# Suricata User Guide

*Release 5.0.2*

**OISF**

**Apr 07, 2020**

# Contents


**1 What is Suricata** **1**
1.1 About the Open Information Security Foundation . . . 1

**2 Quickstart guide** **3**
2.1 Installation . . . 3
2.2 Basic setup . . . 3
2.3 Signatures . . . 4
2.4 Running Suricata . . . 4
2.5 Alerting . . . 5
2.6 EVE Json . . . 5

**3 Installation** **7**
3.1 Source . . . 7
3.2 Binary packages . . . 9
3.3 Advanced Installation . . . 9

**4 Upgrading** **11**
4.1 General instructions . . . 11
4.2 Upgrading 4.1 to 5.0 . . . 11

**5 Command Line Options** **13**
5.1 Unit Tests . . . 16

**6 Suricata Rules** **17**
6.1 Rules Format . . . 17
6.2 Meta Keywords . . . 22
6.3 IP Keywords . . . 25
6.4 TCP keywords . . . 29
6.5 UDP keywords . . . 32
6.6 ICMP keywords . . . 32
6.7 Payload Keywords . . . 35
6.8 Transformations . . . 53
6.9 Prefiltering Keywords . . . 55
6.10 Flow Keywords . . . 59
6.11 Bypass Keyword . . . 64
6.12 HTTP Keywords . . . 64
6.13 File Keywords . . . 85

6.14 DNS Keywords . . . 88
6.15 SSL/TLS Keywords . . . 89
6.16 SSH Keywords . . . 93
6.17 JA3 Keywords . . . 95
6.18 Modbus Keyword . . . 96
6.19 DNP3 Keywords . . . 99
6.20 ENIP/CIP Keywords . . . 101
6.21 FTP/FTP-DATA Keywords . . . 102
6.22 Kerberos Keywords . . . 103
6.23 SNMP keywords . . . 105
6.24 Base64 keywords . . . 106
6.25 SIP Keywords . . . 107
6.26 Generic App Layer Keywords . . . 110
6.27 Xbits Keyword . . . 111
6.28 Thresholding Keywords . . . 113
6.29 IP Reputation Keyword . . . 115
6.30 Datasets . . . 116
6.31 Lua Scripting . . . 119
6.32 Differences From Snort . . . 120

**7 Rule Management** **131**
7.1 Rule Management with Suricata-Update . . . 131
7.2 Adding Your Own Rules . . . 133
7.3 Rule Reloads . . . 134

**8 Making sense out of Alerts** **135**

**9 Performance** **137**
9.1 Runmodes . . . 137
9.2 Packet Capture . . . 141
9.3 Tuning Considerations . . . 143
9.4 Hyperscan . . . 145
9.5 High Performance Configuration . . . 147
9.6 Statistics . . . 153
9.7 Ignoring Traffic . . . 155
9.8 Packet Profiling . . . 157
9.9 Rule Profiling . . . 157
9.10 Tcmalloc . . . 158

**10 Configuration** **161**
10.1 Suricata.yaml . . . 161
10.2 Global-Thresholds . . . 209
10.3 Snort.conf to Suricata.yaml . . . 212
10.4 Multi Tenancy . . . 216
10.5 Dropping Privileges After Startup . . . 220

**11 Reputation** **221**
11.1 IP Reputation . . . 221

**12 Init Scripts** **225**

**13 Setting up IPS/inline for Linux** **227**
13.1 Setting up IPS with Netfilter . . . 227
13.2 Settings up IPS at Layer 2 . . . 231

**14 Setting up IPS/inline for Windows** **233**

**15 Output** **235**
15.1 EVE . . . 235
15.2 Lua Output . . . 269
15.3 Syslog Alerting Compatibility . . . 270
15.4 Custom http logging . . . 272
15.5 Custom tls logging . . . 273
15.6 Log Rotation . . . 273

