: refer (sum)
- **sip.register**: register (sum)
- **sip.sessions**: total sessions (sum)
- **sip.subscribe**: subscribe (sum)
- **sip.total_requests**: total requests (sum)
- **sip.total_responses**: total responses (sum)
- **sip.update**: update (sum)
- **smtp.b64_attachments**: total base64 attachments decoded (sum)
- **smtp.b64_decoded_bytes**: total base64 decoded bytes (sum)
- **smtp.concurrent_sessions**: total concurrent smtp sessions (now)
- **smtp.max_concurrent_sessions**: maximum concurrent smtp sessions (max)
- **smtp.non_encoded_attachments**: total non-encoded attachments extracted (sum)
- **smtp.non_encoded_bytes**: total non-encoded extracted bytes (sum)
- **smtp.packets**: total packets processed (sum)
- **smtp.qp_attachments**: total quoted-printable attachments decoded (sum)
- **smtp.qp_decoded_bytes**: total quoted-printable decoded bytes (sum)
- **smtp.sessions**: total smtp sessions (sum)
- **smtp.uu_attachments**: total uu attachments decoded (sum)
- **smtp.uu_decoded_bytes**: total uu decoded bytes (sum)
- **snort.attribute_table_hosts**: total number of hosts in table (sum)
- **snort.attribute_table_reloads**: number of times hosts table was reloaded (sum)
- **snort.conf_reloads**: number of times configuration was reloaded (sum)
- **snort.daq_reloads**: number of times daq configuration was reloaded (sum)
- **snort.inspector_deletions**: number of times inspectors were deleted (sum)
- **snort.local_commands**: total local commands processed (sum)
- **snort.policy_reloads**: number of times policies were reloaded (sum)
- **snort.remote_commands**: total remote commands processed (sum)
- **snort.signals**: total signals processed (sum)
- **ssh.concurrent_sessions**: total concurrent ssh sessions (now)
- **ssh.max_concurrent_sessions**: maximum concurrent ssh sessions (max)

- **ssh.packets**: total packets (sum)
- **ssl.alert**: total ssl alert records (sum)
- **ssl.bad_handshakes**: total bad handshakes (sum)
- **ssl.certificate**: total ssl certificates (sum)
- **ssl.change_cipher**: total change cipher records (sum)
- **ssl.client_application**: total client application records (sum)
- **ssl.client_hello**: total client hellos (sum)
- **ssl.client_key_exchange**: total client key exchanges (sum)
- **ssl.concurrent_sessions**: total concurrent ssl sessions (now)
- **ssl.decoded**: ssl packets decoded (sum)
- **ssl.detection_disabled**: total detection disabled (sum)
- **ssl.finished**: total handshakes finished (sum)
- **ssl.handshakes_completed**: total completed ssl handshakes (sum)
- **ssl.max_concurrent_sessions**: maximum concurrent ssl sessions (max)
- **ssl.packets**: total packets processed (sum)
- **ssl.server_application**: total server application records (sum)
- **ssl.server_done**: total server done (sum)
- **ssl.server_hello**: total server hellos (sum)
- **ssl.server_key_exchange**: total server key exchanges (sum)
- **ssl.sessions_ignored**: total sessions ignore (sum)
- **ssl.unrecognized_records**: total unrecognized records (sum)
- **stream.file_excess_prunes**: file sessions pruned due to excess (sum)
- **stream.file_flows**: total file sessions (sum)
- **stream.file_ha_prunes**: file sessions pruned by high availability sync (sum)
- **stream.file_idle_prunes**: file sessions pruned due to timeout (sum)
- **stream.file_memcap_prunes**: file sessions pruned due to memcap (sum)
- **stream.file_preemptive_prunes**: file sessions pruned during preemptive pruning (sum)
- **stream.file_total_prunes**: total file sessions pruned (sum)
- **stream.file_uni_prunes**: file uni sessions pruned (sum)
- **stream_icmp.created**: icmp session trackers created (sum)
- **stream.icmp_excess_prunes**: icmp sessions pruned due to excess (sum)
- **stream.icmp_flows**: total icmp sessions (sum)
- **stream.icmp_ha_prunes**: icmp sessions pruned by high availability sync (sum)
- **stream.icmp_idle_prunes**: icmp sessions pruned due to timeout (sum)
- **stream_icmp.max**: max icmp sessions (max)

- **stream.icmp_memcap_prunes**: icmp sessions pruned due to memcap (sum)
- **stream.icmp_preemptive_prunes**: icmp sessions pruned during preemptive pruning (sum)
- **stream_icmp.prunes**: icmp session prunes (sum)
- **stream_icmp.released**: icmp session trackers released (sum)
- **stream_icmp.sessions**: total icmp sessions (sum)
- **stream_icmp.timeouts**: icmp session timeouts (sum)
- **stream.icmp_total_prunes**: total icmp sessions pruned (sum)
- **stream.icmp_uni_prunes**: icmp uni sessions pruned (sum)
- **stream_ip.alerts**: alerts generated (sum)
- **stream_ip.anomalies**: anomalies detected (sum)
- **stream_ip.created**: ip session trackers created (sum)
- **stream_ip.current_frags**: current fragments (now)
- **stream_ip.discards**: fragments discarded (sum)
- **stream_ip.drops**: fragments dropped (sum)
- **stream.ip_excess_prunes**: ip sessions pruned due to excess (sum)
- **stream.ip_flows**: total ip sessions (sum)
- **stream_ip.fragmented_bytes**: total fragmented bytes (sum)
- **stream_ip.frag_timeouts**: datagrams abandoned (sum)
- **stream.ip_ha_prunes**: ip sessions pruned by high availability sync (sum)
- **stream.ip_idle_prunes**: ip sessions pruned due to timeout (sum)
- **stream_ip.max_frags**: max fragments (sum)
- **stream_ip.max**: max ip sessions (max)
- **stream.ip_memcap_prunes**: ip sessions pruned due to memcap (sum)
- **stream_ip.nodes_deleted**: fragments deleted from tracker (sum)
- **stream_ip.nodes_inserted**: fragments added to tracker (sum)
- **stream_ip.overlaps**: overlapping fragments (sum)
- **stream.ip_preemptive_prunes**: ip sessions pruned during preemptive pruning (sum)
- **stream_ip.prunes**: ip session prunes (sum)
- **stream_ip.reassembled_bytes**: total reassembled bytes (sum)
- **stream_ip.reassembled**: reassembled datagrams (sum)
- **stream_ip.released**: ip session trackers released (sum)
- **stream_ip.sessions**: total ip sessions (sum)
- **stream_ip.timeouts**: ip session timeouts (sum)
- **stream_ip.total_frags**: total fragments (sum)
- **stream.ip_total_prunes**: total ip sessions pruned (sum)

- **stream_ip.trackers_added**: datagram trackers created (sum)
- **stream_ip.trackers_cleared**: datagram trackers cleared (sum)
- **stream_ip.trackers_completed**: datagram trackers completed (sum)
- **stream_ip.trackers_freed**: datagram trackers released (sum)
- **stream.ip_uni_prunes**: ip uni sessions pruned (sum)
- **stream_tcp.client_cleanups**: number of times data from server was flushed when session released (sum)
- **stream_tcp.closing**: number of sessions currently closing (now)
- **stream_tcp.created**: tcp session trackers created (sum)
- **stream_tcp.data_trackers**: tcp session tracking started on data (sum)
- **stream_tcp.discards**: tcp packets discarded (sum)
- **stream_tcp.established**: number of sessions currently established (now)
- **stream_tcp.events**: events generated (sum)
- **stream_tcp.exceeded_max_bytes**: number of times the maximum queued byte limit was reached (sum)
- **stream_tcp.exceeded_max_segs**: number of times the maximum queued segment limit was reached (sum)
- **stream.tcp_excess_prunes**: tcp sessions pruned due to excess (sum)
- **stream_tcp.fins**: number of fin packets (sum)
- **stream.tcp_flows**: total tcp sessions (sum)
- **stream_tcp.gaps**: missing data between PDUs (sum)
- **stream.tcp_ha_prunes**: tcp sessions pruned by high availability sync (sum)
- **stream.tcp_idle_prunes**: tcp sessions pruned due to timeout (sum)
- **stream_tcp.ignored**: tcp packets ignored (sum)
- **stream_tcp.initializing**: number of sessions currently initializing (now)
- **stream_tcp.instantiated**: new sessions instantiated (sum)
- **stream_tcp.internal_events**: 135:X events generated (sum)
- **stream_tcp.max**: max tcp sessions (max)
- **stream.tcp_memcap_prunes**: tcp sessions pruned due to memcap (sum)
- **stream_tcp.memory**: current memory in use (now)
- **stream_tcp.overlaps**: overlapping segments queued (sum)
- **stream.tcp_preemptive_prunes**: tcp sessions pruned during preemptive pruning (sum)
- **stream_tcp.prunes**: tcp session prunes (sum)
- **stream_tcp.rebuilt_buffers**: rebuilt PDU sections (sum)
- **stream_tcp.rebuilt_bytes**: total rebuilt bytes (sum)
- **stream_tcp.rebuilt_packets**: total reassembled PDUs (sum)
- **stream_tcp.released**: tcp session trackers released (sum)
- **stream_tcp.resets**: number of reset packets (sum)

- **stream_tcp.restarts**: sessions restarted (sum)
- **stream_tcp.resyns**: SYN received on established session (sum)
- **stream_tcp.segs_queued**: total segments queued (sum)
- **stream_tcp.segs_released**: total segments released (sum)
- **stream_tcp.segs_split**: tcp segments split when reassembling PDUs (sum)
- **stream_tcp.segs_used**: queued tcp segments applied to reassembled PDUs (sum)
- **stream_tcp.server_cleanups**: number of times data from client was flushed when session released (sum)
- **stream_tcp.sessions**: total tcp sessions (sum)
- **stream_tcp.setups**: session initializations (sum)
- **stream_tcp.syn_acks**: number of syn-ack packets (sum)
- **stream_tcp.syn_ack_trackers**: tcp session tracking started on syn-ack (sum)
- **stream_tcp.syns**: number of syn packets (sum)
- **stream_tcp.syn_trackers**: tcp session tracking started on syn (sum)
- **stream_tcp.three_way_trackers**: tcp session tracking started on ack (sum)
- **stream_tcp.timeouts**: tcp session timeouts (sum)
- **stream.tcp_total_prunes**: total tcp sessions pruned (sum)
- **stream.tcp_uni_prunes**: tcp uni sessions pruned (sum)
- **stream_tcp.untracked**: tcp packets not tracked (sum)
- **stream_udp.created**: udp session trackers created (sum)
- **stream.udp_excess_prunes**: udp sessions pruned due to excess (sum)
- **stream.udp_flows**: total udp sessions (sum)
- **stream.udp_ha_prunes**: udp sessions pruned by high availability sync (sum)
- **stream.udp_idle_prunes**: udp sessions pruned due to timeout (sum)
- **stream_udp.ignored**: udp packets ignored (sum)
- **stream_udp.max**: max udp sessions (max)
- **stream.udp_memcap_prunes**: udp sessions pruned due to memcap (sum)
- **stream.udp_preemptive_prunes**: udp sessions pruned during preemptive pruning (sum)
- **stream_udp.prunes**: udp session prunes (sum)
- **stream_udp.released**: udp session trackers released (sum)
- **stream_udp.sessions**: total udp sessions (sum)
- **stream_udp.timeouts**: udp session timeouts (sum)
- **stream.udp_total_prunes**: total udp sessions pruned (sum)
- **stream.udp_uni_prunes**: udp uni sessions pruned (sum)
- **stream.user_excess_prunes**: user sessions pruned due to excess (sum)
- **stream.user_flows**: total user sessions (sum)

- **stream.user_ha_prunes**: user sessions pruned by high availability sync (sum)
- **stream.user_idle_prunes**: user sessions pruned due to timeout (sum)
- **stream.user_memcap_prunes**: user sessions pruned due to memcap (sum)
- **stream.user_preemptive_prunes**: user sessions pruned during preemptive pruning (sum)
- **stream.user_total_prunes**: total user sessions pruned (sum)
- **stream.user_uni_prunes**: user uni sessions pruned (sum)
- **tcp.bad_tcp4_checksum**: nonzero tcp over ip checksums (sum)
- **tcp.bad_tcp6_checksum**: nonzero tcp over ipv6 checksums (sum)
- **tcp_connector.messages**: total messages (sum)
- **telnet.concurrent_sessions**: total concurrent Telnet sessions (now)
- **telnet.max_concurrent_sessions**: maximum concurrent Telnet sessions (max)
- **telnet.total_packets**: total packets (sum)
- **udp.bad_udp4_checksum**: nonzero udp over ipv4 checksums (sum)
- **udp.bad_udp6_checksum**: nonzero udp over ipv6 checksums (sum)
- **wizard.tcp_hits**: tcp identifications (sum)
- **wizard.tcp_scans**: tcp payload scans (sum)
- **wizard.udp_hits**: udp identifications (sum)
- **wizard.udp_scans**: udp payload scans (sum)
- **wizard.user_hits**: user identifications (sum)
- **wizard.user_scans**: user payload scans (sum)

