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.

⚠NOTE

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.

Syntax

```
dce_opnum:<opnum-list>;

opnum-list   = opnum-item | opnum-item ',' opnum-list
opnum-item   = opnum | opnum-range
opnum-range  = opnum '-' opnum
opnum        = 0-65535
```

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 netbios rules were doing 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.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 94

03-02-2022-24064301-Hauptpost-0606

byte_test and byte_jump with dce

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 preprocessor will know the endianness of the request, it will be able to do the correct conversion.

byte_test

Syntax

```
byte_test:<convert>, [!]<operator>, <value>, <offset> [, relative], dce;

convert    = 1 | 2 | 4 (only with option "dce")
operator   = '<' | '=' | '>' | '&' | '^'
value      = 0 - 4294967295
offset     = -65535 to 65535
```

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.



byte_jump

Syntax

```
byte_jump:<convert>, <offset>[, relative][, multiplier <mult_value>] \
    [, align][, post_offet <adjustment_value>], dce;

convert           = 1 | 2 | 4 (only with option "dce")
offset            = -65535 to 65535
mult_value        = 0 - 65535
adjustment_value  = -65535 to 65535
```

Example

```
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.

Example of rule complexity reduction

The following two rules using the new rule options replace 64 (set and isset flowbit) rules that are necessary if the new rule options are not used:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET [135,139,445,593,1024:] \
    (msg:"dns R_Dnssrv funcs2 overflow attempt"; flow:established,to_server; \
    dce_iface:50abc2a4-574d-40b3-9d66-ee4fd5fba076; dce_opnum:0-11; dce_stub_data; \
    pcre:"/^.{12}(\x00\x00\x00\x00|.{12})/sR"; byte_jump:4,-4,relative,align,dce; \
    byte_test:4,>,256,4,relative,dce; reference:bugtraq,23470; reference:cve,2007-1748; \
    classtype:attempted-admin; sid:1000068;)

alert udp $EXTERNAL_NET any -> $HOME_NET [135,1024:] \
    (msg:"dns R_Dnssrv funcs2 overflow attempt"; flow:established,to_server; \
    dce_iface:50abc2a4-574d-40b3-9d66-ee4fd5fba076; dce_opnum:0-11; dce_stub_data; \
    pcre:"/^.{12}(\x00\x00\x00\x00|.{12})/sR"; byte_jump:4,-4,relative,align,dce; \
    byte_test:4,>,256,4,relative,dce; reference:bugtraq,23470; reference:cve,2007-1748; \
    classtype:attempted-admin; sid:1000069;)
```

2.2.14 Sensitive Data Preprocessor

The Sensitive Data preprocessor is a Snort module that performs detection and filtering of Personally Identifiable Information (PII). This information includes credit card numbers, U.S. Social Security numbers, and email addresses. A limited regular expression syntax is also included for defining your own PII.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 95

Dependencies

The Stream5 preprocessor must be enabled for the Sensitive Data preprocessor to work.

Preprocessor Configuration

Sensitive Data configuration is split into two parts: the preprocessor config, and the rule options. The preprocessor config starts with:

```
preprocessor sensitive_data:
```

Option syntax

Option	Argument	Required	Default
alert_threshold	<number>	NO	alert_threshold 25
mask_output	NONE	NO	OFF
ssn_file	<filename>	NO	OFF

```
alert_threshold     = 1 - 65535
```

Option explanations

`alert_threshold`

The preprocessor will alert when any combination of PII are detected in a session. This option specifies how many need to be detected before alerting. This should be set higher than the highest individual count in your "sd_pattern" rules.

`mask_output`

This option replaces all but the last 4 digits of a detected PII with "X"s. This is only done on credit card & Social Security numbers, where an organization's regulations may prevent them from seeing unencrypted numbers.

`ssn_file`

A Social Security number is broken up into 3 sections: Area (3 digits), Group (2 digits), and Serial (4 digits). On a monthly basis, the Social Security Administration publishes a list of which Group numbers are in use for each Area. These numbers can be updated in Snort by supplying a CSV file with the new maximum Group numbers to use. By default, Snort recognizes Social Security numbers issued up through November 2009.

Example preprocessor config

```
preprocessor sensitive_data: alert_threshold 25 \
                             mask_output \
                             ssn_file ssn_groups_Jan10.csv
```

Rule Options

Snort rules are used to specify which PII the preprocessor should look for. A new rule option is provided by the preprocessor:

`sd_pattern`

164447095646012585319

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 96

03-02-2022-24064301-Hauptpost-0607

This rule option specifies what type of PII a rule should detect.

Syntax

```
sd_pattern:<count>, <pattern>;
```

```
count   = 1 - 255
pattern = any string
```

Option Explanations

`count`

This dictates how many times a PII pattern must be matched for an alert to be generated. The count is tracked across all packets in a session.

`pattern`

This is where the pattern of the PII gets specified. There are a few built-in patterns to choose from:

`credit_card`

The "credit_card" pattern matches 15- and 16-digit credit card numbers. These numbers may have spaces, dashes, or nothing in between groups. This covers Visa, Mastercard, Discover, and American Express. Credit card numbers matched this way have their check digits verified using the Luhn algorithm.

`us_social`

This pattern matches against 9-digit U.S. Social Security numbers. The SSNs are expected to have dashes between the Area, Group, and Serial sections.
SSNs have no check digits, but the preprocessor will check matches against the list of currently allocated group numbers.

`us_social_nodashes`

This pattern matches U.S. Social Security numbers without dashes separating the Area, Group, and Serial sections.

`email`

This pattern matches against email addresses.

If the pattern specified is not one of the above built-in patterns, then it is the definition of a custom PII pattern. Custom PII types are defined using a limited regex-style syntax. The following special characters and escape sequences are supported:

\d	matches any digit
\D	matches any non-digit
\l	matches any letter
\L	matches any non-letter
\w	matches any alphanumeric character
\W	matches any non-alphanumeric character
{num}	used to repeat a character or escape sequence "num" times. example: "{3}" matches 3 digits.
?	makes the previous character or escape sequence optional. example: " ?" matches an optional space. This behaves in a greedy manner.
\\	matches a backslash
\{, \}	matches { and }
\?	matches a question mark.

Other characters in the pattern will be matched literally.

⚠NOTE

Unlike PCRE, \w in this rule option does NOT match underscores.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 97

164447095646012585319

Examples

```
sd_pattern: 2,us_social;
```

Alerts when 2 social security numbers (with dashes) appear in a session.

```
sd_pattern: 5,(\d{3})\d{3}-\d{4};
```

Alerts on 5 U.S. phone numbers, following the format (123)456-7890

Whole rule example:

```
alert tcp $HOME_NET $HIGH_PORTS -> $EXTERNAL_NET $SMTP_PORTS \
(msg:"Credit Card numbers sent over email"; gid:138; sid:1000; rev:1; \
sd_pattern:4,credit_card; metadata:service smtp;)
```

Caveats

sd_pattern is not compatible with other rule options. Trying to use other rule options with sd_pattern will result in an error message.

Rules using sd_pattern must use GID 138.

1644470956460125853 19

2.2.15 Normalizer

When operating Snort in inline mode, it is helpful to normalize packets to help minimize the chances of evasion.

To enable the normalizer, use the following when configuring Snort:

```
./configure --enable-normalizer
```

The normalize preprocessor is activated via the conf as outlined below. There are also many new preprocessor and decoder rules to alert on or drop packets with "abnormal" encodings.

Note that in the following, fields are cleared only if they are non-zero. Also, normalizations will only be enabled if the selected DAQ supports packet replacement and is operating in inline mode.

If a policy is configured for inline_test or passive mode, any normalization statements in the policy config are ignored.

IP4 Normalizations

IP4 normalizations are enabled with:

```
preprocessor normalize_ip4: [df], [rf], [tos], [trim]
```

Base normalizations enabled with "preprocessor normalize_ip4" include:

- TTL normalization if enabled (explained below).
- NOP all options octets.

Optional normalizations include:

- df don't fragment: clear this bit on incoming packets.
- rf reserved flag: clear this bit on incoming packets.
- tos type of service (differentiated services) field: clear this byte.
- trim truncate packets with excess payload to the datagram length specified in the IP header + the layer 2 header (eg ethernet), but don't truncate below minimum frame length. This is automatically disabled if the DAQ can't inject packets.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 98

03-02-2022-24064301-Hauptpost-0608

IP6 Normalizations

IP6 normalizations are enabled with:

```
preprocessor normalize_ip6
```

Base normalizations enabled with "preprocessor `normalize_ip6`" include:

- **Hop limit normalizaton if enabled (explained below).**
- NOP all options octets in hop-by-hop and destination options extension headers.

ICMP4/6 Normalizations

ICMP4 and ICMP6 normalizations are enabled with:

```
preprocessor normalize_icmp4
preprocessor normalize_icmp6
```



Base normalizations enabled with the above include:

- Clear the code field in echo requests and replies.

TCP Normalizations

TCP normalizations are enabled with:

```
preprocessor normalize_tcp: \
    [ips] [urp] \
    [ecn <ecn_type>], \
    [opts [allow <allowed_opt>+]]

<ecn_type> ::= stream | packet

<allowed_opt> ::= \
    sack | echo | partial_order | conn_count | alt_checksum | md5 | <num>

<sack> ::= { 4, 5 }
<echo> ::= { 6, 7 }
<partial_order> ::= { 9, 10 }
<conn_count> ::= { 11, 12, 13 }
<alt_checksum> ::= { 14, 15 }
<md5> ::= { 19 }
<num> ::= (3..255)
```

Base normalizations enabled with "preprocessor `normalize_tcp`" include:

- Remove data on SYN.
- Clear the reserved bits in the TCP header.
- Clear the urgent pointer if the urgent flag is not set.
- Clear the urgent pointer and the urgent flag if there is no payload.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 99

- Set the urgent pointer to the payload length if it is greater than the payload length.
- Clear the urgent flag if the urgent pointer is not set.
- Clear any option padding bytes.
- Remove any data from RST packet.
- Trim data to window.
- Trim data to MSS.

Optional normalizations include:

- `ips`

  ensure consistency in retransmitted data (also forces reassembly policy to "first"). Any segments that can't be properly reassembled will be dropped.

- `urp`

  urgent pointer: don't adjust the urgent pointer if it is greater than payload length.

- `ecn packet`

  clear ECN flags on a per packet basis (regardless of negotiation).

- `ecn stream`

  clear ECN flags if usage wasn't negotiated. Should also enable `require_3whs`.

- `opts`

  NOP all option bytes other than maximum segment size, window scaling, timestamp, and any explicitly allowed with the allow keyword. You can allow options to pass by name or number.

- `opts`

  if timestamp is present but invalid, or valid but not negotiated, NOP the timestamp octets.

- `opts`

  if timestamp was negotiated but not present, block the packet.

- `opts`

  clear TS ECR if ACK flag is not set.

- `opts`

  MSS and window scale options are NOP'd if SYN flag is not set.

- `opts`

  trim payload length to MSS if longer.



TTL Normalization

TTL normalization pertains to both IP4 TTL (time-to-live) and IP6 (hop limit) and is only performed if both the relevant base normalization is enabled (as described above) and the minimum and new TTL values are configured, as follows:

```
config min_ttl: <min_ttl>
config new_ttl: <new_ttl>

<min_ttl> ::= (1..255)
<new_ttl> ::= (<min_ttl>+1..255)
```


Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 100

03-02-2022-24064301-HauptPost-0609

If new_ttl ¿ min_ttl, then if a packet is received with a TTL ¡ min_ttl, the TTL will be set to new_ttl.

Note that this configuration item was deprecated in 2.8.6:

```
preprocessor stream5_tcp: min_ttl <#>
```

By default min_ttl = 1 (TTL normalization is disabled). When TTL normalization is turned on the new_ttl is set to 5 by default.