**16 Lua support** **275**
16.1 Lua usage in Suricata . . . 275
16.2 Lua functions . . . 275

**17 File Extraction** **291**
17.1 Architecture . . . 291
17.2 Settings . . . 291
17.3 Output . . . 292
17.4 Rules . . . 293
17.5 MD5 . . . 294

**18 Public Data Sets** **297**

**19 Using Capture Hardware** **299**
19.1 Endace DAG . . . 299
19.2 Napatech . . . 300
19.3 Myricom . . . 306
19.4 eBPF and XDP . . . 307
19.5 Netmap . . . 317

**20 Interacting via Unix Socket** **321**
20.1 Introduction . . . 321
20.2 Commands in standard running mode . . . 322
20.3 Commands on the cmd prompt . . . 323
20.4 Pcap processing mode . . . 323
20.5 Build your own client . . . 324

**21 3rd Party Integration** **327**
21.1 Symantec SSL Visibility (BlueCoat) . . . 327

**22 Man Pages** **329**
22.1 Suricata . . . 329
22.2 Suricata Socket Control . . . 333
22.3 Suricata Control . . . 335
22.4 Suricata Control Filestore . . . 336

**23 Acknowledgements** **339**

**24 Licenses** **341**
24.1 GNU General Public License . . . 341
24.2 Creative Commons Attribution-NonCommercial 4.0 International Public License . . . 345
24.3 Suricata Source Code . . . 350
24.4 Suricata Documentation . . . 350

**Index** **351**

CHAPTER 1

# What is Suricata

Suricata is a high performance Network IDS, IPS and Network Security Monitoring engine. It is open source and owned by a community-run non-profit foundation, the Open Information Security Foundation (OISF). Suricata is developed by the OISF.

## 1.1 About the Open Information Security Foundation

The Open Information Security Foundation is a non-profit foundation organized to build community and to support open-source security technologies like Suricata, the world-class IDS/IPS engine.

### 1.1.1 License

The Suricata source code is licensed under version 2 of the *GNU General Public License*. This documentation is licensed under the *Creative Commons Attribution-NonCommercial 4.0 International Public License*.

CHAPTER 2

# Quickstart guide

This guide will give you a quick start to run Suricata and will focus only on the basics. For more details read through the different more specific chapters.

## 2.1 Installation

It's assumed that you run a recent Ubuntu release as the official PPA can be used for the installation.

Installation steps:

```
sudo add-apt-repository ppa:oisf/suricata-stable
sudo apt update
sudo apt install suricata jq
```

The dedicated PPA repository is added and after updating the index Suricata can be installed. It's recommended to install the tool `jq` as well to work with the EVE Json output which is used later in this guide.

For the installation on other systems or to use specific compile options see *Installation*.

When the installation is done you can check what version of Suricata you have running and with what options as well as the service state:

```
sudo suricata --build-info
sudo systemctl status suricata
```

## 2.2 Basic setup

You should check on which interface Suricata should be running and also the IP(s) of the interface:

```
ip a

2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group␣
↪default qlen 1000
link/ether 00:11:22:33:44:55 brd ff:ff:ff:ff:ff:ff
inet 10.0.0.23/24 brd 10.23.0.255 scope global noprefixroute enp1s0
```

Use that information to configure Suricata:

```
sudo vim /etc/suricata/suricata.yaml
```

There will be a lot of possible configuration options, we focus on the setup of the `HOME_NET` variable and the network interface configuration. The `HOME_NET` variable should include, in most scenarios, the IP you have configured on the interface you use to monitor and all the local networks in use. The default already includes the RFC 1918 networks. In this example `10.0.0.23` is already included within `10.0.0.0/8`. If no other networks are used the other predefined can be removed.

In this example the interface name is `enp1s0` so at the `af-packet` section the interface name needs to match. An example interface config might look like this:

Capture settings:

```
af-packet:
    - interface: enp1s0
      cluster-id: 99
      cluster-type: cluster_flow
      defrag: yes
      use-mmap: yes
      tpacket-v3: yes
```

This configuration uses the most recent recommended settings for the IDS runmode for basic setups. There are a lot of possible configuration options which are described in dedicated chapters and are especially relevant for high performance setups.