20.6 Generators

- **105**: back_orifice
- **106**: rpc_decode
- **112**: arp_spoof
- **116**: arp
- **116**: auth
- **116**: ciscometadata
- **116**: decode
- **116**: eapol
- **116**: erspan2
- **116**: erspan3
- **116**: esp
- **116**: eth
- **116**: fabricpath

- **116**: gre
- **116**: gtp
- **116**: icmp4
- **116**: icmp6
- **116**: igmp
- **116**: ipv4
- **116**: ipv6
- **116**: llc
- **116**: mpls
- **116**: pbb
- **116**: pgm
- **116**: pppoe
- **116**: tcp
- **116**: token_ring
- **116**: udp
- **116**: vlan
- **116**: wlan
- **119**: http_inspect
- **122**: port_scan
- **123**: stream_ip
- **124**: smtp
- **125**: ftp_server
- **126**: telnet
- **128**: ssh
- **129**: stream_tcp
- **131**: dns
- **133**: dce_http_proxy
- **133**: dce_http_server
- **133**: dce_smb
- **133**: dce_tcp
- **133**: dce_udp
- **134**: latency
- **135**: stream
- **136**: reputation
- **137**: ssl

- **140**: sip
- **141**: imap
- **142**: pop
- **143**: gtp_inspect
- **144**: modbus
- **145**: dnp3
- **146**: file_id
- **175**: domain_filter
- **219**: http2_inspect
- **256**: dpx

20.7 Builtin Rules

- **105:1** (back_orifice) BO traffic detected
- **105:2** (back_orifice) BO client traffic detected
- **105:3** (back_orifice) BO server traffic detected
- **105:4** (back_orifice) BO Snort buffer attack
- **106:1** (rpc_decode) fragmented RPC records
- **106:2** (rpc_decode) multiple RPC records
- **106:3** (rpc_decode) large RPC record fragment
- **106:4** (rpc_decode) incomplete RPC segment
- **106:5** (rpc_decode) zero-length RPC fragment
- **112:1** (arp_spoof) unicast ARP request
- **112:2** (arp_spoof) ethernet/ARP mismatch request for source
- **112:3** (arp_spoof) ethernet/ARP mismatch request for destination
- **112:4** (arp_spoof) attempted ARP cache overwrite attack
- **116:1** (ipv4) not IPv4 datagram
- **116:2** (ipv4) IPv4 header length < minimum
- **116:3** (ipv4) IPv4 datagram length < header field
- **116:4** (ipv4) IPv4 options found with bad lengths
- **116:5** (ipv4) truncated IPv4 options
- **116:6** (ipv4) IPv4 datagram length > captured length
- **116:45** (tcp) TCP packet length is smaller than 20 bytes
- **116:46** (tcp) TCP data offset is less than 5
- **116:47** (tcp) TCP header length exceeds packet length
- **116:54** (tcp) TCP options found with bad lengths

- **116:55** (tcp) truncated TCP options
- **116:56** (tcp) T/TCP detected
- **116:57** (tcp) obsolete TCP options found
- **116:58** (tcp) experimental TCP options found
- **116:59** (tcp) TCP window scale option found with length > 14
- **116:95** (udp) truncated UDP header
- **116:96** (udp) invalid UDP header, length field < 8
- **116:97** (udp) short UDP packet, length field > payload length
- **116:98** (udp) long UDP packet, length field < payload length
- **116:105** (icmp4) ICMP header truncated
- **116:106** (icmp4) ICMP timestamp header truncated
- **116:107** (icmp4) ICMP address header truncated
- **116:109** (arp) truncated ARP
- **116:110** (eapol) truncated EAP header
- **116:111** (eapol) EAP key truncated
- **116:112** (eapol) EAP header truncated
- **116:120** (pppoe) bad PPPOE frame detected
- **116:130** (vlan) bad VLAN frame
- **116:131** (llc) bad LLC header
- **116:132** (llc) bad extra LLC info
- **116:133** (wlan) bad 802.11 LLC header
- **116:134** (wlan) bad 802.11 extra LLC info
- **116:140** (token_ring) bad Token Ring header
- **116:141** (token_ring) bad Token Ring ETHLLC header
- **116:142** (token_ring) bad Token Ring MRLEN header
- **116:143** (token_ring) bad Token Ring MR header
- **116:150** (decode) loopback IP
- **116:151** (decode) same src/dst IP
- **116:160** (gre) GRE header length > payload length
- **116:161** (gre) multiple encapsulations in packet
- **116:162** (gre) invalid GRE version
- **116:163** (gre) invalid GRE header
- **116:164** (gre) invalid GRE v.1 PPTP header
- **116:165** (gre) GRE trans header length > payload length
- **116:170** (mpls) bad MPLS frame

- **116:171** (mpls) MPLS label 0 appears in non-bottom header
- **116:172** (mpls) MPLS label 1 appears in bottom header
- **116:173** (mpls) MPLS label 2 appears in non-bottom header
- **116:174** (mpls) MPLS label 3 appears in header
- **116:175** (mpls) MPLS label 4, 5,.. or 15 appears in header
- **116:176** (mpls) too many MPLS headers
- **116:250** (icmp4) ICMP original IP header truncated
- **116:251** (icmp4) ICMP version and original IP header versions differ
- **116:252** (icmp4) ICMP original datagram length < original IP header length
- **116:253** (icmp4) ICMP original IP payload < 64 bits
- **116:254** (icmp4) ICMP original IP payload > 576 bytes
- **116:255** (icmp4) ICMP original IP fragmented and offset not 0
- **116:270** (ipv6) IPv6 packet below TTL limit
- **116:271** (ipv6) IPv6 header claims to not be IPv6
- **116:272** (ipv6) IPv6 truncated extension header
- **116:273** (ipv6) IPv6 truncated header
- **116:274** (ipv6) IPv6 datagram length < header field
- **116:275** (ipv6) IPv6 datagram length > captured length
- **116:276** (ipv6) IPv6 packet with destination address ::0
- **116:277** (ipv6) IPv6 packet with multicast source address
- **116:278** (ipv6) IPv6 packet with reserved multicast destination address
- **116:279** (ipv6) IPv6 header includes an undefined option type
- **116:280** (ipv6) IPv6 address includes an unassigned multicast scope value
- **116:281** (ipv6) IPv6 header includes an invalid value for the *next header* field
- **116:282** (ipv6) IPv6 header includes a routing extension header followed by a hop-by-hop header
- **116:283** (ipv6) IPv6 header includes two routing extension headers
- **116:285** (icmp6) ICMPv6 packet of type 2 (message too big) with MTU field < 1280
- **116:286** (icmp6) ICMPv6 packet of type 1 (destination unreachable) with non-RFC 2463 code
- **116:287** (icmp6) ICMPv6 router solicitation packet with a code not equal to 0
- **116:288** (icmp6) ICMPv6 router advertisement packet with a code not equal to 0
- **116:289** (icmp6) ICMPv6 router solicitation packet with the reserved field not equal to 0
- **116:290** (icmp6) ICMPv6 router advertisement packet with the reachable time field set > 1 hour
- **116:291** (ipv6) IPV6 tunneled over IPv4, IPv6 header truncated, possible Linux kernel attack
- **116:292** (ipv6) IPv6 header has destination options followed by a routing header
- **116:293** (decode) two or more IP (v4 and/or v6) encapsulation layers present

- **116:294** (esp) truncated encapsulated security payload header
- **116:295** (ipv6) IPv6 header includes an option which is too big for the containing header
- **116:296** (ipv6) IPv6 packet includes out-of-order extension headers
- **116:297** (gtp) two or more GTP encapsulation layers present
- **116:298** (gtp) GTP header length is invalid
- **116:400** (tcp) XMAS attack detected
- **116:401** (tcp) Nmap XMAS attack detected
- **116:402** (tcp) DOS NAPTHA vulnerability detected
- **116:403** (tcp) SYN to multicast address
- **116:404** (ipv4) IPv4 packet with zero TTL
- **116:405** (ipv4) IPv4 packet with bad frag bits (both MF and DF set)
- **116:406** (udp) invalid IPv6 UDP packet, checksum zero
- **116:407** (ipv4) IPv4 packet frag offset + length exceed maximum
- **116:408** (ipv4) IPv4 packet from *current net* source address
- **116:409** (ipv4) IPv4 packet to *current net* dest address
- **116:410** (ipv4) IPv4 packet from multicast source address
- **116:411** (ipv4) IPv4 packet from reserved source address
- **116:412** (ipv4) IPv4 packet to reserved dest address
- **116:413** (ipv4) IPv4 packet from broadcast source address
- **116:414** (ipv4) IPv4 packet to broadcast dest address
- **116:415** (icmp4) ICMP4 packet to multicast dest address
- **116:416** (icmp4) ICMP4 packet to broadcast dest address
- **116:418** (icmp4) ICMP4 type other
- **116:419** (tcp) TCP urgent pointer exceeds payload length or no payload
- **116:420** (tcp) TCP SYN with FIN
- **116:421** (tcp) TCP SYN with RST
- **116:422** (tcp) TCP PDU missing ack for established session
- **116:423** (tcp) TCP has no SYN, ACK, or RST
- **116:424** (eth) truncated ethernet header
- **116:424** (pbb) truncated ethernet header
- **116:425** (ipv4) truncated IPv4 header
- **116:426** (icmp4) truncated ICMP4 header
- **116:427** (icmp6) truncated ICMPv6 header
- **116:428** (ipv4) IPv4 packet below TTL limit
- **116:429** (ipv6) IPv6 packet has zero hop limit

- **116:430** (ipv4) IPv4 packet both DF and offset set
- **116:431** (icmp6) ICMPv6 type not decoded
- **116:432** (icmp6) ICMPv6 packet to multicast address
- **116:433** (tcp) DDOS shaft SYN flood
- **116:434** (icmp4) ICMP ping Nmap
- **116:435** (icmp4) ICMP icmpenum v1.1.1
- **116:436** (icmp4) ICMP redirect host
- **116:437** (icmp4) ICMP redirect net
- **116:438** (icmp4) ICMP traceroute ipopts
- **116:439** (icmp4) ICMP source quench
- **116:440** (icmp4) broadscan smurf scanner
- **116:441** (icmp4) ICMP destination unreachable communication administratively prohibited
- **116:442** (icmp4) ICMP destination unreachable communication with destination host is administratively prohibited
- **116:443** (icmp4) ICMP destination unreachable communication with destination network is administratively prohibited
- **116:444** (ipv4) IPv4 option set
- **116:445** (udp) large UDP packet (> 4000 bytes)
- **116:446** (tcp) TCP port 0 traffic
- **116:447** (udp) UDP port 0 traffic
- **116:448** (ipv4) IPv4 reserved bit set
- **116:449** (decode) unassigned/reserved IP protocol
- **116:450** (decode) bad IP protocol
- **116:451** (icmp4) ICMP path MTU denial of service attempt
- **116:452** (icmp4) Linux ICMP header DOS attempt
- **116:453** (ipv6) ISATAP-addressed IPv6 traffic spoofing attempt
- **116:454** (pgm) PGM nak list overflow attempt
- **116:455** (igmp) DOS IGMP IP options validation attempt
- **116:456** (ipv6) too many IPv6 extension headers
- **116:457** (icmp6) ICMPv6 packet of type 1 (destination unreachable) with non-RFC 4443 code
- **116:458** (ipv6) bogus fragmentation packet, possible BSD attack
- **116:459** (decode) fragment with zero length
- **116:460** (icmp6) ICMPv6 node info query/response packet with a code greater than 2
- **116:461** (ipv6) IPv6 routing type 0 extension header
- **116:462** (erspan2) ERSpan header version mismatch
- **116:463** (erspan2) captured length < ERSpan type2 header length
- **116:464** (erspan3) captured < ERSpan type3 header length

- **116:465** (auth) truncated authentication header
- **116:466** (auth) bad authentication header length
- **116:467** (fabricpath) truncated FabricPath header
- **116:468** (ciscometadata) truncated Cisco Metadata header
- **116:469** (ciscometadata) invalid Cisco Metadata option length
- **116:470** (ciscometadata) invalid Cisco Metadata option type
- **116:471** (ciscometadata) invalid Cisco Metadata SGT
- **116:472** (decode) too many protocols present
- **116:473** (decode) ether type out of range
- **116:474** (icmp6) ICMPv6 not encapsulated in IPv6
- **116:475** (ipv6) IPv6 mobility header includes an invalid value for the *payload protocol* field
- **119:1** (http_inspect) ascii encoding
- **119:2** (http_inspect) double decoding attack
- **119:3** (http_inspect) u encoding
- **119:4** (http_inspect) bare byte unicode encoding
- **119:5** (http_inspect) obsolete event—deleted
- **119:6** (http_inspect) UTF-8 encoding
- **119:7** (http_inspect) unicode map code point encoding in URI
- **119:8** (http_inspect) multi_slash encoding
- **119:9** (http_inspect) backslash used in URI path
- **119:10** (http_inspect) self directory traversal
- **119:11** (http_inspect) directory traversal
- **119:12** (http_inspect) apache whitespace (tab)
- **119:13** (http_inspect) HTTP header line terminated by LF without a CR
- **119:14** (http_inspect) non-RFC defined char
- **119:15** (http_inspect) oversize request-uri directory
- **119:16** (http_inspect) oversize chunk encoding
- **119:17** (http_inspect) unauthorized proxy use detected
- **119:18** (http_inspect) webroot directory traversal
- **119:19** (http_inspect) long header
- **119:20** (http_inspect) max header fields
- **119:21** (http_inspect) multiple content length
- **119:22** (http_inspect) obsolete event—deleted
- **119:23** (http_inspect) invalid IP in true-client-IP/XFF header
- **119:24** (http_inspect) multiple host hdrs detected