2.3 Decoder and Preprocessor Rules

Decoder and preprocessor rules allow one to enable and disable decoder and preprocessor events on a rule by rule basis. They also allow one to specify the rule type or action of a decoder or preprocessor event on a rule by rule basis.

Decoder config options will still determine whether or not to generate decoder events. For example, if config disable_decode_alerts is in snort.conf, decoder events will not be generated regardless of whether or not there are corresponding rules for the event. Also note that if the decoder is configured to enable drops, e.g. config enable_decode_drops, these options will take precedence over the event type of the rule. A packet will be dropped if either a decoder config drop option is in snort.conf or the decoder or preprocessor rule type is drop. Of course, the drop cases only apply if Snort is running inline. See doc/README.decode for config options that control decoder events.



2.3.1 Configuring

The following options to configure will enable decoder and preprocessor rules:

```
$ ./configure --enable-decoder-preprocessor-rules
```

The decoder and preprocessor rules are located in the preproc_rules/ directory in the top level source tree, and have the names decoder.rules and preprocessor.rules respectively. These files are updated as new decoder and preprocessor events are added to Snort. The gen-msg.map under etc directory is also updated with new decoder and preprocessor rules.

To enable these rules in snort.conf, define the path to where the rules are located and uncomment the include lines in snort.conf that reference the rules files.

```
var PREPROC_RULE_PATH /path/to/preproc_rules
...
include $PREPROC_RULE_PATH/preprocessor.rules
include $PREPROC_RULE_PATH/decoder.rules
```

To disable any rule, just comment it with a # or remove the rule completely from the file (commenting is recommended).

To change the rule type or action of a decoder/preprocessor rule, just replace alert with the desired rule type. Any one of the following rule types can be used:

```
alert
log
pass
drop
sdrop
reject
```

For example one can change:

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 101

```
alert ( msg: "DECODE_NOT_IPV4_DGRAM"; sid: 1; gid: 116; rev: 1; \
    metadata: rule-type decode ; classtype:protocol-command-decode;)
```

to

```
drop ( msg: "DECODE_NOT_IPV4_DGRAM"; sid: 1; gid: 116; rev: 1; \
    metadata: rule-type decode ; classtype:protocol-command-decode;)
```

to drop (as well as alert on) packets where the Ethernet protocol is IPv4 but version field in IPv4 header has a value other than 4.

See `README.decode`, `README.gre` and the various preprocessor READMEs for descriptions of the rules in `decoder.rules` and `preprocessor.rules`.

The generator ids (gid) for different preprocessors and the decoder are as follows:

Generator Id	Module
105	Back Orifice preprocessor
106	RPC Decode preprocessor
112	Arpspoof preprocessor
116	Snort Decoder
119	HTTP Inspect preprocessor (Client)
120	HTTP Inspect preprocessor (Server)
122	Portscan preprocessor
123	Frag3 preprocessor
124	SMTP preprocessor
125	FTP (FTP) preprocessor
126	FTP (Telnet) preprocessor
127	ISAKMP preprocessor
128	SSH preprocessor
129	Stream5 preprocessor
131	DNS preprocessor
132	Skype preprocessor
133	DceRpc2 preprocessor
134	PPM preprocessor
137	SSL preprocessor
139	SDF preprocessor



2.3.2 Reverting to original behavior

If you have configured snort to use decoder and preprocessor rules, the following config option in `snort.conf` will make Snort revert to the old behavior:

```
config autogenerate_preprocessor_decoder_rules
```

Note that if you want to revert to the old behavior, you also have to remove the decoder and preprocessor rules and any reference to them from `snort.conf`, otherwise they will be loaded. This option applies to rules not specified and the default behavior is to alert.

2.4 Event Processing

Snort provides a variety of mechanisms to tune event processing to suit your needs:

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 102

03-02-2022-24064301-Hauptpost-0610

- Detection Filters

  You can use detection filters to specify a threshold that must be exceeded before a rule generates an event. This is covered in section 3.7.10.

- Rate Filters

  You can use rate filters to change a rule action when the number or rate of events indicates a possible attack.

- Event Filters

  You can use event filters to reduce the number of logged events for noisy rules. This can be tuned to significantly reduce false alarms.

- Event Suppression

  You can completely suppress the logging of unintersting events.

2.4.1 Rate Filtering

rate_filter provides rate based attack prevention by allowing users to configure a new action to take for a specified time when a given rate is exceeded. Multiple rate filters can be defined on the same rule, in which case they are evaluated in the order they appear in the configuration file, and the first applicable action is taken.



Format

Rate filters are used as standalone configurations (outside of a rule) and have the following format:

```
rate_filter \
    gen_id <gid>, sig_id <sid>, \
    track <by_src|by_dst|by_rule>, \
    count <c>, seconds <s>, \
    new_action alert|drop|pass|log|sdrop|reject, \
    timeout <seconds> \
    [, apply_to <ip-list>]
```

The options are described in the table below - all are required except apply_to, which is optional.

Examples

Example 1 - allow a maximum of 100 connection attempts per second from any one IP address, and block further connection attempts from that IP address for 10 seconds:

```
rate_filter \
    gen_id 135, sig_id 1, \
    track by_src, \
    count 100, seconds 1, \
    new_action drop, timeout 10
```

Example 2 - allow a maximum of 100 successful simultaneous connections from any one IP address, and block further connections from that IP address for 10 seconds:

```
rate_filter \
    gen_id 135, sig_id 2, \
    track by_src, \
    count 100, seconds 0, \
    new_action drop, timeout 10
```


Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 103

Option	Description
`track by_src | by_dst | by_rule`	rate is tracked either by source IP address, destination IP address, or by rule. This means the match statistics are maintained for each unique source IP address, for each unique destination IP address, or they are aggregated at rule level. For rules related to Stream5 sessions, source and destination means client and server respectively. `track by_rule` and `apply_to` may not be used together.
`count c`	the maximum number of rule matches in s seconds before the rate filter limit to is exceeded. c must be nonzero value.
`seconds s`	the time period over which `count` is accrued. 0 seconds means `count` is a total count instead of a specific rate. For example, `rate_filter` may be used to detect if the number of connections to a specific server exceed a specific count. 0 seconds only applies to internal rules (gen_id 135) and other use will produce a fatal error by Snort.
`new_action alert | drop | pass | log | sdrop | reject`	`new_action` replaces rule action for t seconds. `drop`, `reject`, and `sdrop` can be used only when snort is used in inline mode. `sdrop` and `reject` are conditionally compiled with GIDS.
`timeout t`	revert to the original rule action after t seconds. If t is 0, then rule action is never reverted back. An `event_filter` may be used to manage number of alerts after the rule action is enabled by `rate_filter`.
`apply_to <ip-list>`	restrict the configuration to only to source or destination IP address (indicated by track parameter) determined by `<ip-list>`. `track by_rule` and `apply_to` may not be used together. Note that events are generated during the timeout period, even if the rate falls below the configured limit.

1644470956460125853 19

2.4.2 Event Filtering

Event filtering can be used to reduce the number of logged alerts for noisy rules by limiting the number of times a particular event is logged during a specified time interval. This can be tuned to significantly reduce false alarms.

There are 3 types of event filters:

- `limit`

  Alerts on the 1st *m* events during the time interval, then ignores events for the rest of the time interval.

- `threshold`

  Alerts every *m* times we see this event during the time interval.

- `both`

  Alerts once per time interval after seeing *m* occurrences of the event, then ignores any additional events during the time interval.

Format

```
event_filter \
    gen_id <gid>, sig_id <sid>, \
    type <limit|threshold|both>, \
    track <by_src|by_dst>, \
    count <c>, seconds <s>

threshold \
    gen_id <gid>, sig_id <sid>, \
    type <limit|threshold|both>, \
    track <by_src|by_dst>, \
```


Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 104

03-02-2022-24064301-Hauptpost-0611

```
count <c>, seconds <s>
```

threshold is an alias for event_filter. Both formats are equivalent and support the options described below - all are required. threshold is deprecated and will not be supported in future releases.

Option	Description
gen_id <gid>	Specify the generator ID of an associated rule. gen_id 0, sig_id 0 can be used to specify a "global" threshold that applies to all rules.
sig_id <sid>	Specify the signature ID of an associated rule. sig_id 0 specifies a "global" filter because it applies to all sig_ids for the given gen_id.
type limit|threshold|both	type limit alerts on the 1st m events during the time interval, then ignores events for the rest of the time interval. Type threshold alerts every m times we see this event during the time interval. Type both alerts once per time interval after seeing m occurrences of the event, then ignores any additional events during the time interval.
track by_src|by_dst	rate is tracked either by source IP address, or destination IP address. This means count is maintained for each unique source IP addresses, or for each unique destination IP addresses. Ports or anything else are not tracked.
count c	number of rule matching in s seconds that will cause event_filter limit to be exceeded. c must be nonzero value.
seconds s	time period over which count is accrued. s must be nonzero value.



NOTE

Only one event_filter may be defined for a given gen_id, sig_id. If more than one event_filter is applied to a specific gen_id, sig_id pair, Snort will terminate with an error while reading the configuration information.

event_filters with sig_id 0 are considered "global" because they apply to all rules with the given gen_id. If gen_id is also 0, then the filter applies to all rules. (gen_id 0, sig_id != 0 is not allowed). Standard filtering tests are applied first, if they do not block an event from being logged, the global filtering test is applied. Thresholds in a rule (deprecated) will override a global event_filter. Global event_filters do not override what's in a signature or a more specific stand-alone event_filter.

NOTE

event_filters can be used to suppress excessive rate_filter alerts, however, the first new_action event of the timeout period is never suppressed. Such events indicate a change of state that are significant to the user monitoring the network.

Examples

Limit logging to 1 event per 60 seconds:

```
event_filter \
    gen_id 1, sig_id 1851, \
    type limit, track by_src,  \
    count 1, seconds 60
```

Limit logging to every 3rd event:

```
event_filter \
    gen_id 1, sig_id 1852,  \
    type threshold, track by_src, \
    count 3, seconds 60
```


Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 105

Limit logging to just 1 event per 60 seconds, but only if we exceed 30 events in 60 seconds:

```
event_filter \
    gen_id 1, sig_id 1853, \
    type both, track by_src, \
    count 30, seconds 60
```

Limit to logging 1 event per 60 seconds per IP triggering each rule (rule gen_id is 1):

```
event_filter \
    gen_id 1, sig_id 0, \
    type limit, track by_src, \
    count 1, seconds 60
```

Limit to logging 1 event per 60 seconds per IP, triggering each rule for each event generator:

```
event_filter \
    gen_id 0, sig_id 0, \
    type limit, track by_src, \
    count 1, seconds 60
```



Events in Snort are generated in the usual way, event filters are handled as part of the output system. Read gen-msg.map for details on gen ids.

Users can also configure a memcap for threshold with a "config:" option:

```
config event_filter: memcap <bytes>

# this is deprecated:
config threshold: memcap <bytes>
```

2.4.3 Event Suppression

Event suppression stops specified events from firing without removing the rule from the rule base. Suppression uses an IP list to select specific networks and users for suppression. Suppression tests are performed prior to either standard or global thresholding tests.

Suppression are standalone configurations that reference generators, SIDs, and IP addresses via an IP list . This allows a rule to be completely suppressed, or suppressed when the causative traffic is going to or coming from a specific IP or group of IP addresses.

You may apply multiple suppressions to a non-zero SID. You may also combine one `event_filter` and several suppressions to the same non-zero SID.

Format

The suppress configuration has two forms:

```
suppress \
    gen_id <gid>, sig_id <sid>, \

suppress \
    gen_id <gid>, sig_id <sid>, \
    track <by_src|by_dst>, ip <ip-list>
```


Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 106

03-02-2022-24064301-Hauptpost-0612

Option	Description
gen_id <gid>	Specify the generator ID of an associated rule. gen_id 0, sig_id 0 can be used to specify a "global" threshold that applies to all rules.
sig_id <sid>	Specify the signature ID of an associated rule. sig_id 0 specifies a "global" filter because it applies to all sig_ids for the given gen_id.
track by_src|by_dst	Suppress by source IP address or destination IP address. This is optional, but if present, ip must be provided as well.
ip <list>	Restrict the suppression to only source or destination IP addresses (indicated by track parameter) determined by ¡list¿. If track is provided, ip must be provided as well.