## 2.3 Signatures

As Suricata uses Signatures to trigger alerts it's necessary to install those and keep them updated. Signatures are also called rules, thus the name rule-files. With the tool `suricata-update` rules can be fetched, updated and managed to be provided for suricata.

In this guide we just run the default mode which fetches the ET Open ruleset:

```
sudo suricata-update
```

Afterwards the rules are installed at `/var/lib/suricata/rules` which is also the default at the config and uses the sole `suricata.rules` file.

## 2.4 Running Suricata

With the rules installed, Suricata can run properly and thus we restart it:

```
sudo systemctl restart suricata
```

To make sure Suricata is running check the Suricata log:

```
sudo tail /var/log/suricata/suricata.log
```

The last line should look like this:

```
<Notice> - all 4 packet processing threads, 4 management threads initialized, engine␣
↪started.
```

The amount of threads depends on the system and the configuration.

To see statistics the `stats.log` can be checked:

```
sudo tail -f /var/log/suricata/stats.log
```

Every 20 seconds by default it will show updated informations about the current state, like how many packets have been processed and what type of traffic was decoded.

## 2.5 Alerting

To test the IDS functionality of Suricata it's best to test with a signature. The signature with ID `2100498` from the ET Open ruleset is written specific for such test cases.

2100498:

```
alert ip any any -> any any (msg:"GPL ATTACK_RESPONSE id check returned root";␣
↪content:"uid=0|28|root|29|"; classtype:bad-unknown; sid:2100498; rev:7;␣
↪metadata:created_at 2010_09_23, updated_at 2010_09_23;)
```

The syntax and logic behind those signatures is covered in other chapters. This will alert on any IP traffic that has the content within its payload. This rule can be triggered quite easy. Before we trigger it, we will tail on the `fast.log` so we see the result.

Ruletrigger:

```
sudo tail -f /var/log/suricata/fast.log
curl http://testmyids.com/
```

The following output should now be seen in the log:

```
[1:2100498:7] GPL ATTACK_RESPONSE id check returned root [**] [Classification:␣
↪Potentially Bad Traffic] [Priority: 2] {TCP} 217.160.0.187:80 -> 10.0.0.23:41618
```

This should include the timestamp and the IP of your system.

## 2.6 EVE Json

The more advanced output is the EVE Json output which is explained in detail in *Eve JSON Output*. To see what this looks like it's recommended to use `jq` to parse the JSON output.

Alert:

```
sudo tail -f /var/log/suricata/eve.json | jq 'select(.event_type=="alert")'
```

This will also show much more details and meta-data that are related to the triggered alert.

Stats:

```
sudo tail -f /var/log/suricata/eve.json | jq 'select(.event_type=="stats")|.stats.
↪capture.kernel_packets'
```

This will only show the amount of packets processed.

CHAPTER 3

# Installation

Before Suricata can be used it has to be installed. Suricata can be installed on various distributions using binary packages: *Binary packages*.

For people familiar with compiling their own software, the Source method is recommended.

Advanced users can check the advanced guides, see *Advanced Installation*.

## 3.1 Source

Installing from the source distribution files gives the most control over the Suricata installation.

Basic steps:

```
tar xzvf suricata-5.0.0.tar.gz
cd suricata-5.0.0
./configure
make
make install
```

This will install Suricata into `/usr/local/bin/`, use the default configuration in `/usr/local/etc/suricata/` and will output to `/usr/local/var/log/suricata`

### 3.1.1 Common configure options

**`--disable-gccmarch-native`**
: Do not optimize the binary for the hardware it is built on. Add this flag if the binary is meant to be portable or if Suricata is to be used in a VM.