- **119:25** (http_inspect) hostname exceeds 255 characters
- **119:26** (http_inspect) too much whitespace in header (not implemented yet)
- **119:27** (http_inspect) client consecutive small chunk sizes
- **119:28** (http_inspect) POST or PUT w/o content-length or chunks
- **119:29** (http_inspect) multiple true ips in a session
- **119:30** (http_inspect) both true-client-IP and XFF hdrs present
- **119:31** (http_inspect) unknown method
- **119:32** (http_inspect) simple request
- **119:33** (http_inspect) unescaped space in HTTP URI
- **119:34** (http_inspect) too many pipelined requests
- **119:101** (http_inspect) anomalous http server on undefined HTTP port
- **119:102** (http_inspect) invalid status code in HTTP response
- **119:103** (http_inspect) unused event number—should not appear
- **119:104** (http_inspect) HTTP response has UTF charset that failed to normalize
- **119:105** (http_inspect) HTTP response has UTF-7 charset
- **119:106** (http_inspect) HTTP response gzip decompression failed
- **119:107** (http_inspect) server consecutive small chunk sizes
- **119:108** (http_inspect) unused event number—should not appear
- **119:109** (http_inspect) javascript obfuscation levels exceeds 1
- **119:110** (http_inspect) javascript whitespaces exceeds max allowed
- **119:111** (http_inspect) multiple encodings within javascript obfuscated data
- **119:112** (http_inspect) SWF file zlib decompression failure
- **119:113** (http_inspect) SWF file LZMA decompression failure
- **119:114** (http_inspect) PDF file deflate decompression failure
- **119:115** (http_inspect) PDF file unsupported compression type
- **119:116** (http_inspect) PDF file cascaded compression
- **119:117** (http_inspect) PDF file parse failure
- **119:201** (http_inspect) not HTTP traffic
- **119:202** (http_inspect) chunk length has excessive leading zeros
- **119:203** (http_inspect) white space before or between messages
- **119:204** (http_inspect) request message without URI
- **119:205** (http_inspect) control character in reason phrase
- **119:206** (http_inspect) illegal extra whitespace in start line
- **119:207** (http_inspect) corrupted HTTP version
- **119:208** (http_inspect) unknown HTTP version

- **119:209** (http_inspect) format error in HTTP header
- **119:210** (http_inspect) chunk header options present
- **119:211** (http_inspect) URI badly formatted
- **119:212** (http_inspect) unrecognized type of percent encoding in URI
- **119:213** (http_inspect) HTTP chunk misformatted
- **119:214** (http_inspect) white space adjacent to chunk length
- **119:215** (http_inspect) white space within header name
- **119:216** (http_inspect) excessive gzip compression
- **119:217** (http_inspect) gzip decompression failed
- **119:218** (http_inspect) HTTP 0.9 requested followed by another request
- **119:219** (http_inspect) HTTP 0.9 request following a normal request
- **119:220** (http_inspect) message has both Content-Length and Transfer-Encoding
- **119:221** (http_inspect) status code implying no body combined with Transfer-Encoding or nonzero Content-Length
- **119:222** (http_inspect) Transfer-Encoding not ending with chunked
- **119:223** (http_inspect) Transfer-Encoding with encodings before chunked
- **119:224** (http_inspect) misformatted HTTP traffic
- **119:225** (http_inspect) unsupported Content-Encoding used
- **119:226** (http_inspect) unknown Content-Encoding used
- **119:227** (http_inspect) multiple Content-Encodings applied
- **119:228** (http_inspect) server response before client request
- **119:229** (http_inspect) PDF/SWF decompression of server response too big
- **119:230** (http_inspect) nonprinting character in HTTP message header name
- **119:231** (http_inspect) bad Content-Length value in HTTP header
- **119:232** (http_inspect) HTTP header line wrapped
- **119:233** (http_inspect) HTTP header line terminated by CR without a LF
- **119:234** (http_inspect) chunk terminated by nonstandard separator
- **119:235** (http_inspect) chunk length terminated by LF without CR
- **119:236** (http_inspect) more than one response with 100 status code
- **119:237** (http_inspect) 100 status code not in response to Expect header
- **119:238** (http_inspect) 1XX status code other than 100 or 101
- **119:239** (http_inspect) Expect header sent without a message body
- **119:240** (http_inspect) HTTP 1.0 message with Transfer-Encoding header
- **119:241** (http_inspect) Content-Transfer-Encoding used as HTTP header
- **119:242** (http_inspect) illegal field in chunked message trailers
- **119:243** (http_inspect) header field inappropriately appears twice or has two values

- **119:244** (http_inspect) invalid value chunked in Content-Encoding header
- **119:245** (http_inspect) 206 response sent to a request without a Range header
- **119:246** (http_inspect) *HTTP* in version field not all upper case
- **119:247** (http_inspect) white space embedded in critical header value
- **119:248** (http_inspect) gzip compressed data followed by unexpected non-gzip data
- **122:1** (port_scan) TCP portscan
- **122:2** (port_scan) TCP decoy portscan
- **122:3** (port_scan) TCP portsweep
- **122:4** (port_scan) TCP distributed portscan
- **122:5** (port_scan) TCP filtered portscan
- **122:6** (port_scan) TCP filtered decoy portscan
- **122:7** (port_scan) TCP filtered portsweep
- **122:8** (port_scan) TCP filtered distributed portscan
- **122:9** (port_scan) IP protocol scan
- **122:10** (port_scan) IP decoy protocol scan
- **122:11** (port_scan) IP protocol sweep
- **122:12** (port_scan) IP distributed protocol scan
- **122:13** (port_scan) IP filtered protocol scan
- **122:14** (port_scan) IP filtered decoy protocol scan
- **122:15** (port_scan) IP filtered protocol sweep
- **122:16** (port_scan) IP filtered distributed protocol scan
- **122:17** (port_scan) UDP portscan
- **122:18** (port_scan) UDP decoy portscan
- **122:19** (port_scan) UDP portsweep
- **122:20** (port_scan) UDP distributed portscan
- **122:21** (port_scan) UDP filtered portscan
- **122:22** (port_scan) UDP filtered decoy portscan
- **122:23** (port_scan) UDP filtered portsweep
- **122:24** (port_scan) UDP filtered distributed portscan
- **122:25** (port_scan) ICMP sweep
- **122:26** (port_scan) ICMP filtered sweep
- **122:27** (port_scan) open port
- **123:1** (stream_ip) inconsistent IP options on fragmented packets
- **123:2** (stream_ip) teardrop attack
- **123:3** (stream_ip) short fragment, possible DOS attempt

- **123:4** (stream_ip) fragment packet ends after defragmented packet
- **123:5** (stream_ip) zero-byte fragment packet
- **123:6** (stream_ip) bad fragment size, packet size is negative
- **123:7** (stream_ip) bad fragment size, packet size is greater than 65536
- **123:8** (stream_ip) fragmentation overlap
- **123:11** (stream_ip) TTL value less than configured minimum, not using for reassembly
- **123:12** (stream_ip) excessive fragment overlap
- **123:13** (stream_ip) tiny fragment
- **124:1** (smtp) attempted command buffer overflow
- **124:2** (smtp) attempted data header buffer overflow
- **124:3** (smtp) attempted response buffer overflow
- **124:4** (smtp) attempted specific command buffer overflow
- **124:5** (smtp) unknown command
- **124:6** (smtp) illegal command
- **124:7** (smtp) attempted header name buffer overflow
- **124:8** (smtp) attempted X-Link2State command buffer overflow
- **124:10** (smtp) base64 decoding failed
- **124:11** (smtp) quoted-printable decoding failed
- **124:13** (smtp) Unix-to-Unix decoding failed
- **124:14** (smtp) Cyrus SASL authentication attack
- **124:15** (smtp) attempted authentication command buffer overflow
- **125:1** (ftp_server) TELNET cmd on FTP command channel
- **125:2** (ftp_server) invalid FTP command
- **125:3** (ftp_server) FTP command parameters were too long
- **125:4** (ftp_server) FTP command parameters were malformed
- **125:5** (ftp_server) FTP command parameters contained potential string format
- **125:6** (ftp_server) FTP response message was too long
- **125:7** (ftp_server) FTP traffic encrypted
- **125:8** (ftp_server) FTP bounce attempt
- **125:9** (ftp_server) evasive (incomplete) TELNET cmd on FTP command channel
- **126:1** (telnet) consecutive Telnet AYT commands beyond threshold
- **126:2** (telnet) Telnet traffic encrypted
- **126:3** (telnet) Telnet subnegotiation begin command without subnegotiation end
- **128:1** (ssh) challenge-response overflow exploit
- **128:2** (ssh) SSH1 CRC32 exploit

- **128:3** (ssh) server version string overflow
- **128:5** (ssh) bad message direction
- **128:6** (ssh) payload size incorrect for the given payload
- **128:7** (ssh) failed to detect SSH version string
- **129:1** (stream_tcp) SYN on established session
- **129:2** (stream_tcp) data on SYN packet
- **129:3** (stream_tcp) data sent on stream not accepting data
- **129:4** (stream_tcp) TCP timestamp is outside of PAWS window
- **129:5** (stream_tcp) bad segment, adjusted size $\Leftarrow$ 0 (deprecated)
- **129:6** (stream_tcp) window size (after scaling) larger than policy allows
- **129:7** (stream_tcp) limit on number of overlapping TCP packets reached
- **129:8** (stream_tcp) data sent on stream after TCP reset sent
- **129:9** (stream_tcp) TCP client possibly hijacked, different ethernet address
- **129:10** (stream_tcp) TCP server possibly hijacked, different ethernet address
- **129:11** (stream_tcp) TCP data with no TCP flags set
- **129:12** (stream_tcp) consecutive TCP small segments exceeding threshold
- **129:13** (stream_tcp) 4-way handshake detected
- **129:14** (stream_tcp) TCP timestamp is missing
- **129:15** (stream_tcp) reset outside window
- **129:16** (stream_tcp) FIN number is greater than prior FIN
- **129:17** (stream_tcp) ACK number is greater than prior FIN
- **129:18** (stream_tcp) data sent on stream after TCP reset received
- **129:19** (stream_tcp) TCP window closed before receiving data
- **129:20** (stream_tcp) TCP session without 3-way handshake
- **131:1** (dns) obsolete DNS RR types
- **131:2** (dns) experimental DNS RR types
- **131:3** (dns) DNS client rdata txt overflow
- **133:2** (dce_smb) SMB - bad NetBIOS session service session type
- **133:3** (dce_smb) SMB - bad SMB message type
- **133:4** (dce_smb) SMB - bad SMB Id (not \xffSMB for SMB1 or not \xfeSMB for SMB2)
- **133:5** (dce_smb) SMB - bad word count or structure size
- **133:6** (dce_smb) SMB - bad byte count
- **133:7** (dce_smb) SMB - bad format type
- **133:8** (dce_smb) SMB - bad offset
- **133:9** (dce_smb) SMB - zero total data count

- **133:10** (dce_smb) SMB - NetBIOS data length less than SMB header length
- **133:12** (dce_smb) SMB - remaining NetBIOS data length less than command byte count
- **133:13** (dce_smb) SMB - remaining NetBIOS data length less than command data size
- **133:14** (dce_smb) SMB - remaining total data count less than this command data size
- **133:15** (dce_smb) SMB - total data sent (STDu64) greater than command total data expected
- **133:16** (dce_smb) SMB - byte count less than command data size (STDu64)
- **133:17** (dce_smb) SMB - invalid command data size for byte count
- **133:18** (dce_smb) SMB - excessive tree connect requests with pending tree connect responses
- **133:19** (dce_smb) SMB - excessive read requests with pending read responses
- **133:20** (dce_smb) SMB - excessive command chaining
- **133:21** (dce_smb) SMB - multiple chained tree connect requests
- **133:22** (dce_smb) SMB - multiple chained tree connect requests
- **133:23** (dce_smb) SMB - chained/compounded login followed by logoff
- **133:24** (dce_smb) SMB - chained/compounded tree connect followed by tree disconnect
- **133:25** (dce_smb) SMB - chained/compounded open pipe followed by close pipe
- **133:26** (dce_smb) SMB - invalid share access
- **133:27** (dce_tcp) connection oriented DCE/RPC - invalid major version
- **133:28** (dce_tcp) connection oriented DCE/RPC - invalid minor version
- **133:29** (dce_tcp) connection-oriented DCE/RPC - invalid PDU type
- **133:30** (dce_tcp) connection-oriented DCE/RPC - fragment length less than header size
- **133:32** (dce_tcp) connection-oriented DCE/RPC - no context items specified
- **133:33** (dce_tcp) connection-oriented DCE/RPC -no transfer syntaxes specified
- **133:34** (dce_tcp) connection-oriented DCE/RPC - fragment length on non-last fragment less than maximum negotiated fragment transmit size for client
- **133:35** (dce_tcp) connection-oriented DCE/RPC - fragment length greater than maximum negotiated fragment transmit size
- **133:36** (dce_tcp) connection-oriented DCE/RPC - alter context byte order different from bind
- **133:37** (dce_tcp) connection-oriented DCE/RPC - call id of non first/last fragment different from call id established for fragmented request
- **133:38** (dce_tcp) connection-oriented DCE/RPC - opnum of non first/last fragment different from opnum established for fragmented request
- **133:39** (dce_tcp) connection-oriented DCE/RPC - context id of non first/last fragment different from context id established for fragmented request
- **133:40** (dce_udp) connection-less DCE/RPC - invalid major version
- **133:41** (dce_udp) connection-less DCE/RPC - invalid PDU type
- **133:42** (dce_udp) connection-less DCE/RPC - data length less than header size
- **133:43** (dce_udp) connection-less DCE/RPC - bad sequence number