Examples

Suppress this event completely:

```
suppress gen_id 1, sig_id 1852:
```

Suppress this event from this IP:

```
suppress gen_id 1, sig_id 1852, track by_src, ip 10.1.1.54
```

Suppress this event to this CIDR block:

```
suppress gen_id 1, sig_id 1852, track by_dst, ip 10.1.1.0/24
```



2.4.4 Event Logging

Snort supports logging multiple events per packet/stream that are prioritized with different insertion methods, such as max content length or event ordering using the event queue.

The general configuration of the event queue is as follows:

```
config event_queue: [max_queue [size]] [log [size]] [order_events [TYPE]]
```

Event Queue Configuration Options There are three configuration options to the configuration parameter 'event_queue'.

1. max_queue

   This determines the maximum size of the event queue. For example, if the event queue has a max size of 8, only 8 events will be stored for a single packet or stream.

   The default value is 8.

2. log

   This determines the number of events to log for a given packet or stream. You can't log more than the max_event number that was specified.

   The default value is 3.

3. order_events

   This argument determines the way that the incoming events are ordered. We currently have two different methods:

   - priority - The highest priority (1 being the highest) events are ordered first.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 107

- content_length - Rules are ordered before decode or preprocessor alerts, and rules that have a longer content are ordered before rules with shorter contents.

The method in which events are ordered does not affect rule types such as pass, alert, log, etc.

The default value is content_length.

Event Queue Configuration Examples The default configuration:

```
config event_queue: max_queue 8 log 3 order_events content_length
```

Example of a reconfigured event queue:

```
config event_queue: max_queue 10 log 3 order_events content_length
```

Use the default event queue values, but change event order:

```
config event_queue: order_events priority
```

Use the default event queue values but change the number of logged events:

```
config event_queue: log 2
```



2.5 Performance Profiling

Snort can provide statistics on rule and preprocessor performance. Each require only a simple `config` option to `snort.conf` and Snort will print statistics on the worst (or all) performers on exit. When a file name is provided in `profile_rules` or `profile_preprocs`, the statistics will be saved in these files. If `append` is not specified, a new file will be created each time Snort is run. The filenames will have timestamps appended to them. These files will be found in the logging directory.

To use this feature, you must build snort with the `--enable-perfprofiling` option to the configure script.

2.5.1 Rule Profiling

Format

```
config profile_rules: \
    print [all | <num>], \
    sort <sort_option> \
    [,filename <filename> [append]]
```

- <num> is the number of rules to print
- <sort_option> is one of:
  - checks
  - matches
  - nomatches
  - avg_ticks
  - avg_ticks_per_match
  - avg_ticks_per_nomatch
  - total_ticks
- <filename> is the output filename
- [append] dictates that the output will go to the same file each time (optional)

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 108

03-02-2022-24064301-Hauptpost-0613

```
Rule Profile Statistics (worst 4 rules)
==========================================================
   Num      SID GID Rev      Checks   Matches    Alerts          Ticks Avg/Check  Avg/Match Avg/Nonmatch
   ---      --- --- ---      ------   -------    ------          ----- ---------  --------- ------------
     1     2389   1  12           1         1         1         385698  385698.0   385698.0          0.0
     2     2178   1  17           2         0         0         107822   53911.0        0.0      53911.0
     3     2179   1   8           2         0         0          92458   46229.0        0.0      46229.0
     4     1734   1  37           2         0         0          90054   45027.0        0.0      45027.0
```

Figure 2.1: Rule Profiling Example Output

Examples

- Print all rules, sort by avg_ticks (default configuration if option is turned on)

  ```
  config profile_rules
  ```

- Print all rules, sort by avg_ticks, and append to file `rules_stats.txt`

  ```
  config profile_rules:  filename rules_stats.txt append
  ```

- Print the top 10 rules, based on highest average time

  ```
  config profile_rules:  print 10, sort avg_ticks
  ```

- Print all rules, sorted by number of checks

  ```
  config profile_rules:  print all, sort checks
  ```

- Print top 100 rules, based on total time

  ```
  config profile_rules:  print 100, sort total_ticks
  ```

- Print with default options, save results to performance.txt each time

  ```
  config profile_rules:  filename performance.txt append
  ```

- Print top 20 rules, save results to perf.txt with timestamp in filename

  ```
  config profile_rules:  print 20, filename perf.txt
  ```

Output

Snort will print a table much like the following at exit.

Configuration line used to print the above table:

```
config profile_rules:  print 4, sort total_ticks
```

The columns represent:

- Number (rank)
- Sig ID
- Generator ID
- Checks (number of times rule was evaluated after fast pattern match within portgroup or any->any rules)
- Matches (number of times ALL rule options matched, will be high for rules that have no options)
- Alerts (number of alerts generated from this rule)
- CPU Ticks
- Avg Ticks per Check
- Avg Ticks per Match

164470956460125853 19

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 109

- Avg Ticks per Nonmatch

Interpreting this info is the key. The Microsecs (or Ticks) column is important because that is the total time spent evaluating a given rule. But, if that rule is causing alerts, it makes sense to leave it alone.

A high Avg/Check is a poor performing rule, that most likely contains PCRE. High Checks and low Avg/Check is usually an any->any rule with few rule options and no content. Quick to check, the few options may or may not match. We are looking at moving some of these into code, especially those with low SIDs.

By default, this information will be printed to the console when Snort exits. You can use the "filename" option in snort.conf to specify a file where this will be written. If "append" is not specified, a new file will be created each time Snort is run. The filenames will have timestamps appended to them. These files will be found in the logging directory.

2.5.2 Preprocessor Profiling

Format

```
config profile_preprocs: \
    print [all | <num>], \
    sort <sort_option> \
    [, filename <filename> [append]]
```



- <num> is the number of preprocessors to print
- <sort_option> is one of:
  - checks
  - avg_ticks
  - total_ticks
- <filename> is the output filename
- [append] dictates that the output will go to the same file each time (optional)

Examples

- Print all preprocessors, sort by avg_ticks (default configuration if option is turned on)
  `config profile_preprocs`
- Print all preprocessors, sort by avg_ticks, and append to file `preprocs_stats.txt`
  `config profile_preprocs: filename preprocs_stats.txt append`
- Print the top 10 preprocessors, based on highest average time
  `config profile_preprocs: print 10, sort avg_ticks`
- Print all preprocessors, sorted by number of checks
  `config profile_preprocs: print all, sort checks`

Output

Snort will print a table much like the following at exit.

Configuration line used to print the above table:

```
config profile_rules: \
    print 3, sort total_ticks
```


Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 110

03-02-2022-24064301-Hauptpost-0614



Preprocessor Profile Statistics (all)

Num	Preprocessor	Layer	Checks	Exits	Microsecs	Avg/Check	Pct of Caller	Pct of Total
1	ftptelnet_ftp	0	2697	2697	135720	50.32	0.20	0.20
2	detect	0	930237	930237	31645670	34.02	47.20	47.20
1	rule eval	1	1347969	1347969	26758596	19.85	84.56	39.91
1	rule tree eval	2	1669390	1669390	26605086	15.94	99.43	39.68
1	pcre	3	488652	488652	18994719	38.87	71.40	28.33
2	asn1	3	1	1	8	8.56	0.00	0.00
3	uricontent	3	647122	647122	2638614	4.08	9.92	3.94
4	content	3	1043099	1043099	3154396	3.02	11.86	4.70
5	ftpbounce	3	23	23	19	0.87	0.00	0.00
6	byte_jump	3	9007	9007	3321	0.37	0.01	0.00
7	byte_test	3	239015	239015	64401	0.27	0.24	0.10
8	icmp_seq	3	2	2	0	0.16	0.00	0.00
9	fragbits	3	65259	65259	10168	0.16	0.04	0.02
10	isdataat	3	5085	5085	757	0.15	0.00	0.00
11	flags	3	4147	4147	517	0.12	0.00	0.00
12	flowbits	3	2002630	2002630	212231	0.11	0.80	0.32
13	ack	3	4042	4042	261	0.06	0.00	0.00
14	flow	3	1347822	1347822	79002	0.06	0.30	0.12
15	icode	3	75538	75538	4280	0.06	0.02	0.01
16	itype	3	27009	27009	1524	0.06	0.01	0.00
17	icmp_id	3	41150	41150	1618	0.04	0.01	0.00
18	ip_proto	3	142625	142625	5004	0.04	0.02	0.01
19	ipopts	3	13690	13690	457	0.03	0.00	0.00
2	rtn eval	2	55836	55836	22763	0.41	0.09	0.03
2	mpse	1	492836	492836	4135697	8.39	13.07	6.17
3	frag3	0	76925	76925	1683797	21.89	2.51	2.51
1	frag3insert	1	70885	70885	434980	6.14	25.83	0.65
2	frag3rebuild	1	5419	5419	6280	1.16	0.37	0.01
4	dcerpc	0	127332	127332	2426830	19.06	3.62	3.62
5	s5	0	809682	809682	14195602	17.53	21.17	21.17
1	s5tcp	1	765281	765281	14128577	18.46	99.53	21.07
1	s5TcpState	2	742464	742464	13223585	17.81	93.59	19.72
1	s5TcpFlush	3	51987	51987	92918	1.79	0.70	0.14
1	s5TcpProcessRebuilt	4	47355	47355	14548497	307.22	15657.23	21.70
2	s5TcpBuildPacket	4	47360	47360	41711	0.88	44.89	0.06
2	s5TcpData	3	250035	250035	141490	0.57	1.07	0.21
1	s5TcpPktInsert	4	88173	88173	110136	1.25	77.84	0.16
2	s5TcpNewSess	2	60880	60880	81779	1.34	0.58	0.12
6	eventq	0	2089428	2089428	26690209	12.77	39.81	39.81
7	httpinspect	0	296030	296030	1862359	6.29	2.78	2.78
8	smtp	0	137653	137653	227982	1.66	0.34	0.34
9	decode	0	1057635	1057635	1162456	1.10	1.73	1.73
10	ftptelnet_telnet	0	175	175	175	1.00	0.00	0.00
11	sfportscan	0	881153	881153	518655	0.59	0.77	0.77
12	backorifice	0	35369	35369	4875	0.14	0.01	0.01
13	dns	0	16639	16639	1346	0.08	0.00	0.00
total	total	0	1018323	1018323	67046412	65.84	0.00	0.00

Figure 2.2: Preprocessor Profiling Example Output

The columns represent:

- Number (rank) - The number is indented for each layer. Layer 1 preprocessors are listed under their respective caller (and sorted similarly).
- Preprocessor Name
- Layer - When printing a specific number of preprocessors all subtasks info for a particular preprocessor is printed for each layer 0 preprocessor stat.
- Checks (number of times preprocessor decided to look at a packet, ports matched, app layer header was correct, etc)
- Exits (number of corresponding exits – just to verify code is instrumented correctly, should ALWAYS match Checks, unless an exception was trapped)
- CPU Ticks
- Avg Ticks per Check
- Percent of caller - For non layer 0 preprocessors, i.e. subroutines within preprocessors, this identifies the percent of the caller's ticks that is spent for this subtask.

1644470956460125853 19

Because of task swapping, non-instrumented code, and other factors, the Pct of Caller field will not add up to 100% of the caller's time. It does give a reasonable indication of how much relative time is spent within each subtask.

By default, this information will be printed to the console when Snort exits. You can use the "filename" option in snort.conf to specify a file where this will be written. If "append" is not specified, a new file will be created each time Snort is run. The filenames will have timestamps appended to them. These files will be found in the logging directory.

2.5.3 Packet Performance Monitoring (PPM)

PPM provides thresholding mechanisms that can be used to provide a basic level of latency control for snort. It does not provide a hard and fast latency guarantee but should in effect provide a good average latency control. Both rules and packets can be checked for latency. The action taken upon detection of excessive latency is configurable. The following sections describe configuration, sample output, and some implementation details worth noting.

To use PPM, you must build with the –enable-ppm or the –enable-sourcefire option to configure.

PPM is configured as follows:

```
# Packet configuration:
config ppm: max-pkt-time <micro-secs>, \
    fastpath-expensive-packets, \
    pkt-log, \
    debug-pkts

# Rule configuration:
config ppm: max-rule-time <micro-secs>, \
    threshold count, \
    suspend-expensive-rules, \
    suspend-timeout <seconds>, \
    rule-log [log] [alert]
```

Packets and rules can be configured separately, as above, or together in just one config ppm statement. Packet and rule monitoring is independent, so one or both or neither may be enabled.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 112

03-02-2022-24064301-Hauptpost-0615

Configuration

Packet Configuration Options

`max-pkt-time <micro-secs>`

- enables packet latency thresholding using 'micros-secs' as the limit.
- default is 0 (packet latency thresholding disabled)
- reasonable starting defaults: 100/250/1000 for 1G/100M/5M nets

`fastpath-expensive-packets`

- enables stopping further inspection of a packet if the max time is exceeded
- default is off

`pkt-log`

- enables logging packet event if packet exceeds max-pkt-time
- logging is to syslog or console depending upon snort configuration
- default is no logging

`debug-pkts`

- enables per packet timing stats to be printed after each packet
- default is off

Rule Configuration Options

`max-rule-time <micro-secs>`

- enables rule latency thresholding using 'micros-secs' as the limit.
- default is 0 (rule latency thresholding disabled)
- reasonable starting defaults: 100/250/1000 for 1G/100M/5M nets

`threshold <count>`

- sets the number of cumulative rule time excesses before disabling a rule
- default is 5

`suspend-expensive-rules`

- enables suspending rule inspection if the max rule time is exceeded
- default is off

`suspend-timeout <seconds>`

- rule suspension time in seconds
- default is 60 seconds
- set to zero to permanently disable expensive rules

1644709564601258531 9

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 113

```
rule-log [log] [alert]
```

- enables event logging output for rules
- default is no logging
- one or both of the options 'log' and 'alert' must be used with 'rule-log'
- the log option enables output to syslog or console depending upon snort configuration

Examples

Example 1: The following enables packet tracking:

```
config ppm: max-pkt-time 100
```

The following enables rule tracking:

```
config ppm: max-rule-time 50, threshold 5
```

1644470956460125853 19

If fastpath-expensive-packets or suspend-expensive-rules is not used, then no action is taken other than to increment the count of the number of packets that should be fastpath'd or the rules that should be suspended. A summary of this information is printed out when snort exits.

Example 2:

The following suspends rules and aborts packet inspection. These rules were used to generate the sample output that follows.

```
config ppm: \
    max-pkt-time 50, fastpath-expensive-packets, \
    pkt-log, debug-pkt

config ppm: \
    max-rule-time 50, threshold 5, suspend-expensive-rules, \
    suspend-timeout 300, rule-log log alert
```

Sample Snort Output

Sample Snort Startup Output

```
Packet Performance Monitor Config:
  ticks per usec  : 1600 ticks
  max packet time : 50 usecs
  packet action   : fastpath-expensive-packets
  packet logging  : log
  debug-pkts      : disabled

Rule Performance Monitor Config:
  ticks per usec  : 1600 ticks
  max rule time   : 50 usecs
  rule action     : suspend-expensive-rules
  rule threshold  : 5
  suspend timeout : 300 secs
  rule logging    : alert log
```


Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 114

03-02-2022-24064301-Hauptpost-0616

Sample Snort Run-time Output

```
...
PPM: Process-BeginPkt[61] caplen=60
PPM: Pkt[61] Used= 8.15385 usecs
PPM: Process-EndPkt[61]

PPM: Process-BeginPkt[62] caplen=342
PPM: Pkt[62] Used= 65.3659 usecs
PPM: Process-EndPkt[62]

PPM: Pkt-Event Pkt[63] used=56.0438 usecs, 0 rules, 1 nc-rules tested, packet fastpathed.
PPM: Process-BeginPkt[63] caplen=60
PPM: Pkt[63] Used= 8.394 usecs
PPM: Process-EndPkt[63]

PPM: Process-BeginPkt[64] caplen=60
PPM: Pkt[64] Used= 8.21764 usecs
PPM: Process-EndPkt[64]
...
```



Sample Snort Exit Output

```
Packet Performance Summary:
   max packet time       : 50 usecs
   packet events         : 1
   avg pkt time          : 0.633125 usecs
Rule Performance Summary:
   max rule time         : 50 usecs
   rule events           : 0
   avg nc-rule time      : 0.2675 usecs
```

Implementation Details

- Enforcement of packet and rule processing times is done after processing each rule. Latency control is not enforced after each preprocessor.
- This implementation is software based and does not use an interrupt driven timing mechanism and is therefore subject to the granularity of the software based timing tests. Due to the granularity of the timing measurements any individual packet may exceed the user specified packet or rule processing time limit. Therefore this implementation cannot implement a precise latency guarantee with strict timing guarantees. Hence the reason this is considered a best effort approach.
- Since this implementation depends on hardware based high performance frequency counters, latency thresholding is presently only available on Intel and PPC platforms.
- Time checks are made based on the total system time, not processor usage by Snort. This was a conscious design decision because when a system is loaded, the latency for a packet is based on the total system time, not just the processor time the Snort application receives. Therefore, it is recommended that you tune your thresholding to operate optimally when your system is under load.

2.6 Output Modules

Output modules are new as of version 1.6. They allow Snort to be much more flexible in the formatting and presentation of output to its users. The output modules are run when the alert or logging subsystems of Snort are called, after

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 115

the preprocessors and detection engine. The format of the directives in the config file is very similar to that of the preprocessors.

Multiple output plugins may be specified in the Snort configuration file. When multiple plugins of the same type (log, alert) are specified, they are stacked and called in sequence when an event occurs. As with the standard logging and alerting systems, output plugins send their data to /var/log/snort by default or to a user directed directory (using the -l command line switch).

Output modules are loaded at runtime by specifying the output keyword in the config file:

```
output <name>: <options>

output alert_syslog: log_auth log_alert
```

2.6.1 alert_syslog

This module sends alerts to the syslog facility (much like the -s command line switch). This module also allows the user to specify the logging facility and priority within the Snort config file, giving users greater flexibility in logging alerts.



Available Keywords

Facilities

- log_auth
- log_authpriv
- log_daemon
- log_local0
- log_local1
- log_local2
- log_local3
- log_local4
- log_local5
- log_local6
- log_local7
- log_user

Priorities

- log_emerg
- log_alert
- log_crit
- log_err
- log_warning
- log_notice
- log_info
- log_debug

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 116

03-02-2022-24064301-Hauptpost-0617

Options

- log_cons
- log_ndelay
- log_perror
- log_pid

Format

```
alert_syslog: \
    <facility> <priority> <options>
```



As WIN32 does not run syslog servers locally by default, a hostname and port can be passed as options. The default host is 127.0.0.1. The default port is 514.

```
output alert_syslog: \
    [host=<hostname[:<port>],] \
    <facility> <priority> <options>
```

Example

```
output alert_syslog: host=10.1.1.1:514, <facility> <priority> <options>
```



2.6.2 alert_fast

This will print Snort alerts in a quick one-line format to a specified output file. It is a faster alerting method than full alerts because it doesn't need to print all of the packet headers to the output file and because it logs to only 1 file.

Format

```
output alert_fast: [<filename> ["packet"] [<limit>]]
<limit> ::= <number>[('G'|'M'|K')]
```

- filename: the name of the log file. The default name is ¡logdir¿/alert. You may specify "stdout" for terminal output. The name may include an absolute or relative path.
- packet: this option will cause multiline entries with full packet headers to be logged. By default, only brief single-line entries are logged.
- limit: an optional limit on file size which defaults to 128 MB. The minimum is 1 KB. See 2.6.13 for more information.

Example

```
output alert_fast: alert.fast
```


Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 117

2.6.3 alert_full

This will print Snort alert messages with full packet headers. The alerts will be written in the default logging directory (/var/log/snort) or in the logging directory specified at the command line.

Inside the logging directory, a directory will be created per IP. These files will be decoded packet dumps of the packets that triggered the alerts. The creation of these files slows Snort down considerably. This output method is discouraged for all but the lightest traffic situations.

Format

```
output alert_full: [<filename> [<limit>]]
<limit> ::= <number>[('G'|'M'|K')]
```

- `filename`: the name of the log file. The default name is ¡logdir¿/alert. You may specify "stdout" for terminal output. The name may include an absolute or relative path.
- `limit`: an optional limit on file size which defaults to 128 MB. The minimum is 1 KB. See 2.6.13 for more information.



Example

```
output alert_full: alert.full
```

2.6.4 alert_unixsock

Sets up a UNIX domain socket and sends alert reports to it. External programs/processes can listen in on this socket and receive Snort alert and packet data in real time. This is currently an experimental interface.

Format

```
alert_unixsock
```

Example

```
output alert_unixsock
```

2.6.5 log_tcpdump

The log_tcpdump module logs packets to a tcpdump-formatted file. This is useful for performing post-process analysis on collected traffic with the vast number of tools that are available for examining tcpdump-formatted files.

Format

```
output log_tcpdump: [<filename> [<limit>]]
<limit> ::= <number>[('G'|'M'|K')]
```

- `filename`: the name of the log file. The default name is ¡logdir¿/snort.log. The name may include an absolute or relative path. A UNIX timestamp is appended to the filename.
- `limit`: an optional limit on file size which defaults to 128 MB. When a sequence of packets is to be logged, the aggregate size is used to test the rollover condition. See 2.6.13 for more information.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 118

03-02-2022-24064301-Hauptpost-0618

Example

```
output log_tcpdump: snort.log
```

2.6.6 database

This module from Jed Pickel sends Snort data to a variety of SQL databases. More information on installing and configuring this module can be found on the [91]incident.org web page. The arguments to this plugin are the name of the database to be logged to and a parameter list. Parameters are specified with the format parameter = argument. see Figure 2.3 for example usage.

Format

```
database: <log | alert>, <database type>, <parameter list>
```

The following parameters are available:



host - Host to connect to. If a non-zero-length string is specified, TCP/IP communication is used. Without a host name, it will connect using a local UNIX domain socket.

port - Port number to connect to at the server host, or socket filename extension for UNIX-domain connections.

dbname - Database name

user - Database username for authentication

password - Password used if the database demands password authentication

sensor_name - Specify your own name for this Snort sensor. If you do not specify a name, one will be generated automatically

encoding - Because the packet payload and option data is binary, there is no one simple and portable way to store it in a database. Blobs are not used because they are not portable across databases. So i leave the encoding option to you. You can choose from the following options. Each has its own advantages and disadvantages:

hex (default) - Represent binary data as a hex string.

Storage requirements - 2x the size of the binary
Searchability - very good
Human readability - not readable unless you are a true geek, requires post processing

base64 - Represent binary data as a base64 string.

Storage requirements - ~1.3x the size of the binary
Searchability - impossible without post processing
Human readability - not readable requires post processing

ascii - Represent binary data as an ASCII string. This is the only option where you will actually lose data. Non-ASCII Data is represented as a '.'. If you choose this option, then data for IP and TCP options will still be represented as hex because it does not make any sense for that data to be ASCII.