- **133:44** (dce_smb) SMB - invalid SMB version 1 seen
- **133:45** (dce_smb) SMB - invalid SMB version 2 seen
- **133:46** (dce_smb) SMB - invalid user, tree connect, file binding
- **133:47** (dce_smb) SMB - excessive command compounding
- **133:48** (dce_smb) SMB - zero data count
- **133:50** (dce_smb) SMB - maximum number of outstanding requests exceeded
- **133:51** (dce_smb) SMB - outstanding requests with same MID
- **133:52** (dce_smb) SMB - deprecated dialect negotiated
- **133:53** (dce_smb) SMB - deprecated command used
- **133:54** (dce_smb) SMB - unusual command used
- **133:55** (dce_smb) SMB - invalid setup count for command
- **133:56** (dce_smb) SMB - client attempted multiple dialect negotiations on session
- **133:57** (dce_smb) SMB - client attempted to create or set a file's attributes to readonly/hidden/system
- **133:58** (dce_smb) SMB - file offset provided is greater than file size specified
- **133:59** (dce_smb) SMB - next command specified in SMB2 header is beyond payload boundary
- **134:1** (latency) rule tree suspended due to latency
- **134:2** (latency) rule tree re-enabled after suspend timeout
- **134:3** (latency) packet fastpathed due to latency
- **135:1** (stream) TCP SYN received
- **135:2** (stream) TCP session established
- **135:3** (stream) TCP session cleared
- **136:1** (reputation) packets blacklisted
- **136:2** (reputation) packets whitelisted
- **136:3** (reputation) packets monitored
- **137:1** (ssl) invalid client HELLO after server HELLO detected
- **137:2** (ssl) invalid server HELLO without client HELLO detected
- **137:3** (ssl) heartbeat read overrun attempt detected
- **137:4** (ssl) large heartbeat response detected
- **140:2** (sip) empty request URI
- **140:3** (sip) URI is too long
- **140:4** (sip) empty call-Id
- **140:5** (sip) Call-Id is too long
- **140:6** (sip) CSeq number is too large or negative
- **140:7** (sip) request name in CSeq is too long
- **140:8** (sip) empty From header

- **140:9** (sip) From header is too long
- **140:10** (sip) empty To header
- **140:11** (sip) To header is too long
- **140:12** (sip) empty Via header
- **140:13** (sip) Via header is too long
- **140:14** (sip) empty Contact
- **140:15** (sip) contact is too long
- **140:16** (sip) content length is too large or negative
- **140:17** (sip) multiple SIP messages in a packet
- **140:18** (sip) content length mismatch
- **140:19** (sip) request name is invalid
- **140:20** (sip) Invite replay attack
- **140:21** (sip) illegal session information modification
- **140:22** (sip) response status code is not a 3 digit number
- **140:23** (sip) empty Content-type header
- **140:24** (sip) SIP version is invalid
- **140:25** (sip) mismatch in METHOD of request and the CSEQ header
- **140:26** (sip) method is unknown
- **140:27** (sip) maximum dialogs within a session reached
- **141:1** (imap) unknown IMAP3 command
- **141:2** (imap) unknown IMAP3 response
- **141:4** (imap) base64 decoding failed
- **141:5** (imap) quoted-printable decoding failed
- **141:7** (imap) Unix-to-Unix decoding failed
- **142:1** (pop) unknown POP3 command
- **142:2** (pop) unknown POP3 response
- **142:4** (pop) base64 decoding failed
- **142:5** (pop) quoted-printable decoding failed
- **142:7** (pop) Unix-to-Unix decoding failed
- **143:1** (gtp_inspect) message length is invalid
- **143:2** (gtp_inspect) information element length is invalid
- **143:3** (gtp_inspect) information elements are out of order
- **144:1** (modbus) length in Modbus MBAP header does not match the length needed for the given function
- **144:2** (modbus) Modbus protocol ID is non-zero
- **144:3** (modbus) reserved Modbus function code in use

- **145:1** (dnp3) DNP3 link-layer frame contains bad CRC
- **145:2** (dnp3) DNP3 link-layer frame was dropped
- **145:3** (dnp3) DNP3 transport-layer segment was dropped during reassembly
- **145:4** (dnp3) DNP3 reassembly buffer was cleared without reassembling a complete message
- **145:5** (dnp3) DNP3 link-layer frame uses a reserved address
- **145:6** (dnp3) DNP3 application-layer fragment uses a reserved function code
- **175:1** (domain_filter) configured domain detected
- **256:1** (dpx) too much data sent to port

20.8 Command Set

- **appid.enable_debug**(proto, src_ip, src_port, dst_ip, dst_port): enable appid debugging
- **appid.disable_debug**(): disable appid debugging
- **packet_capture.enable**(filter): dump raw packets
- **packet_capture.disable**(): stop packet dump
- **packet_tracer.enable**(proto, src_ip, src_port, dst_ip, dst_port): enable packet tracer debugging
- **packet_tracer.disable**(): disable packet tracer
- **snort.show_plugins**(): show available plugins
- **snort.delete_inspector**(inspector): delete an inspector from the default policy
- **snort.dump_stats**(): show summary statistics
- **snort.rotate_stats**(): roll perfmonitor log files
- **snort.reload_config**(filename): load new configuration
- **snort.reload_policy**(filename): reload part or all of the default policy
- **snort.reload_module**(module): reload module
- **snort.reload_daq**(): reload daq module
- **snort.reload_hosts**(filename): load a new hosts table
- **snort.pause**(): suspend packet processing
- **snort.resume**(): continue packet processing
- **snort.detach**(): exit shell w/o shutdown
- **snort.quit**(): shutdown and dump-stats
- **snort.help**(): this output

20.9 Signals

Important
Signal numbers are for the system that generated this documentation and are not applicable elsewhere.

- **term**(15): shutdown normally
- **int**(2): shutdown normally
- **quit**(3): shutdown as if started with --dirty-pig
- **stats**(10): dump stats to stdout
- **rotate**(12): rotate stats files
- **reload**(1): reload config file
- **hosts**(23): reload hosts file