Storage requirements - slightly larger than the binary because some characters are escaped (&,<,>)
Searchability - very good for searching for a text string impossible if you want to search for binary
human readability - very good

detail - How much detailed data do you want to store? The options are:

full (default) - Log all details of a packet that caused an alert (including IP/TCP options and the payload)

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 119

```
output database: \
    log, mysql, dbname=snort user=snort host=localhost password=xyz
```

Figure 2.3: Database Output Plugin Configuration

fast - Log only a minimum amount of data. You severely limit the potential of some analysis applications if you choose this option, but this is still the best choice for some applications. The following fields are logged: `timestamp`, `signature`, `source ip`, `destination ip`, `source port`, `destination port`, `tcp flags`, and `protocol`)

Furthermore, there is a logging method and database type that must be defined. There are two logging types available, `log` and `alert`. Setting the type to log attaches the database logging functionality to the log facility within the program. If you set the type to log, the plugin will be called on the log output chain. Setting the type to alert attaches the plugin to the alert output chain within the program.

There are five database types available in the current version of the plugin. These are `mssql`, `mysql`, `postgresql`, `oracle`, and `odbc`. Set the type to match the database you are using.



⚠NOTE

The database output plugin does not have the ability to handle alerts that are generated by using the `tag` keyword. See section 3.7.5 for more details.

2.6.7 csv

The csv output plugin allows alert data to be written in a format easily importable to a database. The output fields and their order may be customized.

Format

```
output alert_csv: [<filename> [<format> [<limit>]]]
<format> ::= "default"|<list>
<list> ::= <field>(,<field>)*
<field> ::= "dst"|"src"|"ttl" ...
<limit> ::= <number>[('G'|'M'|K')]
```

- `filename`: the name of the log file. The default name is ¡logdir¿/alert.csv. You may specify "stdout" for terminal output. The name may include an absolute or relative path.
- `format`: The list of formatting options is below. If the formatting option is "default", the output is in the order of the formatting options listed.
  - `timestamp`
  - `sig_generator`
  - `sig_id`
  - `sig_rev`
  - `msg`
  - `proto`
  - `src`
  - `srcport`
  - `dst`

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 120

03-02-2022-24064301-Hauptpost-0619

- dstport
- ethsrc
- ethdst
- ethlen
- tcpflags
- tcpseq
- tcpack
- tcplen
- tcpwindow
- ttl
- tos
- id
- dgmlen
- iplen
- icmptype
- icmpcode
- icmpid
- icmpseq

- `limit`: an optional limit on file size which defaults to 128 MB. The minimum is 1 KB. See 2.6.13 for more information.



Example

```
output alert_csv: /var/log/alert.csv default

output alert_csv: /var/log/alert.csv timestamp, msg
```

2.6.8 unified

The unified output plugin is designed to be the fastest possible method of logging Snort events. The unified output plugin logs events in binary format, allowing another programs to handle complex logging mechanisms that would otherwise diminish the performance of Snort.

The name *unified* is a misnomer, as the unified output plugin creates two different files, an *alert* file, and a *log* file. The alert file contains the high-level details of an event (eg: IPs, protocol, port, message id). The log file contains the detailed packet information (a packet dump with the associated event ID). Both file types are written in a binary format described in *spo_unified.h*.

⚠NOTE

Files have the file creation time (in Unix Epoch format) appended to each file when it is created.

Format

```
output alert_unified: <base file name> [, <limit <file size limit in MB>]
output log_unified: <base file name> [, <limit <file size limit in MB>]
```


Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 121

Example

```
output alert_unified: snort.alert, limit 128
output log_unified: snort.log, limit 128
```

2.6.9 unified 2

The unified2 output plugin is a replacement for the unified output plugin. It has the same performance characteristics, but a slightly different logging format. See section 2.6.8 on unified logging for more information.

Unified2 can work in one of three modes, packet logging, alert logging, or true unified logging. Packet logging includes a capture of the entire packet and is specified with `log_unified2`. Likewise, alert logging will only log events and is specified with `alert_unified2`. To include both logging styles in a single, unified file, simply specify `unified2`.

When MPLS support is turned on, MPLS labels can be included in unified2 events. Use option `mpls_event_types` to enable this. If option `mpls_event_types` is not used, then MPLS labels will be not be included in unified2 events.



NOTE

By default, unified 2 files have the file creation time (in Unix Epoch format) appended to each file when it is created.

Format

```
output alert_unified2: \
    filename <base filename> [, <limit <size in MB>] [, nostamp] [, mpls_event_types]

output log_unified2: \
    filename <base filename> [, <limit <size in MB>] [, nostamp]

output unified2: \
    filename <base file name> [, <limit <size in MB>] [, nostamp] [, mpls_event_types]
```

Example

```
output alert_unified2: filename snort.alert, limit 128, nostamp
output log_unified2: filename snort.log, limit 128, nostamp
output unified2: filename merged.log, limit 128, nostamp
output unified2: filename merged.log, limit 128, nostamp, mpls_event_types
```

2.6.10 alert_prelude

NOTE

support to use alert_prelude is not built in by default. To use alert_prelude, snort must be built with the –enable-prelude argument passed to ./configure.

The alert_prelude output plugin is used to log to a Prelude database. For more information on Prelude, see `http://www.prelude-ids.o`

Format

```
output alert_prelude: \
```


Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 122

```
profile=<name of prelude profile> \
[ info=<priority number for info priority alerts>] \
[ low=<priority number for low priority alerts>] \
[ medium=<priority number for medium priority alerts>]
```

Example

```
output alert_prelude: profile=snort info=4 low=3 medium=2
```

2.6.11 log null

Sometimes it is useful to be able to create rules that will alert to certain types of traffic but will not cause packet log entries. In Snort 1.8.2, the log_null plugin was introduced. This is equivalent to using the -n command line option but it is able to work within a ruletype.

Format

```
output log_null
```

Example

```
output log_null  # like using snort -n

ruletype info {
    type alert
    output alert_fast: info.alert
    output log_null
}
```

2.6.12 alert_aruba_action

> **NOTE**
> Support to use alert_aruba_action is not built in by default. To use alert_aruba_action, snort must be built with the –enable-aruba argument passed to ./configure.

Communicates with an Aruba Networks wireless mobility controller to change the status of authenticated users. This allows Snort to take action against users on the Aruba controller to control their network privilege levels.

For more information on Aruba Networks access control, see `http://www.arubanetworks.com/`.

Format

```
output alert_aruba_action: \
    <controller address> <secrettype> <secret> <action>
```

The following parameters are required:

`controller address` - Aruba mobility controller address.

`secrettype` - Secret type, one of "sha1", "md5" or "cleartext".

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 123

16444709564601258531 9

secret - Authentication secret configured on the Aruba mobility controller with the "aaa xml-api client" configuration command, represented as a sha1 or md5 hash, or a cleartext password.

action - Action to apply to the source IP address of the traffic generating an alert.

blacklist - Blacklist the station by disabling all radio communication.

setrole:rolename - Change the user's role to the specified rolename.

Example

```
output alert_aruba_action: \
    10.3.9.6 cleartext foobar setrole:quarantine_role
```

2.6.13 Log Limits

This section pertains to logs produced by `alert_fast`, `alert_full`, `alert_csv`, and `log_tcpdump`. `unified` and `unified2` also may be given limits. Those limits are described in the respective sections.

When a configured limit is reached, the current log is closed and a new log is opened with a UNIX timestamp appended to the configured log name.

Limits are configured as follows:

```
<limit> ::= <number>[(<gb>|<mb>|<kb>)]
<gb> ::= 'G'|'g'
<mb> ::= 'M'|'m'
<kb> ::= 'K'|'k'
```

Rollover will occur at most once per second so if limit is too small for logging rate, limit will be exceeded. Rollover works correctly if snort is stopped/restarted.

16444709564601258531 9

2.7 Host Attribute Table

Starting with version 2.8.1, Snort has the capability to use information from an outside source to determine both the protocol for use with Snort rules, and IP-Frag policy (see section 2.2.1) and TCP Stream reassembly policies (see section 2.2.2). This information is stored in an attribute table, which is loaded at startup. The table is re-read during run time upon receipt of signal number 30.

Snort associates a given packet with its attribute data from the table, if applicable.

For rule evaluation, service information is used instead of the ports when the protocol metadata in the rule matches the service corresponding to the traffic. If the rule doesn't have protocol metadata, or the traffic doesn't have any matching service information, the rule relies on the port information.

⚠NOTE

To use a host attribute table, Snort must be configured with the –enable-targetbased flag.

2.7.1 Configuration Format

```
attribute_table filename <path to file>
```


Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 124

03-02-2022-24064301-Hauptpost-0621

2.7.2 Attribute Table File Format

The attribute table uses an XML format and consists of two sections, a mapping section, used to reduce the size of the file for common data elements, and the host attribute section. The mapping section is optional.

An example of the file format is shown below.

```
<SNORT_ATTRIBUTES>
    <ATTRIBUTE_MAP>
        <ENTRY>
            <ID>1</ID>
            <VALUE>Linux</VALUE>
        </ENTRY>
        <ENTRY>
            <ID>2</ID>
            <VALUE>ssh</VALUE>
        </ENTRY>
    </ATTRIBUTE_MAP>
    <ATTRIBUTE_TABLE>
        <HOST>
            <IP>192.168.1.234</IP>
            <OPERATING_SYSTEM>
                <NAME>
                    <ATTRIBUTE_ID>1</ATTRIBUTE_ID>
                    <CONFIDENCE>100</CONFIDENCE>
                </NAME>
                <VENDOR>
                    <ATTRIBUTE_VALUE>Red Hat</ATTRIBUTE_VALUE>
                    <CONFIDENCE>99</CONFIDENCE>
                </VENDOR>
                <VERSION>
                    <ATTRIBUTE_VALUE>2.6</ATTRIBUTE_VALUE>
                    <CONFIDENCE>98</CONFIDENCE>
                </VERSION>
                <FRAG_POLICY>linux</FRAG_POLICY>
                <STREAM_POLICY>linux</STREAM_POLICY>
            </OPERATING_SYSTEM>
            <SERVICES>
                <SERVICE>
                    <PORT>
                        <ATTRIBUTE_VALUE>22</ATTRIBUTE_VALUE>
                        <CONFIDENCE>100</CONFIDENCE>
                    </PORT>
                    <IPPROTO>
                        <ATTRIBUTE_VALUE>tcp</ATTRIBUTE_VALUE>
                        <CONFIDENCE>100</CONFIDENCE>
                    </IPPROTO>
                    <PROTOCOL>
                        <ATTRIBUTE_ID>2</ATTRIBUTE_ID>
                        <CONFIDENCE>100</CONFIDENCE>
                    </PROTOCOL>
                    <APPLICATION>
                        <ATTRIBUTE_VALUE>OpenSSH</ATTRIBUTE_VALUE>
                        <CONFIDENCE>100</CONFIDENCE>
                        <VERSION>
                            <ATTRIBUTE_VALUE>3.9p1</ATTRIBUTE_VALUE>
                            <CONFIDENCE>93</CONFIDENCE>
```


Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 125

16447095646012585319

```
                    </VERSION>
                </APPLICATION>
            </SERVICE>
            <SERVICE>
                <PORT>
                    <ATTRIBUTE_VALUE>2300</ATTRIBUTE_VALUE>
                    <CONFIDENCE>100</CONFIDENCE>
                </PORT>
                <IPPROTO>
                    <ATTRIBUTE_VALUE>tcp</ATTRIBUTE_VALUE>
                    <CONFIDENCE>100</CONFIDENCE>
                </IPPROTO>
                <PROTOCOL>
                    <ATTRIBUTE_VALUE>telnet</ATTRIBUTE_VALUE>
                    <CONFIDENCE>100</CONFIDENCE>
                </PROTOCOL>
                <APPLICATION>
                    <ATTRIBUTE_VALUE>telnet</ATTRIBUTE_VALUE>
                    <CONFIDENCE>50</CONFIDENCE>
                </APPLICATION>
            </SERVICE>
        </SERVICES>
        <CLIENTS>
            <CLIENT>
                <IPPROTO>
                    <ATTRIBUTE_VALUE>tcp</ATTRIBUTE_VALUE>
                    <CONFIDENCE>100</CONFIDENCE>
                </IPPROTO>
                <PROTOCOL>
                    <ATTRIBUTE_VALUE>http</ATTRIBUTE_VALUE>
                    <CONFIDENCE>91</CONFIDENCE>
                </PROTOCOL>
                <APPLICATION>
                    <ATTRIBUTE_VALUE>IE Http Browser</ATTRIBUTE_VALUE>
                    <CONFIDENCE>90</CONFIDENCE>
                    <VERSION>
                        <ATTRIBUTE_VALUE>6.0</ATTRIBUTE_VALUE>
                        <CONFIDENCE>89</CONFIDENCE>
                    </VERSION>
                </APPLICATION>
            </CLIENT>
        </CLIENTS>
    </HOST>
  </ATTRIBUTE_TABLE>
</SNORT_ATTRIBUTES>
```



164470956460125853 19

NOTE

With Snort 2.8.1, for a given host entry, the stream and IP frag information are both used. Of the service attributes, only the IP protocol (tcp, udp, etc), port, and protocol (http, ssh, etc) are used. The application and version for a given service attribute, and any client attributes are ignored. They will be used in a future release.

A DTD for verification of the Host Attribute Table XML file is provided with the snort packages.

The confidence metric may be used to indicate the validity of a given service or client application and its respective elements. That field is not currently used by Snort, but may be in future releases.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 126

03-02-2022-24064301-Hauptpost-0622

2.7.3 Attribute Table Example

In the example above, a host running Red Hat 2.6 is described. This host has an IP address of 192.168.1.234. On that host, TCP port 22 is ssh (running Open SSH), and TCP port 2300 is telnet.

The IP stack fragmentation and stream reassembly is mimicked by the "linux" configuration (see sections 2.2.1 and 2.2.2).

Attribute Table Affect on preprocessors

- Network Layer Preprocessors

  Each of the network layer preprocessors (frag3 and stream5) make use of the respective `FRAG_POLICY` and `STREAM_POLICY` in terms of how data is handled for reassembly for packets being received by that host.

- Application Layer Preprocessors

  The application layer preprocessors (HTTP, SMTP, FTP, Telnet, etc) make use of the `SERVICE` information for connections destined to that host on that port.

  For example, even if the telnet portion of the FTP/Telnet preprocessor is only configured to inspect port 23, Snort will inspect packets for a connection to 192.168.1.234 port 2300 as telnet.

  Conversely, if, for example, HTTP Inspect is configured to inspect traffic on port 2300, HTTP Inspect will NOT process the packets on a connection to 192.168.1.234 port 2300 because it is identified as telnet.

  Below is a list of the common services used by Snort's application layer preprocessors and Snort rules (see below).

http	ftp	ftp-data	telnet	smtp	ssh	tftp
dcerpc	netbios-dgm	netbios-ns	netbios-ssn	nntp	finger	sunrpc
dns	isakmp	mysql	oracle	cvs	shell	x11
imap	pop2	pop3	snmp			

16444709564601258531 9

Attribute Table Affect on rules

Similar to the application layer preprocessors, rules configured for specific ports that have a service metadata will be processed based on the service identified by the attribute table.

When both service metadata is present in the rule and in the connection, Snort uses the service rather than the port. If there are rules that use the service and other rules that do not but the port matches, Snort will ONLY inspect the rules that have the service that matches the connection.

The following few scenarios identify whether a rule will be inspected or not.

- Alert: Rule Has Service Metadata, Connection Service Matches

  The following rule will be inspected and alert on traffic to host 192.168.1.234 port 2300 because it is identified as telnet.

```
alert tcp any any -> any 23 (msg:"Telnet traffic"; flow:to_server,established;
sid:10000001; metadata: service telnet;)
```

- Alert: Rule Has Multiple Service Metadata, Connection Service Matches One of them

  The following rule will be inspected and alert on traffic to host 192.168.1.234 port 2300 because it is identified as telnet.

```
alert tcp any any -> any 23 (msg:"Telnet traffic"; flow:to_server,established;
sid:10000002; metadata: service telnet, service smtp;)
```


Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 127

- No Alert: Rule Has Service Metadata, Connection Service Does Not Match, Port Matches

  The following rule will NOT be inspected and NOT alert on traffic to host 192.168.1.234 port 2300 because that traffic is identified as telnet, but the service is ssh.

```
alert tcp any any -> any 2300 (msg:"SSH traffic"; flow:to_server,established;
sid:10000003; metadata: service ssh;)
```

- Alert: Rule Has No Service Metadata, Port Matches

  The following rule will be inspected and alert on traffic to host 192.168.1.234 port 2300 because the port matches.

```
alert tcp any any -> any 2300 (msg:"Port 2300 traffic"; flow:to_server,established;
sid:10000004;)
```

- Alert: Rule Has No Service Metadata, Packet has service + other rules with service

  The first rule will NOT be inspected and NOT alert on traffic to host 192.168.1.234 port 2300 because the service is identified as telnet and there are other rules with that service.

```
alert tcp any any -> any 2300 (msg:"Port 2300 traffic"; flow:to_server,established;
sid:10000005;)
alert tcp any any -> any 2300 (msg:"Port 2300 traffic"; flow:to_server,established;
sid:10000006; metadata: service telnet;)
```

- No Alert: Rule Has No Service Metadata, Port Does Not Match

  The following rule will NOT be inspected and NOT alert on traffic to host 192.168.1.234 port 2300 because the port does not match.

```
alert tcp any any -> any 23 (msg:"Port 23 traffic"; flow:to_server,established;
sid:10000007;)
```

16444709564601258531 9

2.8 Dynamic Modules

Dynamically loadable modules were introduced with Snort 2.6. They can be loaded via directives in `snort.conf` or via command-line options.

⚠NOTE

To disable use of dynamic modules, Snort must be configured with the `--disable-dynamicplugin` flag.

2.8.1 Format

```
<directive> <parameters>
```

2.8.2 Directives

Syntax	Description
`dynamicpreprocessor [ file <shared library path> | directory <directory of shared libraries> ]`	Tells snort to load the dynamic preprocessor shared library (if file is used) or all dynamic preprocessor shared libraries (if directory is used). Specify `file`, followed by the full or relative path to the shared library. Or, specify `directory`, followed by the full or relative path to a directory of preprocessor shared libraries. (Same effect as `--dynamic-preprocessor-lib` or `--dynamic-preprocessor-lib-dir` options). See chapter 4 for more information on dynamic preprocessor libraries.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 128

03-02-2022-24064301-Hauptpost-0623

dynamicengine [file <shared library path> | directory <directory of shared libraries>]	Tells snort to load the dynamic engine shared library (if file is used) or all dynamic engine shared libraries (if directory is used). Specify file, followed by the full or relative path to the shared library. Or, specify directory, followed by the full or relative path to a directory of preprocessor shared libraries. (Same effect as --dynamic-engine-lib or --dynamic-preprocessor-lib-dir options). See chapter 4 for more information on dynamic engine libraries.
dynamicdetection [file <shared library path> | directory <directory of shared libraries>]	Tells snort to load the dynamic detection rules shared library (if file is used) or all dynamic detection rules shared libraries (if directory is used). Specify file, followed by the full or relative path to the shared library. Or, specify directory, followed by the full or relative path to a directory of detection rules shared libraries. (Same effect as --dynamic-detection-lib or --dynamic-detection-lib-dir options). See chapter 4 for more information on dynamic detection rules libraries.

2.9 Reloading a Snort Configuration



Snort now supports reloading a configuration in lieu of restarting Snort in so as to provide seamless traffic inspection during a configuration change. A separate thread will parse and create a swappable configuration object while the main Snort packet processing thread continues inspecting traffic under the current configuration. When a swappable configuration object is ready for use, the main Snort packet processing thread will swap in the new configuration to use and will continue processing under the new configuration. Note that for some preprocessors, existing session data will continue to use the configuration under which they were created in order to continue with proper state for that session. All newly created sessions will, however, use the new configuration.

2.9.1 Enabling support

To enable support for reloading a configuration, add `--enable-reload` to configure when compiling.

There is also an ancillary option that determines how Snort should behave if any non-reloadable options are changed (see section 2.9.3 below). This option is enabled by default and the behavior is for Snort to restart if any non-reloadable options are added/modified/removed. To disable this behavior and have Snort exit instead of restart, add `--disable-reload-error-restart` in addition to `--enable-reload` to configure when compiling.

⚠NOTE

This functionality is not currently supported in Windows.

2.9.2 Reloading a configuration

First modify your snort.conf (the file passed to the `-c` option on the command line).

Then, to initiate a reload, send Snort a `SIGHUP` signal, e.g.

```
$ kill -SIGHUP <snort pid>
```

⚠NOTE

If reload support is not enabled, Snort will restart (as it always has) upon receipt of a SIGHUP.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 129

> **NOTE**
> An invalid configuration will still result in Snort fatal erroring, so you should test your new configuration before issuing a reload, e.g. `$ snort -c snort.conf -T`

2.9.3 Non-reloadable configuration options

There are a number of option changes that are currently non-reloadable because they require changes to output, startup memory allocations, etc. Modifying any of these options will cause Snort to restart (as a `SIGHUP` previously did) or exit (if `--disable-reload-error-restart` was used to configure Snort).

Reloadable configuration options of note:

- Adding/modifying/removing text rules and variables are reloadable.
- Adding/modifying/removing preprocessor configurations are reloadable (except as noted below).

Non-reloadable configuration options of note:

- Adding/modifying/removing shared objects via dynamicdetection, dynamicengine and dynamicpreprocessor are not reloadable, i.e. any new/modified/removed shared objects will require a restart.
- Any changes to output will require a restart.

Changes to the following options are not reloadable:

```
attribute_table
config alertfile
config asn1
config chroot
config daemon
config detection_filter
config flowbits_size
config interface
config logdir
config max_attribute_hosts
config nolog
config no_promisc
config pkt_count
config rate_filter
config read_bin_file
config response
config set_gid
config set_uid
config snaplen
config threshold
dynamicdetection
dynamicengine
dynamicpreprocessor
output
```

In certain cases, only some of the parameters to a config option or preprocessor configuration are not reloadable. Those parameters are listed below the relevant config option or preprocessor.

```
config ppm: max-rule-time <int>
 rule-log
config profile_rules
```


1644470956460125853 19

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 130

03-02-2022-24064301-Hauptpost-0624

```
 filename
 print
 sort
config profile_preprocs
 filename
 print
 sort
preprocessor dcerpc2
 memcap
preprocessor frag3_global
 max_frags
 memcap
 prealloc_frags
 prealloc_memcap
 disabled
preprocessor perfmonitor
 file
 snortfile
preprocessor sfportscan
 memcap
 logfile
 disabled
preprocessor stream5_global
 memcap
 max_tcp
 max_udp
 max_icmp
 track_tcp
 track_udp
 track_icmp
```



2.10 Multiple Configurations

Snort now supports multiple configurations based on VLAN Id or IP subnet within a single instance of Snort. This will allow administrators to specify multiple snort configuration files and bind each configuration to one or more VLANs or subnets rather than running one Snort for each configuration required. Each unique snort configuration file will create a new configuration instance within snort. VLANs/Subnets not bound to any specific configuration will use the default configuration. Each configuration can have different preprocessor settings and detection rules.