20.10 Configuration Changes

```
change ->  dynamicdetection ==> 'snort.--plugin_path=<path>'
change ->  dynamicengine ==> 'snort.--plugin_path=<path>'
change ->  dynamicpreprocessor ==> 'snort.--plugin_path=<path>'
change ->  dynamicsidechannel ==> 'snort.--plugin_path=<path>'
change -> alertfile: 'config alertfile:' ==> 'alert_fast.file'
change -> alertfile: 'config alertfile:' ==> 'alert_full.file'
change -> attribute_table: 'STREAM_POLICY' ==> 'hosts: tcp_policy'
change -> attribute_table: 'filename <file_name>' ==> 'hosts[]'
change -> config ' addressspace_agnostic'  ==> ' packets. address_space_agnostic'
change -> config ' checksum_mode'  ==> ' network. checksum_eval'
change -> config ' daq'  ==> ' daq. type'
change -> config ' daq_dir'  ==> ' daq. dir'
change -> config ' daq_mode'  ==> ' daq. mode'
change -> config ' daq_var'  ==> ' daq. var'
change -> config ' detection_filter'  ==> ' alerts. detection_filter_memcap'
change -> config ' enable_deep_teredo_inspection'  ==> ' udp. deep_teredo_inspection'
change -> config ' event_filter'  ==> ' alerts. event_filter_memcap'
change -> config ' max_attribute_hosts'  ==> ' attribute_table. max_hosts'
change -> config ' max_attribute_services_per_host'  ==> ' attribute_table.  ↩
    max_services_per_host'
change -> config ' nopcre'  ==> ' detection. pcre_enable'
change -> config ' pkt_count'  ==> ' packets. limit'
change -> config ' rate_filter'  ==> ' alerts. rate_filter_memcap'
change -> config ' react'  ==> ' react. page'
change -> config ' threshold'  ==> ' alerts. event_filter_memcap'
change -> csv: 'dgmlen' ==> 'dgm_len'
change -> csv: 'dst' ==> 'dst_addr'
change -> csv: 'dstport' ==> 'dst_port'
change -> csv: 'ethdst' ==> 'eth_dst'
change -> csv: 'ethlen' ==> 'eth_len'
change -> csv: 'ethsrc' ==> 'eth_src'
change -> csv: 'ethtype' ==> 'eth_type'
change -> csv: 'icmpcode' ==> 'icmp_code'
change -> csv: 'icmpid' ==> 'icmp_id'
change -> csv: 'icmpseq' ==> 'icmp_seq'
change -> csv: 'icmptype' ==> 'icmp_type'
```

```
change -> csv: 'iplen' ==> 'ip_len'
change -> csv: 'sig_generator' ==> 'gid'
change -> csv: 'sig_id' ==> 'sid'
change -> csv: 'sig_rev' ==> 'rev'
change -> csv: 'src' ==> 'src_addr'
change -> csv: 'srcport' ==> 'src_port'
change -> csv: 'tcpack' ==> 'tcp_ack'
change -> csv: 'tcpflags' ==> 'tcp_flags'
change -> csv: 'tcplen' ==> 'tcp_len'
change -> csv: 'tcpseq' ==> 'tcp_seq'
change -> csv: 'tcpwindow' ==> 'tcp_win'
change -> csv: 'udplength' ==> 'udp_len'
change -> detection: 'ac' ==> 'ac_full_q'
change -> detection: 'ac-banded' ==> 'ac_banded'
change -> detection: 'ac-bnfa' ==> 'ac_bnfa_q'
change -> detection: 'ac-bnfa-nq' ==> 'ac_bnfa'
change -> detection: 'ac-bnfa-q' ==> 'ac_bnfa_q'
change -> detection: 'ac-nq' ==> 'ac_full'
change -> detection: 'ac-q' ==> 'ac_full_q'
change -> detection: 'ac-sparsebands' ==> 'ac_sparse_bands'
change -> detection: 'ac-split' ==> 'ac_full_q'
change -> detection: 'ac-split' ==> 'split_any_any'
change -> detection: 'ac-std' ==> 'ac_std'
change -> detection: 'acs' ==> 'ac_sparse'
change -> detection: 'bleedover-port-limit' ==> 'bleedover_port_limit'
change -> detection: 'intel-cpm' ==> 'intel_cpm'
change -> detection: 'lowmem' ==> 'lowmem_q'
change -> detection: 'lowmem-nq' ==> 'lowmem'
change -> detection: 'lowmem-q' ==> 'lowmem_q'
change -> detection: 'max-pattern-len' ==> 'max_pattern_len'
change -> detection: 'search-method' ==> 'search_method'
change -> detection: 'search-optimize' ==> 'search_optimize'
change -> detection: 'split-any-any' ==> 'split_any_any'
change -> dns: 'ports' ==> 'bindings'
change -> event_filter: 'gen_id' ==> 'gid'
change -> event_filter: 'sig_id' ==> 'sid'
change -> event_filter: 'threshold' ==> 'event_filter'
change -> file: 'config file: file_block_timeout' ==> 'block_timeout'
change -> file: 'config file: file_type_depth' ==> 'type_depth'
change -> file: 'config file: signature' ==> 'enable_signature'
change -> file: 'config file: type_id' ==> 'enable_type'
change -> frag3_engine: 'min_fragment_length' ==> 'min_frag_length'
change -> frag3_engine: 'overlap_limit' ==> 'max_overlaps'
change -> frag3_engine: 'policy bsd-right' ==> 'policy = bsd_right'
change -> frag3_engine: 'timeout' ==> 'session_timeout'
change -> ftp_telnet_protocol: 'alt_max_param_len' ==> 'cmd_validity'
change -> ftp_telnet_protocol: 'data_chan' ==> 'ignore_data_chan'
change -> ftp_telnet_protocol: 'ports' ==> 'bindings'
change -> gtp: 'ports' ==> 'gtp_ports'
change -> http_inspect: 'http_inspect' ==> 'http_global'
change -> http_inspect_server: 'apache_whitespace' ==> 'profile.apache_whitespace'
change -> http_inspect_server: 'ascii' ==> 'profile.ascii'
change -> http_inspect_server: 'bare_byte' ==> 'profile.bare_byte'
change -> http_inspect_server: 'chunk_length' ==> 'profile.chunk_length'
change -> http_inspect_server: 'client_flow_depth' ==> 'profile.client_flow_depth'
change -> http_inspect_server: 'directory' ==> 'profile.directory'
change -> http_inspect_server: 'double_decode' ==> 'profile.double_decode'
change -> http_inspect_server: 'enable_cookie' ==> 'enable_cookies'
change -> http_inspect_server: 'flow_depth' ==> 'server_flow_depth'
change -> http_inspect_server: 'http_inspect_server' ==> 'http_inspect'
change -> http_inspect_server: 'iis_backslash' ==> 'profile.iis_backslash'
change -> http_inspect_server: 'iis_delimiter' ==> 'profile.iis_delimiter'
```

```
change -> http_inspect_server: 'iis_unicode' ==> 'profile.iis_unicode'
change -> http_inspect_server: 'max_header_length' ==> 'profile.max_header_length'
change -> http_inspect_server: 'max_headers' ==> 'profile.max_headers'
change -> http_inspect_server: 'max_spaces' ==> 'profile.max_spaces'
change -> http_inspect_server: 'multi_slash' ==> 'profile.multi_slash'
change -> http_inspect_server: 'non_rfc_char' ==> 'non_rfc_chars'
change -> http_inspect_server: 'non_strict' ==> 'profile.non_strict'
change -> http_inspect_server: 'normalize_utf' ==> 'profile.normalize_utf'
change -> http_inspect_server: 'ports' ==> 'bindings'
change -> http_inspect_server: 'u_encode' ==> 'profile.u_encode'
change -> http_inspect_server: 'utf_8' ==> 'profile.utf_8'
change -> http_inspect_server: 'webroot' ==> 'profile.webroot'
change -> http_inspect_server: 'whitespace_chars' ==> 'profile.whitespace_chars'
change -> imap: 'ports' ==> 'bindings'
change -> paf_max: 'paf_max [0:63780]' ==> 'max_pdu [1460:63780]'
change -> perfmonitor: 'accumulate' ==> 'reset = false'
change -> perfmonitor: 'flow-file' ==> 'flow_file = true'
change -> perfmonitor: 'flow-ip' ==> 'flow_ip'
change -> perfmonitor: 'flow-ip-file' ==> 'flow_ip_file = true'
change -> perfmonitor: 'flow-ip-memcap' ==> 'flow_ip_memcap'
change -> perfmonitor: 'flow-ports' ==> 'flow_ports'
change -> perfmonitor: 'pktcnt' ==> 'packets'
change -> perfmonitor: 'snortfile' ==> 'file = true'
change -> perfmonitor: 'time' ==> 'seconds'
change -> policy_mode: 'inline_test' ==> 'inline-test'
change -> pop: 'ports' ==> 'bindings'
change -> ppm: 'max-pkt-time' ==> 'max_pkt_time'
change -> ppm: 'max-rule-time' ==> 'max_rule_time'
change -> ppm: 'pkt-log' ==> 'pkt_log'
change -> ppm: 'rule-log' ==> 'rule_log'
change -> ppm: 'suspend-timeout' ==> 'suspend_timeout'
change -> preprocessor 'normalize_ icmp4' ==> 'normalize. icmp4'
change -> preprocessor 'normalize_ icmp6' ==> 'normalize. icmp6'
change -> preprocessor 'normalize_ ip6' ==> 'normalize. ip6'
change -> profile: 'print' ==> 'count'
change -> rate_filter: 'gen_id' ==> 'gid'
change -> rate_filter: 'sig_id' ==> 'sid'
change -> rule_state: 'disabled' ==> 'enable'
change -> rule_state: 'enabled' ==> 'enable'
change -> sfportscan: 'proto' ==> 'protos'
change -> sfportscan: 'scan_type' ==> 'scan_types'
change -> sip: 'ports' ==> 'bindings'
change -> smtp: 'ports' ==> 'bindings'
change -> ssh: 'server_ports' ==> 'bindings'
change -> ssl: 'ports' ==> 'bindings'
change -> stream5_global: 'max_active_responses' ==> 'max_responses'
change -> stream5_global: 'max_icmp' ==> 'max_sessions'
change -> stream5_global: 'max_ip' ==> 'max_sessions'
change -> stream5_global: 'max_tcp' ==> 'max_sessions'
change -> stream5_global: 'max_udp' ==> 'max_sessions'
change -> stream5_global: 'min_response_seconds' ==> 'min_interval'
change -> stream5_global: 'prune_log_max' ==> 'histogram'
change -> stream5_global: 'tcp_cache_nominal_timeout' ==> 'pruning_timeout'
change -> stream5_global: 'tcp_cache_pruning_timeout' ==> 'idle_timeout'
change -> stream5_global: 'udp_cache_nominal_timeout' ==> 'idle_timeout'
change -> stream5_global: 'udp_cache_pruning_timeout' ==> 'pruning_timeout'
change -> stream5_ip: 'timeout' ==> 'session_timeout'
change -> stream5_tcp: 'bind_to' ==> 'bindings'
change -> stream5_tcp: 'dont_reassemble_async' ==> 'reassemble_async'
change -> stream5_tcp: 'max_queued_bytes' ==> 'queue_limit.max_bytes'
change -> stream5_tcp: 'max_queued_segs' ==> 'queue_limit.max_segments'
change -> stream5_tcp: 'policy hpux' ==> 'stream_tcp.policy = hpux11'
```

```
change -> stream5_tcp: 'timeout' ==> 'session_timeout'
change -> stream5_tcp: 'use_static_footprint_sizes' ==> 'footprint'
change -> stream5_udp: 'timeout' ==> 'session_timeout'
change -> suppress: 'gen_id' ==> 'gid'
change -> suppress: 'sig_id' ==> 'sid'
change -> syslog: 'log_alert' ==> 'level = alert'
change -> syslog: 'log_auth' ==> 'facility = auth'
change -> syslog: 'log_authpriv' ==> 'facility = authpriv'
change -> syslog: 'log_cons' ==> 'options = cons'
change -> syslog: 'log_crit' ==> 'level = crit'
change -> syslog: 'log_daemon' ==> 'facility = daemon'
change -> syslog: 'log_debug' ==> 'level = debug'
change -> syslog: 'log_emerg' ==> 'level = emerg'
change -> syslog: 'log_err' ==> 'level = err'
change -> syslog: 'log_info' ==> 'level = info'
change -> syslog: 'log_local0' ==> 'facility = local0'
change -> syslog: 'log_local1' ==> 'facility = local1'
change -> syslog: 'log_local2' ==> 'facility = local2'
change -> syslog: 'log_local3' ==> 'facility = local3'
change -> syslog: 'log_local4' ==> 'facility = local4'
change -> syslog: 'log_local5' ==> 'facility = local5'
change -> syslog: 'log_local6' ==> 'facility = local6'
change -> syslog: 'log_local7' ==> 'facility = local7'
change -> syslog: 'log_ndelay' ==> 'options = ndelay'
change -> syslog: 'log_notice' ==> 'level = notice'
change -> syslog: 'log_perror' ==> 'options = perror'
change -> syslog: 'log_pid' ==> 'options = pid'
change -> syslog: 'log_user' ==> 'facility = user'
change -> syslog: 'log_warning' ==> 'level = warning'
change -> threshold: 'ips_option: threshold' ==> 'event_filter'
change -> unified2: ' alert_unified2' ==> 'unified2'
change -> unified2: ' log_unified2' ==> 'unified2'
change -> unified2: ' unified2' ==> 'unified2'
deleted -> arpspoof: 'unicast'
deleted -> attribute_table: '<FRAG_POLICY>hpux</FRAG_POLICY>'
deleted -> attribute_table: '<FRAG_POLICY>irix</FRAG_POLICY>'
deleted -> attribute_table: '<FRAG_POLICY>old-linux</FRAG_POLICY>'
deleted -> attribute_table: '<FRAG_POLICY>unknown</FRAG_POLICY>'
deleted -> attribute_table: '<STREAM_POLICY>noack</STREAM_POLICY>'
deleted -> attribute_table: '<STREAM_POLICY>unknown</STREAM_POLICY>'
deleted -> config ' cs_dir'
deleted -> config ' disable_attribute_reload_thread'
deleted -> config ' disable_decode_alerts'
deleted -> config ' disable_decode_drops'
deleted -> config ' disable_ipopt_alerts'
deleted -> config ' disable_ipopt_drops'
deleted -> config ' disable_tcpopt_alerts'
deleted -> config ' disable_tcpopt_drops'
deleted -> config ' disable_tcpopt_experimental_alerts'
deleted -> config ' disable_tcpopt_experimental_drops'
deleted -> config ' disable_tcpopt_obsolete_alerts'
deleted -> config ' disable_tcpopt_obsolete_drops'
deleted -> config ' disable_tcpopt_ttcp_alerts'
deleted -> config ' disable_ttcp_alerts'
deleted -> config ' disable_ttcp_drops'
deleted -> config ' dump_dynamic_rules_path'
deleted -> config ' enable_decode_drops'
deleted -> config ' enable_decode_oversized_alerts'
deleted -> config ' enable_decode_oversized_drops'
deleted -> config ' enable_ipopt_drops'
deleted -> config ' enable_tcpopt_drops'
deleted -> config ' enable_tcpopt_experimental_drops'
```

```
deleted -> config ' enable_tcpopt_obsolete_drops'
deleted -> config ' enable_tcpopt_ttcp_drops'
deleted -> config ' enable_ttcp_drops'
deleted -> config ' flexresp2_attempts'
deleted -> config ' flexresp2_interface'
deleted -> config ' flexresp2_memcap'
deleted -> config ' flexresp2_rows'
deleted -> config ' flowbits_size'
deleted -> config ' include_vlan_in_alerts'
deleted -> config ' interface'
deleted -> config ' layer2resets'
deleted -> config ' policy_version'
deleted -> config ' so_rule_memcap'
deleted -> csv: '<filename> can no longer be specific'
deleted -> csv: 'default'
deleted -> csv: 'trheader'
deleted -> detection: 'mwm'
deleted -> dns: 'enable_experimental_types'
deleted -> dns: 'enable_obsolete_types'
deleted -> dns: 'enable_rdata_overflow'
deleted -> fast: '<filename> can no longer be specific'
deleted -> frag3_engine: 'detect_anomalies'
deleted -> frag3_global: 'disabled'
deleted -> ftp_telnet_protocol: 'detect_anomalies'
deleted -> full: '<filename> can no longer be specific'
deleted -> http_inspect: 'disabled'
deleted -> http_inspect_server: 'no_alerts'
deleted -> imap: 'disabled'
deleted -> imap: 'max_mime_mem'
deleted -> imap: 'memcap'
deleted -> perfmonitor: 'atexitonly'
deleted -> perfmonitor: 'atexitonly: base-stats'
deleted -> perfmonitor: 'atexitonly: events-stats'
deleted -> perfmonitor: 'atexitonly: flow-ip-stats'
deleted -> perfmonitor: 'atexitonly: flow-stats'
deleted -> pop: 'disabled'
deleted -> pop: 'max_mime_mem'
deleted -> pop: 'memcap'
deleted -> ppm: 'debug-pkts'
deleted -> react: 'block'
deleted -> react: 'warn'
deleted -> rpc_decode: 'alert_fragments'
deleted -> rpc_decode: 'no_alert_incomplete'
deleted -> rpc_decode: 'no_alert_large_fragments'
deleted -> rpc_decode: 'no_alert_multiple_requests'
deleted -> rule_state: 'action'
deleted -> sfportscan: 'detect_ack_scans'
deleted -> sfportscan: 'disabled'
deleted -> sfportscan: 'logfile'
deleted -> sip: 'disabled'
deleted -> smtp: 'alert_unknown_cmds'
deleted -> smtp: 'disabled'
deleted -> smtp: 'enable_mime_decoding'
deleted -> smtp: 'inspection_type'
deleted -> smtp: 'max_mime_depth'
deleted -> smtp: 'max_mime_mem'
deleted -> smtp: 'memcap'
deleted -> smtp: 'no_alerts'
deleted -> smtp: 'print_cmds'
deleted -> ssh: 'autodetect'
deleted -> ssh: 'enable_badmsgdir'
deleted -> ssh: 'enable_paysize'
```

```
deleted -> ssh: 'enable_protomismatch'
deleted -> ssh: 'enable_recognition'
deleted -> ssh: 'enable_respoverflow'
deleted -> ssh: 'enable_srvoverflow'
deleted -> ssh: 'enable_ssh1crc32'
deleted -> ssl: 'noinspect_encrypted'
deleted -> stream5_global: 'disabled'
deleted -> stream5_global: 'flush_on_alert'
deleted -> stream5_global: 'no_midstream_drop_alerts'
deleted -> stream5_tcp: 'check_session_hijacking'
deleted -> stream5_tcp: 'detect_anomalies'
deleted -> stream5_tcp: 'dont_store_large_packets'
deleted -> stream5_tcp: 'policy noack'
deleted -> stream5_tcp: 'policy unknown'
deleted -> tcpdump: '<filename> can no longer be specific'
deleted -> test: 'file'
deleted -> test: 'stdout'
deleted -> unified2: 'filename'
```