2.10.1 Creating Multiple Configurations

Default configuration for snort is specified using the existing -c option. A default configuration binds multiple vlans or networks to non-default configurations, using the following configuration line:

```
config binding: <path_to_snort.conf> vlan <vlanIdList>
config binding: <path_to_snort.conf> net <ipList>
```

path_to_snort.conf - Refers to the absolute or relative path to the snort.conf for specific configuration.

vlanIdList - Refers to the comma seperated list of vlandIds and vlanId ranges. The format for ranges is two vlanId separated by a "-". Spaces are allowed within ranges. Valid vlanId is any number in 0-4095 range. Negative vland Ids and alphanumeric are not supported.

ipList - Refers to ip subnets. Subnets can be CIDR blocks for IPV6 or IPv4. A maximum of 512 individual IPv4 or IPv6 addresses or CIDRs can be specified.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 131

NOTE
Vlan and Subnets can not be used in the same line. Configurations can be applied based on either Vlans or Subnets not both.

NOTE
Even though Vlan Ids 0 and 4095 are reserved, they are included as valid in terms of configuring Snort.

2.10.2 Configuration Specific Elements

Config Options

Generally config options defined within the default configuration are global by default i.e. their value applies to all other configurations. The following config options are specific to each configuration.

```
policy_id
policy_mode
policy_version
```

The following config options are specific to each configuration. If not defined in a configuration, the default values of the option (not the default configuration values) take effect.

```
config checksum_drop
config disable_decode_alerts
config disable_decode_drops
config disable_ipopt_alerts
config disable_ipopt_drops
config disable_tcpopt_alerts
config disable_tcpopt_drops
config disable_tcpopt_experimental_alerts
config disable_tcpopt_experimental_drops
config disable_tcpopt_obsolete_alerts
config disable_tcpopt_obsolete_drops
config disable_ttcp_alerts
config disable_tcpopt_ttcp_alerts
config disable_ttcp_drops
```

Rules

Rules are specific to configurations but only some parts of a rule can be customized for performance reasons. If a rule is not specified in a configuration then the rule will never raise an event for the configuration. A rule shares all parts of the rule options, including the general options, payload detection options, non-payload detection options, and post-detection options. Parts of the rule header can be specified differently across configurations, limited to:

```
Source IP address and port
Destination IP address and port
Action
```

A higher revision of a rule in one configuration will override other revisions of the same rule in other configurations.

Variables

Variables defined using "var", "portvar" and "ipvar" are specific to configurations. If the rules in a configuration use variables, those variables must be defined in that configuration.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 132

16444709564601258531 9

03-02-2022-24064301-HauPtPost-0625

Preprocessors

Preprocessors configurations can be defined within each vlan or subnet specific configuration. Options controlling specific preprocessor memory usage, through specific limit on memory usage or number of instances, are processed only in default policy. The options control total memory usage for a preprocessor across all policies. These options are ignored in non-default policies without raising an error. A preprocessor must be configured in default configuration before it can be configured in non-default configuration. This is required as some mandatory preprocessor configuration options are processed only in default configuration.

Events and Output

An unique policy id can be assigned by user, to each configuration using the following config line:

```
config policy_id: <id>
```

id - Refers to a 16-bit unsigned value. This policy id will be used to identify alerts from a specific configuration in the unified2 records.



⚠NOTE

If no policy id is specified, snort assigns 0 (zero) value to the configuration.

To enable vlanId logging in unified2 records the following option can be used.

```
output alert_unified2: vlan_event_types (alert logging only)
output unified2: filename <filename>, vlan_event_types (true unified logging)
```

filename - Refers to the absolute or relative filename.

vlan_event_types - When this option is set, snort will use unified2 event type 104 and 105 for IPv4 and IPv6 respectively.

⚠NOTE

Each event logged will have the vlanId from the packet if vlan headers are present otherwise 0 will be used.

2.10.3 How Configuration is applied?

Snort assigns every incoming packet to a unique configuration based on the following criteria. If VLANID is present, then the innermost VLANID is used to find bound configuration. If the bound configuration is the default configuration, then destination IP address is searched to the most specific subnet that is bound to a non-default configuration. The packet is assigned non-default configuration if found otherwise the check is repeated using source IP address. In the end, default configuration is used if no other matching configuration is found.

For addressed based configuration binding, this can lead to conflicts between configurations if source address is bound to one configuration and destination address is bound to another. In this case, snort will use the first configuration in the order of definition, that can be applied to the packet.

2.11 Active Response

Snort 2.9 includes a number of changes to better handle inline operation, including:

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 133

- a single mechanism for all responses
- fully encoded reset or icmp unreachable packets
- updated flexible response rule option
- updated react rule option
- added block and sblock rule actions

These changes are outlined below.

2.11.1 Enabling Active Response

This enables active responses (snort will send TCP RST or ICMP unreachable/port) when dropping a session.

```
./configure --enable-active-response / -DACTIVE_RESPONSE

preprocessor stream5_global: \
    max_active_responses <max_rsp>, \
    min_response_seconds <min_sec>

<max_rsp> ::= (0..25)
<min_sec> ::= (1..300)
```

Active responses will be encoded based on the triggering packet. TTL will be set to the value captured at session pickup.

1644470956460125853 19

2.11.2 Configure Sniping

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 (sent for react) is multiplied similarly. At most 1 ICMP unreachable is sent, if and only if attempts ¿ 0.

```
./configure --enable-active-response

config response: attempts <att>

<att> ::= (1..20)
```

2.11.3 Flexresp

Flexresp and flexresp2 are replaced with flexresp3.

* Flexresp is deleted; these features are no longer avaliable:

```
./configure --enable-flexresp / -DENABLE_RESPOND -DENABLE_RESPONSE
config flexresp: attempts 1
```

* Flexresp2 is deleted; these features are deprecated, non-functional, and will be deleted in a future release:

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 134

03-02-2022-24064301-Hauptpost-0626

```
./configure --enable-flexresp2 / -DENABLE_RESPOND -DENABLE_RESPONSE2

config flexresp2_interface: eth0
config flexresp2_attempts: 4
config flexresp2_memcap: 1000000
config flexresp2_rows: 1000
```

* Flexresp3 is new: the resp rule option keyword is used to configure active responses for rules that fire.

```
./configure --enable-flexresp3 / -DENABLE_RESPOND -DENABLE_RESPONSE3

alert tcp any any -> any 80 (content:"a"; resp:<resp_t>; sid:1;)
```

* resp_t includes all flexresp and flexresp2 options:

```
<resp_t> ::= \
    rst_snd | rst_rcv | rst_all | \
    reset_source | reset_dest | reset_both | icmp_net | \
    icmp_host | icmp_port | icmp_all
```



2.11.4 React

react is a rule option keyword that enables sending an HTML page on a session and then resetting it. This is built with:

```
./configure --enable-react / -DENABLE_REACT
```

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

```
config react: <block.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 selected message, which defaults to:

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 135

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

This is an example rule:

```
drop tcp any any -> any $HTTP_PORTS ( \
    content: "d"; msg:"Unauthorized Access Prohibited!"; \
    react: <react_opts>; sid:4;)

<react_opts> ::= [msg] [, <dep_opts>]
```

These options are deprecated:

```
<dep_opts> ::= [block|warn], [proxy <port#>]
```

The original version sent the web page to one end of the session only if the other end of the session was port 80 or the optional proxy port. The new version always sends the page to the client. If no page should be sent, a resp option can be used instead. The deprecated options are ignored.



2.11.5 Rule Actions

The block and sblock actions have been introduced as synonyms for drop and sdrop to help avoid confusion between packets dropped due to load (eg lack of available buffers for incoming packets) and packets blocked due to Snort's analysis.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 136

03-02-2022-24064301-Hauptpost-0627
03-02-2022-24064301-Hauptpost-0627

Chapter 3

Writing Snort Rules

3.1 The Basics


164447095646012585319

Snort uses a simple, lightweight rules description language that is flexible and quite powerful. There are a number of simple guidelines to remember when developing Snort rules that will help safeguard your sanity.

Most Snort rules are written in a single line. This was required in versions prior to 1.8. In current versions of Snort, rules may span multiple lines by adding a backslash \ to the end of the line.

Snort rules are divided into two logical sections, the rule header and the rule options. The rule header contains the rule's action, protocol, source and destination IP addresses and netmasks, and the source and destination ports information. The rule option section contains alert messages and information on which parts of the packet should be inspected to determine if the rule action should be taken.

Figure 3.1 illustrates a sample Snort rule.

The text up to the first parenthesis is the rule header and the section enclosed in parenthesis contains the rule options. The words before the colons in the rule options section are called option *keywords*.



Note that the rule options section is not specifically required by any rule, they are just used for the sake of making tighter definitions of packets to collect or alert on (or drop, for that matter).

All of the elements in that make up a rule must be true for the indicated rule action to be taken. When taken together, the elements can be considered to form a logical AND statement. At the same time, the various rules in a Snort rules library file can be considered to form a large logical OR statement.

3.2 Rules Headers

3.2.1 Rule Actions

The rule header contains the information that defines the who, where, and what of a packet, as well as what to do in the event that a packet with all the attributes indicated in the rule should show up. The first item in a rule is the rule

```
alert tcp any any -> 192.168.1.0/24 111 \
    (content:"|00 01 86 a5|"; msg:"mountd access";)
```

Figure 3.1: Sample Snort Rule

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 137

action. The rule action tells Snort what to do when it finds a packet that matches the rule criteria. There are 5 available default actions in Snort, alert, log, pass, activate, and dynamic. In addition, if you are running Snort in inline mode, you have additional options which include drop, reject, and sdrop.

1. alert - generate an alert using the selected alert method, and then log the packet
2. log - log the packet
3. pass - ignore the packet
4. activate - alert and then turn on another dynamic rule
5. dynamic - remain idle until activated by an activate rule , then act as a log rule
6. drop - block and log the packet
7. reject - block the packet, log it, and then send a TCP reset if the protocol is TCP or an ICMP port unreachable message if the protocol is UDP.
8. sdrop - block the packet but do not log it.



You can also define your own rule types and associate one or more output plugins with them. You can then use the rule types as actions in Snort rules.

This example will create a type that will log to just tcpdump:

```
ruletype suspicious
{
    type log
    output log_tcpdump: suspicious.log
}
```

This example will create a rule type that will log to syslog and a MySQL database:

```
ruletype redalert
{
      type alert
      output alert_syslog: LOG_AUTH LOG_ALERT
      output database: log, mysql, user=snort dbname=snort host=localhost
}
```

3.2.2 Protocols

The next field in a rule is the protocol. There are four protocols that Snort currently analyzes for suspicious behavior – TCP, UDP, ICMP, and IP. In the future there may be more, such as ARP, IGRP, GRE, OSPF, RIP, IPX, etc.

3.2.3 IP Addresses

The next portion of the rule header deals with the IP address and port information for a given rule. The keyword any may be used to define any address. Snort does not have a mechanism to provide host name lookup for the IP address fields in the config file. The addresses are formed by a straight numeric IP address and a CIDR[3] block. The CIDR block indicates the netmask that should be applied to the rule's address and any incoming packets that are tested against the rule. A CIDR block mask of /24 indicates a Class C network, /16 a Class B network, and /32 indicates a specific machine address. For example, the address/CIDR combination 192.168.1.0/24 would signify the block of addresses from 192.168.1.1 to 192.168.1.255. Any rule that used this designation for, say, the destination address would match on any address in that range. The CIDR designations give us a nice short-hand way to designate large address spaces with just a few characters.

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 138

03-02-2022-24064301-Hauptpost-0628

```
alert tcp !192.168.1.0/24 any -> 192.168.1.0/24 111 \
    (content:"|00 01 86 a5|"; msg:"external mountd access";)
```

Figure 3.2: Example IP Address Negation Rule