20.11 Module Listing

- **ack** (ips_option): rule option to match on TCP ack numbers
- **active** (basic): configure responses
- **alert_csv** (logger): output event in csv format
- **alert_ex** (logger): output gid:sid:rev for alerts
- **alert_fast** (logger): output event with brief text format
- **alert_full** (logger): output event with full packet dump
- **alert_json** (logger): output event in json format
- **alert_sfsocket** (logger): output event over socket
- **alert_syslog** (logger): output event to syslog
- **alert_unixsock** (logger): output event over unix socket
- **alerts** (basic): configure alerts
- **appid** (inspector): application and service identification
- **appids** (ips_option): detection option for application ids
- **arp** (codec): support for address resolution protocol
- **arp_spoof** (inspector): detect ARP attacks and anomalies
- **asn1** (ips_option): rule option for asn1 detection
- **attribute_table** (basic): configure hosts loading
- **auth** (codec): support for IP authentication header
- **back_orifice** (inspector): back orifice detection
- **base64_decode** (ips_option): rule option to decode base64 data - must be used with base64_data option
- **binder** (inspector): configure processing based on CIDRs, ports, services, etc.
- **bufferlen** (ips_option): rule option to check length of current buffer

- **byte_extract** (ips_option): rule option to convert data to an integer variable
- **byte_jump** (ips_option): rule option to move the detection cursor
- **byte_math** (ips_option): rule option to perform mathematical operations on extracted value and a specified value or existing variable
- **byte_test** (ips_option): rule option to convert data to integer and compare
- **ciscometadata** (codec): support for cisco metadata
- **classifications** (basic): define rule categories with priority
- **classtype** (ips_option): general rule option for rule classification
- **content** (ips_option): payload rule option for basic pattern matching
- **cvs** (ips_option): payload rule option for detecting specific attacks
- **daq** (basic): configure packet acquisition interface
- **data_log** (inspector): log selected published data to data.log
- **dce_http_proxy** (inspector): dce over http inspection - client to/from proxy
- **dce_http_server** (inspector): dce over http inspection - proxy to/from server
- **dce_iface** (ips_option): detection option to check dcerpc interface
- **dce_opnum** (ips_option): detection option to check dcerpc operation number
- **dce_smb** (inspector): dce over smb inspection
- **dce_stub_data** (ips_option): sets the cursor to dcerpc stub data
- **dce_tcp** (inspector): dce over tcp inspection
- **dce_udp** (inspector): dce over udp inspection
- **decode** (basic): general decoder rules
- **detection** (basic): configure general IPS rule processing parameters
- **detection_filter** (ips_option): rule option to require multiple hits before a rule generates an event
- **dnp3** (inspector): dnp3 inspection
- **dnp3_data** (ips_option): sets the cursor to dnp3 data
- **dnp3_func** (ips_option): detection option to check DNP3 function code
- **dnp3_ind** (ips_option): detection option to check DNP3 indicator flags
- **dnp3_obj** (ips_option): detection option to check DNP3 object headers
- **dns** (inspector): dns inspection
- **domain_filter** (inspector): alert on configured HTTP domains
- **dpx** (inspector): dynamic inspector example
- **dsize** (ips_option): rule option to test payload size
- **eapol** (codec): support for extensible authentication protocol over LAN
- **erspan2** (codec): support for encapsulated remote switched port analyzer - type 2
- **erspan3** (codec): support for encapsulated remote switched port analyzer - type 3

- **esp** (codec): support for encapsulating security payload
- **eth** (codec): support for ethernet protocol (DLT 1) (DLT 51)
- **event_filter** (basic): configure thresholding of events
- **event_queue** (basic): configure event queue parameters
- **fabricpath** (codec): support for fabricpath
- **file_connector** (connector): implement the file based connector
- **file_data** (ips_option): rule option to set detection cursor to file data
- **file_id** (inspector): configure file identification
- **file_log** (inspector): log file event to file.log
- **file_type** (ips_option): rule option to check file type
- **flags** (ips_option): rule option to test TCP control flags
- **flow** (ips_option): rule option to check session properties
- **flowbits** (ips_option): rule option to set and test arbitrary boolean flags
- **fragbits** (ips_option): rule option to test IP frag flags
- **fragoffset** (ips_option): rule option to test IP frag offset
- **ftp_client** (inspector): FTP client configuration module for use with ftp_server
- **ftp_data** (inspector): FTP data channel handler
- **ftp_server** (inspector): main FTP module; ftp_client should also be configured
- **gid** (ips_option): rule option specifying rule generator
- **gre** (codec): support for generic routing encapsulation
- **gtp** (codec): support for general-packet-radio-service tunneling protocol
- **gtp_info** (ips_option): rule option to check gtp info element
- **gtp_inspect** (inspector): gtp control channel inspection
- **gtp_type** (ips_option): rule option to check gtp types
- **gtp_version** (ips_option): rule option to check GTP version
- **high_availability** (basic): implement flow tracking high availability
- **host_cache** (basic): configure hosts
- **host_tracker** (basic): configure hosts
- **hosts** (basic): configure hosts
- **http2_frame_data** (ips_option): rule option to see HTTP/2 frame body
- **http2_frame_header** (ips_option): rule option to see 9-octet HTTP/2 frame header
- **http2_inspect** (inspector): HTTP/2 inspector
- **http_client_body** (ips_option): rule option to set the detection cursor to the request body
- **http_cookie** (ips_option): rule option to set the detection cursor to the HTTP cookie
- **http_header** (ips_option): rule option to set the detection cursor to the normalized headers

- **http_inspect** (inspector): HTTP inspector
- **http_method** (ips_option): rule option to set the detection cursor to the HTTP request method
- **http_raw_body** (ips_option): rule option to set the detection cursor to the unnormalized message body
- **http_raw_cookie** (ips_option): rule option to set the detection cursor to the unnormalized cookie
- **http_raw_header** (ips_option): rule option to set the detection cursor to the unnormalized headers
- **http_raw_request** (ips_option): rule option to set the detection cursor to the unnormalized request line
- **http_raw_status** (ips_option): rule option to set the detection cursor to the unnormalized status line
- **http_raw_trailer** (ips_option): rule option to set the detection cursor to the unnormalized trailers
- **http_raw_uri** (ips_option): rule option to set the detection cursor to the unnormalized URI
- **http_stat_code** (ips_option): rule option to set the detection cursor to the HTTP status code
- **http_stat_msg** (ips_option): rule option to set the detection cursor to the HTTP status message
- **http_trailer** (ips_option): rule option to set the detection cursor to the normalized trailers
- **http_true_ip** (ips_option): rule option to set the detection cursor to the final client IP address
- **http_uri** (ips_option): rule option to set the detection cursor to the normalized URI buffer
- **http_version** (ips_option): rule option to set the detection cursor to the version buffer
- **icmp4** (codec): support for Internet control message protocol v4
- **icmp6** (codec): support for Internet control message protocol v6
- **icmp_id** (ips_option): rule option to check ICMP ID
- **icmp_seq** (ips_option): rule option to check ICMP sequence number
- **icode** (ips_option): rule option to check ICMP code
- **id** (ips_option): rule option to check the IP ID field
- **igmp** (codec): support for Internet group management protocol
- **imap** (inspector): imap inspection
- **inspection** (basic): configure basic inspection policy parameters
- **ip_proto** (ips_option): rule option to check the IP protocol number
- **ipopts** (ips_option): rule option to check for IP options
- **ips** (basic): configure IPS rule processing
- **ipv4** (codec): support for Internet protocol v4 (DLT 228)
- **ipv6** (codec): support for Internet protocol v6 (DLT 229)
- **isdataat** (ips_option): rule option to check for the presence of payload data
- **itype** (ips_option): rule option to check ICMP type
- **latency** (basic): packet and rule latency monitoring and control
- **llc** (codec): support for logical link control
- **log_codecs** (logger): log protocols in packet by layer
- **log_hext** (logger): output payload suitable for daq hext

- **log_pcap** (logger): log packet in pcap format
- **md5** (ips_option): payload rule option for hash matching
- **memory** (basic): memory management configuration
- **metadata** (ips_option): rule option for conveying arbitrary name, value data within the rule text
- **modbus** (inspector): modbus inspection
- **modbus_data** (ips_option): rule option to set cursor to modbus data
- **modbus_func** (ips_option): rule option to check modbus function code
- **modbus_unit** (ips_option): rule option to check Modbus unit ID
- **mpls** (codec): support for multiprotocol label switching
- **msg** (ips_option): rule option summarizing rule purpose output with events
- **mss** (ips_option): detection for TCP maximum segment size
- **network** (basic): configure basic network parameters
- **normalizer** (inspector): packet scrubbing for inline mode
- **output** (basic): configure general output parameters
- **packet_capture** (inspector): raw packet dumping facility
- **packet_tracer** (basic): generate debug trace messages for packets
- **packets** (basic): configure basic packet handling
- **pbb** (codec): support for 802.1ah protocol
- **pcre** (ips_option): rule option for matching payload data with pcre
- **perf_monitor** (inspector): performance monitoring and flow statistics collection
- **pgm** (codec): support for pragmatic general multicast
- **pkt_data** (ips_option): rule option to set the detection cursor to the normalized packet data
- **pkt_num** (ips_option): alert on raw packet number
- **pop** (inspector): pop inspection
- **port_scan** (inspector): detect various ip, icmp, tcp, and udp port or protocol scans
- **pppoe** (codec): support for point-to-point protocol over ethernet
- **priority** (ips_option): rule option for prioritizing events
- **process** (basic): configure basic process setup
- **profiler** (basic): configure profiling of rules and/or modules
- **rate_filter** (basic): configure rate filters (which change rule actions)
- **raw_data** (ips_option): rule option to set the detection cursor to the raw packet data
- **react** (ips_action): send response to client and terminate session
- **reference** (ips_option): rule option to indicate relevant attack identification system
- **references** (basic): define reference systems used in rules
- **reg_test** (inspector): The regression test inspector (rti) is used when special packet handling is required for a reg test

- **regex** (ips_option): rule option for matching payload data with hyperscan regex
- **reject** (ips_action): terminate session with TCP reset or ICMP unreachable
- **rem** (ips_option): rule option to convey an arbitrary comment in the rule body
- **replace** (ips_option): rule option to overwrite payload data; use with rewrite action
- **reputation** (inspector): reputation inspection
- **rev** (ips_option): rule option to indicate current revision of signature
- **rewrite** (ips_action): overwrite packet contents
- **rpc** (ips_option): rule option to check SUNRPC CALL parameters
- **rpc_decode** (inspector): RPC inspector
- **rule_state** (basic): enable/disable specific IPS rules
- **sd_pattern** (ips_option): rule option for detecting sensitive data
- **search_engine** (basic): configure fast pattern matcher
- **seq** (ips_option): rule option to check TCP sequence number
- **service** (ips_option): rule option to specify list of services for grouping rules
- **session** (ips_option): rule option to check user data from TCP sessions
- **sha256** (ips_option): payload rule option for hash matching
- **sha512** (ips_option): payload rule option for hash matching
- **sid** (ips_option): rule option to indicate signature number
- **side_channel** (basic): implement the side-channel asynchronous messaging subsystem
- **sip** (inspector): sip inspection
- **sip_body** (ips_option): rule option to set the detection cursor to the request body
- **sip_header** (ips_option): rule option to set the detection cursor to the SIP header buffer
- **sip_method** (ips_option): detection option for sip stat code
- **sip_stat_code** (ips_option): detection option for sip stat code
- **smtp** (inspector): smtp inspection
- **snort** (basic): command line configuration and shell commands
- **so** (ips_option): rule option to call custom eval function
- **soid** (ips_option): rule option to specify a shared object rule ID
- **ssh** (inspector): ssh inspection
- **ssl** (inspector): ssl inspection
- **ssl_state** (ips_option): detection option for ssl state
- **ssl_version** (ips_option): detection option for ssl version
- **stream** (inspector): common flow tracking
- **stream_file** (inspector): stream inspector for file flow tracking and processing
- **stream_icmp** (inspector): stream inspector for ICMP flow tracking

- **stream_ip** (inspector): stream inspector for IP flow tracking and defragmentation
- **stream_reassemble** (ips_option): detection option for stream reassembly control
- **stream_size** (ips_option): detection option for stream size checking
- **stream_tcp** (inspector): stream inspector for TCP flow tracking and stream normalization and reassembly
- **stream_udp** (inspector): stream inspector for UDP flow tracking
- **stream_user** (inspector): stream inspector for user flow tracking and reassembly
- **suppress** (basic): configure event suppressions
- **tag** (ips_option): rule option to log additional packets
- **target** (ips_option): rule option to indicate target of attack
- **tcp** (codec): support for transmission control protocol
- **tcp_connector** (connector): implement the tcp stream connector
- **telnet** (inspector): telnet inspection and normalization
- **token_ring** (codec): support for token ring decoding
- **tos** (ips_option): rule option to check type of service field
- **ttl** (ips_option): rule option to check time to live field
- **udp** (codec): support for user datagram protocol
- **unified2** (logger): output event and packet in unified2 format file
- **urg** (ips_option): detection for TCP urgent pointer
- **vlan** (codec): support for local area network
- **window** (ips_option): rule option to check TCP window field
- **wizard** (inspector): inspector that implements port-independent protocol identification
- **wlan** (codec): support for wireless local area network protocol (DLT 105)
- **wscale** (ips_option): detection for TCP window scale

20.12 Plugin Listing

- **codec::arp**: support for address resolution protocol
- **codec::auth**: support for IP authentication header
- **codec::bad_proto**: bad protocol id
- **codec::ciscometadata**: support for cisco metadata
- **codec::eapol**: support for extensible authentication protocol over LAN
- **codec::erspan2**: support for encapsulated remote switched port analyzer - type 2
- **codec::erspan3**: support for encapsulated remote switched port analyzer - type 3
- **codec::esp**: support for encapsulating security payload
- **codec::eth**: support for ethernet protocol (DLT 1) (DLT 51)
- **codec::fabricpath**: support for fabricpath

- **codec::gre**: support for generic routing encapsulation
- **codec::gtp**: support for general-packet-radio-service tunneling protocol
- **codec::icmp4**: support for Internet control message protocol v4
- **codec::icmp4_ip**: support for IP in ICMPv4
- **codec::icmp6**: support for Internet control message protocol v6
- **codec::icmp6_ip**: support for IP in ICMPv6
- **codec::igmp**: support for Internet group management protocol
- **codec::ipv4**: support for Internet protocol v4 (DLT 228)
- **codec::ipv6**: support for Internet protocol v6 (DLT 229)
- **codec::ipv6_dst_opts**: support for ipv6 destination options
- **codec::ipv6_frag**: support for IPv6 fragment decoding
- **codec::ipv6_hop_opts**: support for IPv6 hop options
- **codec::ipv6_mobility**: support for mobility
- **codec::ipv6_no_next**: sentinel codec
- **codec::ipv6_routing**: support for IPv6 routing extension
- **codec::linux_sll**: support for Linux SLL (DLT 113)
- **codec::llc**: support for logical link control
- **codec::mpls**: support for multiprotocol label switching
- **codec::null**: support for null encapsulation (DLT 0)
- **codec::pbb**: support for 802.1ah protocol
- **codec::pflog**: support for OpenBSD PF log (DLT 117)
- **codec::pgm**: support for pragmatic general multicast
- **codec::ppp**: support for point-to-point encapsulation (DLT 9)
- **codec::ppp_encap**: support for point-to-point encapsulation
- **codec::pppoe_disc**: support for point-to-point discovery
- **codec::pppoe_sess**: support for point-to-point session
- **codec::raw**: support for raw IP (DLT 12)
- **codec::slip**: support for slip protocol (DLT 8)
- **codec::tcp**: support for transmission control protocol
- **codec::teredo**: support for teredo
- **codec::token_ring**: support for token ring decoding
- **codec::trans_bridge**: support for trans-bridging
- **codec::udp**: support for user datagram protocol
- **codec::user**: support for user sessions (DLT 230)
- **codec::vlan**: support for local area network

- **codec::wlan**: support for wireless local area network protocol (DLT 105)
- **connector::file_connector**: implement the file based connector
- **connector::tcp_connector**: implement the tcp stream connector
- **inspector::appid**: application and service identification
- **inspector::arp_spoof**: detect ARP attacks and anomalies
- **inspector::back_orifice**: back orifice detection
- **inspector::binder**: configure processing based on CIDRs, ports, services, etc.
- **inspector::data_log**: log selected published data to data.log
- **inspector::dce_http_proxy**: dce over http inspection - client to/from proxy
- **inspector::dce_http_server**: dce over http inspection - proxy to/from server
- **inspector::dce_smb**: dce over smb inspection
- **inspector::dce_tcp**: dce over tcp inspection
- **inspector::dce_udp**: dce over udp inspection
- **inspector::dnp3**: dnp3 inspection
- **inspector::dns**: dns inspection
- **inspector::domain_filter**: alert on configured HTTP domains
- **inspector::dpx**: dynamic inspector example
- **inspector::file_id**: configure file identification
- **inspector::file_log**: log file event to file.log
- **inspector::ftp_client**: FTP inspector client module
- **inspector::ftp_data**: FTP data channel handler
- **inspector::ftp_server**: FTP inspector server module
- **inspector::gtp_inspect**: gtp control channel inspection
- **inspector::http2_inspect**: the HTTP/2 inspector
- **inspector::http_inspect**: the new HTTP inspector!
- **inspector::imap**: imap inspection
- **inspector::modbus**: modbus inspection
- **inspector::normalizer**: packet scrubbing for inline mode
- **inspector::packet_capture**: raw packet dumping facility
- **inspector::perf_monitor**: performance monitoring and flow statistics collection
- **inspector::pop**: pop inspection
- **inspector::port_scan**: detect various ip, icmp, tcp, and udp port or protocol scans
- **inspector::reg_test**: The regression test inspector (rti) is used when special packet handling is required for a reg test
- **inspector::reputation**: reputation inspection
- **inspector::rpc_decode**: RPC inspector

- **inspector::sip**: sip inspection
- **inspector::smtp**: smtp inspection
- **inspector::ssh**: ssh inspection
- **inspector::ssl**: ssl inspection
- **inspector::stream**: common flow tracking
- **inspector::stream_file**: stream inspector for file flow tracking and processing
- **inspector::stream_icmp**: stream inspector for ICMP flow tracking
- **inspector::stream_ip**: stream inspector for IP flow tracking and defragmentation
- **inspector::stream_tcp**: stream inspector for TCP flow tracking and stream normalization and reassembly
- **inspector::stream_udp**: stream inspector for UDP flow tracking
- **inspector::stream_user**: stream inspector for user flow tracking and reassembly
- **inspector::telnet**: telnet inspection and normalization
- **inspector::wizard**: inspector that implements port-independent protocol identification
- **ips_action::react**: send response to client and terminate session
- **ips_action::reject**: terminate session with TCP reset or ICMP unreachable
- **ips_action::rewrite**: overwrite packet contents
- **ips_option::ack**: rule option to match on TCP ack numbers
- **ips_option::appids**: detection option for application ids
- **ips_option::asn1**: rule option for asn1 detection
- **ips_option::base64_data**: set detection cursor to decoded Base64 data
- **ips_option::base64_decode**: rule option to decode base64 data - must be used with base64_data option
- **ips_option::bufferlen**: rule option to check length of current buffer
- **ips_option::byte_extract**: rule option to convert data to an integer variable
- **ips_option::byte_jump**: rule option to move the detection cursor
- **ips_option::byte_math**: rule option to perform mathematical operations on extracted value and a specified value or existing variable
- **ips_option::byte_test**: rule option to convert data to integer and compare
- **ips_option::classtype**: general rule option for rule classification
- **ips_option::content**: payload rule option for basic pattern matching
- **ips_option::cvs**: payload rule option for detecting specific attacks
- **ips_option::dce_iface**: detection option to check dcerpc interface
- **ips_option::dce_opnum**: detection option to check dcerpc operation number
- **ips_option::dce_stub_data**: sets the cursor to dcerpc stub data
- **ips_option::detection_filter**: rule option to require multiple hits before a rule generates an event
- **ips_option::dnp3_data**: sets the cursor to dnp3 data

- **ips_option::dnp3_func**: detection option to check DNP3 function code
- **ips_option::dnp3_ind**: detection option to check DNP3 indicator flags
- **ips_option::dnp3_obj**: detection option to check DNP3 object headers
- **ips_option::dsize**: rule option to test payload size
- **ips_option::file_data**: rule option to set detection cursor to file data
- **ips_option::file_type**: rule option to check file type
- **ips_option::flags**: rule option to test TCP control flags
- **ips_option::flow**: rule option to check session properties
- **ips_option::flowbits**: rule option to set and test arbitrary boolean flags
- **ips_option::fragbits**: rule option to test IP frag flags
- **ips_option::fragoffset**: rule option to test IP frag offset
- **ips_option::gid**: rule option specifying rule generator
- **ips_option::gtp_info**: rule option to check gtp info element
- **ips_option::gtp_type**: rule option to check gtp types
- **ips_option::gtp_version**: rule option to check GTP version
- **ips_option::http2_frame_data**: rule option to see HTTP/2 frame body
- **ips_option::http2_frame_header**: rule option to see 9-octet HTTP/2 frame header
- **ips_option::http_client_body**: rule option to set the detection cursor to the request body
- **ips_option::http_cookie**: rule option to set the detection cursor to the HTTP cookie
- **ips_option::http_header**: rule option to set the detection cursor to the normalized headers
- **ips_option::http_method**: rule option to set the detection cursor to the HTTP request method
- **ips_option::http_raw_body**: rule option to set the detection cursor to the unnormalized message body
- **ips_option::http_raw_cookie**: rule option to set the detection cursor to the unnormalized cookie
- **ips_option::http_raw_header**: rule option to set the detection cursor to the unnormalized headers
- **ips_option::http_raw_request**: rule option to set the detection cursor to the unnormalized request line
- **ips_option::http_raw_status**: rule option to set the detection cursor to the unnormalized status line
- **ips_option::http_raw_trailer**: rule option to set the detection cursor to the unnormalized trailers
- **ips_option::http_raw_uri**: rule option to set the detection cursor to the unnormalized URI
- **ips_option::http_stat_code**: rule option to set the detection cursor to the HTTP status code
- **ips_option::http_stat_msg**: rule option to set the detection cursor to the HTTP status message
- **ips_option::http_trailer**: rule option to set the detection cursor to the normalized trailers
- **ips_option::http_true_ip**: rule option to set the detection cursor to the final client IP address
- **ips_option::http_uri**: rule option to set the detection cursor to the normalized URI buffer
- **ips_option::http_version**: rule option to set the detection cursor to the version buffer
- **ips_option::icmp_id**: rule option to check ICMP ID

- **ips_option::icmp_seq**: rule option to check ICMP sequence number
- **ips_option::icode**: rule option to check ICMP code
- **ips_option::id**: rule option to check the IP ID field
- **ips_option::ip_proto**: rule option to check the IP protocol number
- **ips_option::ipopts**: rule option to check for IP options
- **ips_option::isdataat**: rule option to check for the presence of payload data
- **ips_option::itype**: rule option to check ICMP type
- **ips_option::md5**: payload rule option for hash matching
- **ips_option::metadata**: rule option for conveying arbitrary name, value data within the rule text
- **ips_option::modbus_data**: rule option to set cursor to modbus data
- **ips_option::modbus_func**: rule option to check modbus function code
- **ips_option::modbus_unit**: rule option to check Modbus unit ID
- **ips_option::msg**: rule option summarizing rule purpose output with events
- **ips_option::mss**: detection for TCP maximum segment size
- **ips_option::pcre**: rule option for matching payload data with pcre
- **ips_option::pkt_data**: rule option to set the detection cursor to the normalized packet data
- **ips_option::pkt_num**: alert on raw packet number
- **ips_option::priority**: rule option for prioritizing events
- **ips_option::raw_data**: rule option to set the detection cursor to the raw packet data
- **ips_option::reference**: rule option to indicate relevant attack identification system
- **ips_option::regex**: rule option for matching payload data with hyperscan regex
- **ips_option::rem**: rule option to convey an arbitrary comment in the rule body
- **ips_option::replace**: rule option to overwrite payload data; use with rewrite action
- **ips_option::rev**: rule option to indicate current revision of signature
- **ips_option::rpc**: rule option to check SUNRPC CALL parameters
- **ips_option::sd_pattern**: rule option for detecting sensitive data
- **ips_option::seq**: rule option to check TCP sequence number
- **ips_option::service**: rule option to specify list of services for grouping rules
- **ips_option::session**: rule option to check user data from TCP sessions
- **ips_option::sha256**: payload rule option for hash matching
- **ips_option::sha512**: payload rule option for hash matching
- **ips_option::sid**: rule option to indicate signature number
- **ips_option::sip_body**: rule option to set the detection cursor to the request body
- **ips_option::sip_header**: rule option to set the detection cursor to the SIP header buffer
- **ips_option::sip_method**: detection option for sip stat code