```
alert tcp ![192.168.1.0/24,10.1.1.0/24] any -> \
    [192.168.1.0/24,10.1.1.0/24] 111 (content:"|00 01 86 a5|"; \
    msg:"external mountd access";)
```

Figure 3.3: IP Address Lists

In Figure 3.1, the source IP address was set to match for any computer talking, and the destination address was set to match on the 192.168.1.0 Class C network.

There is an operator that can be applied to IP addresses, the negation operator. This operator tells Snort to match any IP address except the one indicated by the listed IP address. The negation operator is indicated with a !. For example, an easy modification to the initial example is to make it alert on any traffic that originates outside of the local net with the negation operator as shown in Figure 3.2.



This rule's IP addresses indicate any tcp packet with a source IP address not originating from the internal network and a destination address on the internal network.

You may also specify lists of IP addresses. An IP list is specified by enclosing a comma separated list of IP addresses and CIDR blocks within square brackets. For the time being, the IP list may not include spaces between the addresses. See Figure 3.3 for an example of an IP list in action.

3.2.4 Port Numbers

Port numbers may be specified in a number of ways, including any ports, static port definitions, ranges, and by negation. Any ports are a wildcard value, meaning literally any port. Static ports are indicated by a single port number, such as 111 for portmapper, 23 for telnet, or 80 for http, etc. Port ranges are indicated with the range operator :. The range operator may be applied in a number of ways to take on different meanings, such as in Figure 3.4.

Port negation is indicated by using the negation operator !. The negation operator may be applied against any of the other rule types (except any, which would translate to none, how Zen...). For example, if for some twisted reason you wanted to log everything except the X Windows ports, you could do something like the rule in Figure 3.5.

3.2.5 The Direction Operator

The direction operator -> indicates the orientation, or direction, of the traffic that the rule applies to. The IP address and port numbers on the left side of the direction operator is considered to be the traffic coming from the source

```
log udp any any -> 192.168.1.0/24 1:1024
```

log udp traffic coming from any port and destination ports ranging from 1 to 1024

```
log tcp any any -> 192.168.1.0/24 :6000
```

log tcp traffic from any port going to ports less than or equal to 6000

```
log tcp any :1024 -> 192.168.1.0/24 500:
```

log tcp traffic from privileged ports less than or equal to 1024 going to ports greater than or equal to 500

Figure 3.4: Port Range Examples

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 139

```
log tcp any any -> 192.168.1.0/24 !6000:6010
```

Figure 3.5: Example of Port Negation

```
log tcp !192.168.1.0/24 any <> 192.168.1.0/24 23
```

Figure 3.6: Snort rules using the Bidirectional Operator

host, and the address and port information on the right side of the operator is the destination host. There is also a bidirectional operator, which is indicated with a <> symbol. This tells Snort to consider the address/port pairs in either the source or destination orientation. This is handy for recording/analyzing both sides of a conversation, such as telnet or POP3 sessions. An example of the bidirectional operator being used to record both sides of a telnet session is shown in Figure 3.6.

Also, note that there is no <- operator. In Snort versions before 1.8.7, the direction operator did not have proper error checking and many people used an invalid token. The reason the <- does not exist is so that rules always read consistently.

1644470956460125853 19

3.2.6 Activate/Dynamic Rules

⚠NOTE

Activate and Dynamic rules are being phased out in favor of a combination of tagging (3.7.5) and flowbits (3.6.10).

Activate/dynamic rule pairs give Snort a powerful capability. You can now have one rule activate another when it's action is performed for a set number of packets. This is very useful if you want to set Snort up to perform follow on recording when a specific rule goes off. Activate rules act just like alert rules, except they have a *required* option field: activates. Dynamic rules act just like log rules, but they have a different option field: activated_by. Dynamic rules have a second required field as well, count.

Activate rules are just like alerts but also tell Snort to add a rule when a specific network event occurs. Dynamic rules are just like log rules except are dynamically enabled when the activate rule id goes off.

Put 'em together and they look like Figure 3.7.

These rules tell Snort to alert when it detects an IMAP buffer overflow and collect the next 50 packets headed for port 143 coming from outside $HOME_NET headed to $HOME_NET. If the buffer overflow happened and was successful, there's a very good possibility that useful data will be contained within the next 50 (or whatever) packets going to that same service port on the network, so there's value in collecting those packets for later analysis.

3.3 Rule Options

Rule options form the heart of Snort's intrusion detection engine, combining ease of use with power and flexibility. All Snort rule options are separated from each other using the semicolon (;) character. Rule option keywords are separated from their arguments with a colon (:) character.

```
activate tcp !$HOME_NET any -> $HOME_NET 143 (flags:PA; \
    content:"|E8C0FFFFFF|/bin"; activates:1;  \
    msg:"IMAP buffer overflow!";)
dynamic tcp !$HOME_NET any -> $HOME_NET 143 (activated_by:1; count:50;)
```

Figure 3.7: Activate/Dynamic Rule Example

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 140

03-02-2022-24064301-Hauptpost-0629

There are four major categories of rule options.

general These options provide information about the rule but do not have any affect during detection

payload These options all look for data inside the packet payload and can be inter-related

non-payload These options look for non-payload data

post-detection These options are rule specific triggers that happen after a rule has "fired."

3.4 General Rule Options

3.4.1 msg

The msg rule option tells the logging and alerting engine the message to print along with a packet dump or to an alert. It is a simple text string that utilizes the \ as an escape character to indicate a discrete character that might otherwise confuse Snort's rules parser (such as the semi-colon ; character).



Format

```
msg:"<message text>";
```

3.4.2 reference

The reference keyword allows rules to include references to external attack identification systems. The plugin currently supports several specific systems as well as unique URLs. This plugin is to be used by output plugins to provide a link to additional information about the alert produced.

Make sure to also take a look at `http://www.snort.org/pub-bin/sigs-search.cgi/` for a system that is indexing descriptions of alerts based on of the sid (See Section 3.4.4).

Table 3.1: Supported Systems

System	URL Prefix
bugtraq	http://www.securityfocus.com/bid/
cve	http://cve.mitre.org/cgi-bin/cvename.cgi?name=
nessus	http://cgi.nessus.org/plugins/dump.php3?id=
arachnids	(currently down) http://www.whitehats.com/info/IDS
mcafee	http://vil.nai.com/vil/content/v_
osvdb	http://osvdb.org/show/osvdb/
url	http://

Format

```
reference:<id system>, <id>; [reference:<id system>, <id>;]
```

Examples

```
alert tcp any any -> any 7070 (msg:"IDS411/dos-realaudio"; \
    flags:AP; content:"|fff4 fffd 06|"; reference:arachnids,IDS411;)

alert tcp any any -> any 21 (msg:"IDS287/ftp-wuftp260-venglin-linux"; \
```


Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 141

```
flags:AP; content:"|31c031db 31c9b046 cd80 31c031db|"; \
reference:arachnids,IDS287; reference:bugtraq,1387; \
reference:cve,CAN-2000-1574;)
```

3.4.3 gid

The gid keyword (generator id) is used to identify what part of Snort generates the event when a particular rule fires. For example gid 1 is associated with the rules subsystem and various gids over 100 are designated for specific preprocessors and the decoder. See etc/generators in the source tree for the current generator ids in use. Note that the gid keyword is optional and if it is not specified in a rule, it will default to 1 and the rule will be part of the general rule subsystem. To avoid potential conflict with gids defined in Snort (that for some reason aren't noted it etc/generators), it is recommended that values starting at 1,000,000 be used. For general rule writing, it is not recommended that the gid keyword be used. This option should be used with the sid keyword. (See section 3.4.4)

The file etc/gen-msg.map contains contains more information on preprocessor and decoder gids.

Format

```
gid:<generator id>;
```

Example

This example is a rule with a generator id of 1000001.

```
alert tcp any any -> any 80 (content:"BOB"; gid:1000001; sid:1; rev:1;)
```



3.4.4 sid

The sid keyword is used to uniquely identify Snort rules. This information allows output plugins to identify rules easily. This option should be used with the rev keyword. (See section 3.4.5)

- <100 Reserved for future use
- 100-999,999 Rules included with the Snort distribution
- >=1,000,000 Used for local rules

The file sid-msg.map contains a mapping of alert messages to Snort rule IDs. This information is useful when post-processing alert to map an ID to an alert message.

Format

```
sid:<snort rules id>;
```

Example

This example is a rule with the Snort Rule ID of 1000983.

```
alert tcp any any -> any 80 (content:"BOB"; sid:1000983; rev:1;)
```


Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 142

03-02-2022-24064301-Hauptpost-0630

3.4.5 rev

The `rev` keyword is used to uniquely identify revisions of Snort rules. Revisions, along with Snort rule id's, allow signatures and descriptions to be refined and replaced with updated information. This option should be used with the `sid` keyword. (See section 3.4.4)

Format

```
rev:<revision integer>;
```

Example

This example is a rule with the Snort Rule Revision of 1.

```
alert tcp any any -> any 80 (content:"BOB"; sid:1000983; rev:1;)
```

3.4.6 classtype

The `classtype` keyword is used to categorize a rule as detecting an attack that is part of a more general type of attack class. Snort provides a default set of attack classes that are used by the default set of rules it provides. Defining classifications for rules provides a way to better organize the event data Snort produces.

Format

```
classtype:<class name>;
```

Example

```
alert tcp any any -> any 25 (msg:"SMTP expn root"; flags:A+; \
    content:"expn root"; nocase; classtype:attempted-recon;)
```

Attack classifications defined by Snort reside in the `classification.config` file. The file uses the following syntax:

```
config classification:  <class name>,<class description>,<default priority>
```

These attack classifications are listed in Table 3.2. They are currently ordered with 4 default priorities. A priority of 1 (high) is the most severe and 4 (very low) is the least severe.

Table 3.2: Snort Default Classifications

Classtype	Description	Priority
attempted-admin	Attempted Administrator Privilege Gain	high
attempted-user	Attempted User Privilege Gain	high
inappropriate-content	Inappropriate Content was Detected	high
policy-violation	Potential Corporate Privacy Violation	high
shellcode-detect	Executable code was detected	high
successful-admin	Successful Administrator Privilege Gain	high
successful-user	Successful User Privilege Gain	high
trojan-activity	A Network Trojan was detected	high
unsuccessful-user	Unsuccessful User Privilege Gain	high
web-application-attack	Web Application Attack	high

1644470956460125853 19

Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 143

attempted-dos	Attempted Denial of Service	medium
attempted-recon	Attempted Information Leak	medium
bad-unknown	Potentially Bad Traffic	medium
default-login-attempt	Attempt to login by a default username and password	medium
denial-of-service	Detection of a Denial of Service Attack	medium
misc-attack	Misc Attack	medium
non-standard-protocol	Detection of a non-standard protocol or event	medium
rpc-portmap-decode	Decode of an RPC Query	medium
successful-dos	Denial of Service	medium
successful-recon-largescale	Large Scale Information Leak	medium
successful-recon-limited	Information Leak	medium
suspicious-filename-detect	A suspicious filename was detected	medium
suspicious-login	An attempted login using a suspicious username was detected	medium
system-call-detect	A system call was detected	medium
unusual-client-port-connection	A client was using an unusual port	medium
web-application-activity	Access to a potentially vulnerable web application	medium
icmp-event	Generic ICMP event	low
misc-activity	Misc activity	low
network-scan	Detection of a Network Scan	low
not-suspicious	Not Suspicious Traffic	low
protocol-command-decode	Generic Protocol Command Decode	low
string-detect	A suspicious string was detected	low
unknown	Unknown Traffic	low
tcp-connection	A TCP connection was detected	very low



Warnings

The `classtype` option can only use classifications that have been defined in `snort.conf` by using the `config classification` option. Snort provides a default set of classifications in `classification.config` that are used by the rules it provides.

3.4.7 priority

The `priority` tag assigns a severity level to rules. A `classtype` rule assigns a default priority (defined by the `config classification` option) that may be overridden with a priority rule. Examples of each case are given below.

Format

```
priority:<priority integer>;
```

Examples

```
alert tcp any any -> any 80 (msg:"WEB-MISC phf attempt"; flags:A+; \
    content:"/cgi-bin/phf"; priority:10;)

alert tcp any any -> any 80 (msg:"EXPLOIT ntpdx overflow"; \
    dsize:>128; classtype:attempted-admin; priority:10 );
```


Patent Owner, Centripetal Networks, LLC - Ex. 2014, p. 144