- **ips_option::sip_stat_code**: detection option for sip stat code
- **ips_option::so**: rule option to call custom eval function
- **ips_option::soid**: rule option to specify a shared object rule ID
- **ips_option::ssl_state**: detection option for ssl state
- **ips_option::ssl_version**: detection option for ssl version
- **ips_option::stream_reassemble**: detection option for stream reassembly control
- **ips_option::stream_size**: detection option for stream size checking
- **ips_option::tag**: rule option to log additional packets
- **ips_option::target**: rule option to indicate target of attack
- **ips_option::tos**: rule option to check type of service field
- **ips_option::ttl**: rule option to check time to live field
- **ips_option::urg**: detection for TCP urgent pointer
- **ips_option::window**: rule option to check TCP window field
- **ips_option::wscale**: detection for TCP window scale
- **logger::alert_csv**: output event in csv format
- **logger::alert_ex**: output gid:sid:rev for alerts
- **logger::alert_fast**: output event with brief text format
- **logger::alert_full**: output event with full packet dump
- **logger::alert_json**: output event in json format
- **logger::alert_sfsocket**: output event over socket
- **logger::alert_syslog**: output event to syslog
- **logger::alert_unixsock**: output event over unix socket
- **logger::log_codecs**: log protocols in packet by layer
- **logger::log_hext**: output payload suitable for daq hext
- **logger::log_null**: disable logging of packets
- **logger::log_pcap**: log packet in pcap format
- **logger::unified2**: output event and packet in unified2 format file
- **piglet::pp_codec**: Codec piglet
- **piglet::pp_inspector**: Inspector piglet
- **piglet::pp_ips_action**: Ips action piglet
- **piglet::pp_ips_option**: Ips option piglet
- **piglet::pp_logger**: Logger piglet
- **piglet::pp_search_engine**: Search engine piglet
- **piglet::pp_so_rule**: SO rule piglet
- **piglet::pp_test**: Test piglet

- **search_engine::ac_banded**: Aho-Corasick Banded (high memory, moderate performance)
- **search_engine::ac_bnfa**: Aho-Corasick Binary NFA (low memory, high performance) MPSE
- **search_engine::ac_full**: Aho-Corasick Full (high memory, best performance), implements search_all()
- **search_engine::ac_sparse**: Aho-Corasick Sparse (high memory, moderate performance) MPSE
- **search_engine::ac_sparse_bands**: Aho-Corasick Sparse-Banded (high memory, moderate performance) MPSE
- **search_engine::ac_std**: Aho-Corasick Full (high memory, best performance) MPSE
- **search_engine::hyperscan**: intel hyperscan-based mpse with regex support
- **search_engine::lowmem**: Keyword Trie (low memory, moderate performance) MPSE
- **so_rule::3|18758**: SO rule example

20.13 LibDAQ and DAQ Modules

Snort 2.9 introduces the DAQ, or Data Acquisition library, for packet I/O. The DAQ replaces direct calls to libpcap 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. The DAQ library may be useful for other packet processing applications and the modular nature allows you to build new modules for other platforms.

This README summarizes the important things you need to know to use the DAQ.

20.13.1 Building the DAQ Library and DAQ Modules

The DAQ is bundled with Snort but must be built first using these steps:

```
./configure
make
sudo make install
```

This will build and install both static and dynamic DAQ modules.

Note that pcap >= 1.5.0 is required. pcap 1.8.1 is available at the time of this writing and is recommended.

Also, libdnet is required for IPQ and NFQ DAQs. If you get a relocation error trying to build those DAQs, you may need to reinstall libdnet and configure it with something like this:

```
./configure "CFLAGS=-fPIC -g -O2"
```

You may also experience problems trying to find the dynamic dnet library because it isn't always named properly. Try creating a link to the shared library (identified by its .x or .x.y etc. extension) with the same name but with ".so" inserted as follows:

```
$ ln -s libdnet.1.1 libdnet.so.1.1
$ ldconfig -Rv /usr/local/lib 2>&1 | grep dnet
  Adding /usr/local/lib/libdnet.so.1.1
```

Alternatively, you should be able to fix both issues as follows:

```
libtoolize --copy --force
aclocal -I config
autoheader
autoconf
automake --foreign
```

When the DAQ library is built, both static and dynamic flavors will be generated. The various DAQ modules will be built if the requisite headers and libraries are available. You can disable individual modules, etc. with options to configure. For the complete list of configure options, run:

```
./configure --help
```

20.13.2 PCAP Module

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>
```

Immediate (less-buffered or unbuffered) delivery mode can be enabled with:

```
./snort --daq pcap --daq-var immediate=1
```

This immediate delivery mode can be particularly useful on modern Linux systems with TPACKET_V3 support. LibPCAP will attempt to use this mode when it is available, but it introduces some potentially undesirable behavior in exchange for better performance. The most notable behavior change is that the packet timeout will never occur if packets are not being received, causing the poll() to potentially hang indefinitely. Enabling immediate delivery mode will cause LibPCAP to use TPACKET_V2 instead of TPACKET_V3.

- The pcap DAQ does not count filtered packets. *

20.13.3 AFPACKET Module

afpacket functions similar to the pcap DAQ but with better performance:

```
./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 craft the device string as 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:

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

Note
Linux kernel version 2.6.31 or higher is required for the AFPacket DAQ module due to its dependency on both TPACKET v2 and PACKET_TX_RING support.

Fanout (Kernel Loadbalancing)

More recent Linux kernel versions (3.1+) support various kernel-space loadbalancing methods within AFPacket configured using the PACKET_FANOUT ioctl. This allows you to have multiple AFPacket DAQ module instances processing packets from the same interfaces in parallel for significantly improved throughput.

To configure PACKET_FANOUT in the AFPacket DAQ module, two DAQ variables are used:

```
--daq-var fanout_type=<hash|lb|cpu|rollover|rnd|qm>
```

and (optionally):

```
--daq-var fanout_flag=<rollover|defrag>
```

In general, you're going to want to use the *hash* fanout type, but the others have been included for completeness. The *defrag* fanout flag is probably a good idea to correctly handle loadbalancing of flows containing fragmented packets.

Please read the man page for *packet* or packet_mmap.txt in the Linux kernel source for more details on the different fanout types and modifier flags.

20.13.4 NFQ Module

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>]

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

This module can not run unprivileged so ./snort -u -g will produce a warning and won't change user or group.

Notes on iptables are given below.

20.13.5 IPQ Module

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
```

Note that layer 2 resets are not supported with the IPQ DAQ:

```
config layer2resets[: <mac>]
```

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
```

This module can not run unprivileged so ./snort -u -g will produce a warning and won't change user or group.

Notes on iptables are given below.

20.13.6 IPFW Module

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

```
./configure --enable-ipfw
```

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.

Notes on FreeBSD and OpenBSD are given below.

20.13.7 Dump Module

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>
```

The dump DAQ also supports text output of verdicts rendered, injected packets, and other such items. In order to enable text output, the *output* DAQ variable must be set to either *text* (text output only) or *both* (both text and PCAP output will be written). The default filename for the text output is inline-out.txt, but it can be overridden like so:

```
./snort --daq dump --daq-var output=text --daq-var text-file=<filename>
```

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

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
```

20.13.8 Netmap Module

The netmap project is a framework for very high speed packet I/O. It is available on both FreeBSD and Linux with varying amounts of preparatory setup required. Specific notes for each follow.

```
./snort --daq netmap -i <device>
        [--daq-var debug]
```

If you want to run netmap in inline mode, you must craft the device string as 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:

```
em1:em2
```

or this:

```
em1:em2::em3:em4
```

Inline operation performs Layer 2 forwarding with no MAC filtering, akin to the AFPacket module's behavior. All packets received on one interface in an inline pair will be forwarded out the other interface unless dropped by the reader and vice versa.

Important
The interfaces will need to be up and in promiscuous mode in order to function (*ifconfig em1 up promisc*). The DAQ module does not currently do either of these configuration steps for itself.

FreeBSD

In FreeBSD 10.0, netmap has been integrated into the core OS. In order to use it, you must recompile your kernel with the line

```
device netmap
```

added to your kernel config.

Linux

You will need to download the netmap source code from the project's repository:

```
https://code.google.com/p/netmap/
```

Follow the instructions on the project's homepage for compiling and installing the code:

```
http://info.iet.unipi.it/~luigi/netmap/
```

It will involve a standalone kernel module (netmap_lin) as well as patching and rebuilding the kernel module used to drive your network adapters. The following drivers are supported under Linux at the time of writing (June 2014):

```
e1000
e1000e
forcedeth
igb
ixgbe
r8169
virtio
```

TODO: - Support for attaching to only a single ring (queue) on a network adapter. - Support for VALE and netmap pipes.

20.13.9 Notes on iptables

These notes are just a quick reminder that you need to set up iptables to use the IPQ or NFQ DAQs. Doing so may cause problems with your network so tread carefully. The examples below are intentionally incomplete so please read the related documentation first.

Here is a blog post by Marty for historical reference:

```
http://archives.neohapsis.com/archives/snort/2000-11/0394.html
```

You can check this out for queue sizing tips:

```
http://www.inliniac.net/blog/2008/01/23/improving-snort_inlines-nfq-performance. ↩
    html
```

You might find useful IPQ info here:

```
http://snort-inline.sourceforge.net/
```

Use this to examine your iptables:

```
sudo /sbin/iptables -L
```

Use something like this to set up NFQ:

```
sudo /sbin/iptables
    -I <table> [<protocol stuff>] [<state stuff>]
    -j NFQUEUE --queue-num 1
```

Use something like this to set up IPQ:

```
sudo iptables -I FORWARD -j QUEUE
```

Use something like this to "disconnect" snort:

```
sudo /sbin/iptables -D <table> <rule pos>
```

Be sure to start Snort prior to routing packets through NFQ with iptables. Such packets will be dropped until Snort is started.

The queue-num is the number you must give Snort.

If you are running on a system with both NFQ and IPQ support, you may experience some start-up failures of the sort:

The solution seems to be to remove both modules from the kernel like this:

```
modprobe -r nfnetlink_queue
modprobe -r ip_queue
```

and then install the module you want:

```
modprobe ip_queue
```

or:

```
modprobe nfnetlink_queue
```

These DAQs should be run with a snaplen of 65535 since the kernel defrags the packets before queuing. Also, no need to configure frag3.

20.13.10 Notes on FreeBSD::IPFW

Check the online manual at:

```
http://www.freebsd.org/doc/handbook/firewalls-ipfw.html.
```

Here is a brief example to divert icmp packets to Snort at port 8000:

To enable support for divert sockets, place the following lines in the kernel configuration file:

```
options IPFIREWALL
options IPDIVERT
```

(The file in this case was: /usr/src/sys/i386/conf/GENERIC; which is platform dependent.)

You may need to also set these to use the loadable kernel modules:

```
/etc/rc.conf:
firewall_enable="YES"

/boot/loader.conf:
ipfw_load="YES"
ipdivert_load="YES"

$ dmesg | grep ipfw
ipfw2 (+ipv6) initialized, divert loadable, nat loadable, rule-based
forwarding disabled, default to deny, logging disabled

$ kldload -v ipdivert
Loaded ipdivert, id=4

$ ipfw add 75 divert 8000 icmp from any to any
00075 divert 8000 icmp from any to any

$ ipfw list
...
00075 divert 8000 icmp from any to any
00080 allow icmp from any to any
...
```

- Note that on FreeBSD, divert sockets don't work with bridges!

Please refer to the following articles for more information:

https://forums.snort.org/forums/support/topics/snort-inline-on-freebsd-ipfw http://freebsd.rogness.net/snort_inline/

NAT gateway can be used with divert sockets if the network environment is conducive to using NAT.

The steps to set up NAT with ipfw are as follows:

1. Set up NAT with two interface em0 and em1 by adding the following to /etc/rc.conf. Here em0 is connected to external network and em1 to host-only LAN.

```
gateway_enable="YES"
natd_program="/sbin/natd"   # path to natd
natd_enable="YES"           # Enable natd (if firewall_enable == YES)
natd_interface="em0"       # Public interface or IP Address
natd_flags="-dynamic"       # Additional flags
defaultrouter=""
ifconfig_em0="DHCP"
ifconfig_em1="inet 192.168.1.2 netmask 255.255.255.0"
firewall_enable="YES"
firewall_script="/etc/rc.firewall"
firewall_type="simple"
```

2. Add the following divert rules to divert packets to Snort above and below the NAT rule in the "Simple" section of /etc/rc.firewall.

```
   ...
   # Inspect outbound packets (those arriving on "inside" interface)
   # before NAT translation.
   ${fwcmd} add divert 8000 all from any to any in via ${iif}
   case ${natd_enable} in
   [Yy][Ee][Ss])
       if [ -n "${natd_interface}" ]; then
           ${fwcmd} add divert natd all from any to any via
${natd_interface}
       fi
       ;;
   esac
   ...
   # Inspect inbound packets (those arriving on "outside" interface)
   # after NAT translation that aren't blocked for other reasons,
   # after the TCP "established" rule.
   ${fwcmd} add divert 8000 all from any to any in via ${oif}
```

20.13.11 Notes on OpenBSD::IPFW

OpenBSD supports divert sockets as of 4.7, so we use the ipfw DAQ.

Here is one way to set things up:

1. Configure the system to forward packets:

```
$ sysctl net.inet.ip.forwarding=1
$ sysctl net.inet6.ip6.forwarding=1

(You can also put that in /etc/sysctl.conf to enable on boot.)
```

2. Set up interfaces

```
$ dhclient vic1
$ dhclient vic2
```

3. Set up packet filter rules:

```
$ echo "pass out on vic1 divert-packet port 9000 keep-state" > rules.txt
$ echo "pass out on vic2 divert-packet port 9000 keep-state" >> rules.txt

$ pfctl -v -f rules.txt
```

4. Analyze packets diverted to port 9000:

```
$ ./snort --daq ipfw --daq-var port=9000
```

- Note that on OpenBSD, divert sockets don't work with bridges!