**`--prefix`**`=/usr/`
: Installs the Suricata binary into /usr/bin/. Default `/usr/local/`

**`--sysconfdir`**`=/etc`
: Installs the Suricata configuration files into /etc/suricata/. Default `/usr/local/etc/`

**--localstatedir**=/var
: Setups Suricata for logging into /var/log/suricata/. Default `/usr/local/var/log/suricata`

**--enable-lua**
: Enables Lua support for detection and output.

**--enable-geoip**
: Enables GeoIP support for detection.

### 3.1.2 Dependencies

For Suricata's compilation you'll need the following libraries and their development headers installed:

libpcap, libpcre, libmagic, zlib, libyaml

The following tools are required:

make gcc (or clang) pkg-config

For full features, also add:

libjansson, libnss, libgeoip, liblua5.1, libhiredis, libevent

Rust support:

rustc, cargo

Not every distro provides Rust packages yet. Rust can also be installed directly from the Rust project itself: https://www.rust-lang.org/en-US/install.html

#### Ubuntu/Debian

Minimal:

```
apt-get install libpcre3 libpcre3-dbg libpcre3-dev build-essential libpcap-dev   \
                libyaml-0-2 libyaml-dev pkg-config zlib1g zlib1g-dev \
                make libmagic-dev
```

Recommended:

```
apt-get install libpcre3 libpcre3-dbg libpcre3-dev build-essential libpcap-dev   \
                libnet1-dev libyaml-0-2 libyaml-dev pkg-config zlib1g zlib1g-dev \
                libcap-ng-dev libcap-ng0 make libmagic-dev libjansson-dev        \
                libnss3-dev libgeoip-dev liblua5.1-dev libhiredis-dev libevent-dev \
                python-yaml rustc cargo
```

Extra for iptables/nftables IPS integration:

```
apt-get install libnetfilter-queue-dev libnetfilter-queue1  \
                libnetfilter-log-dev libnetfilter-log1      \
                libnfnetlink-dev libnfnetlink0
```

For Rust support:

```
apt-get install rustc cargo
```

## 3.2 Binary packages

### 3.2.1 Ubuntu

For Ubuntu, the OISF maintains a PPA `suricata-stable` that always contains the latest stable release.

To use it:

```
sudo add-apt-repository ppa:oisf/suricata-stable
sudo apt-get update
sudo apt-get install suricata
```

### 3.2.2 Debian

In Debian 9 (Stretch) do:

```
apt-get install suricata
```

In Debian Jessie Suricata is out of date, but an updated version is in Debian Backports.

As root do:

```
echo "deb http://http.debian.net/debian jessie-backports main" > \
    /etc/apt/sources.list.d/backports.list
apt-get update
apt-get install suricata -t jessie-backports
```

### 3.2.3 Fedora

```
dnf install suricata
```

### 3.2.4 RHEL/CentOS

For RedHat Enterprise Linux 7 and CentOS 7 the EPEL repository can be used.

```
yum install epel-release
yum install suricata
```

## 3.3 Advanced Installation

Various installation guides for installing from GIT and for other operating systems are maintained at: https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Suricata_Installation

CHAPTER 4

# Upgrading

## 4.1 General instructions

Suricata can be upgraded by simply installing the new version to the same locations as the already installed version. When installing from source, this means passing the same `--prefix`, `--sysconfdir`, `--localstatedir` and `--datadir` options to `configure`.

```
$ suricata --build-info|grep -A 3 '\-\-prefix'
    --prefix                                    /usr
    --sysconfdir                                /etc
    --localstatedir                             /var
    --datarootdir                               /usr/share
```

### 4.1.1 Configuration Updates

New versions of Suricata will occationally include updated config files: `classification.config` and `reference.config`. Since the Suricata installation will not overwrite these if they exist, they should be manually updated. If there are no local modifications they can simply be overwritten by the ones Suricata supplies.

Major updates include new features, new default settings and often also remove features.

## 4.2 Upgrading 4.1 to 5.0

### 4.2.1 Major changes

- New protocols enabled by default: snmp (new config only)
- New protocols disabled by default: rdp, sip
- New defaults for protocols: nfs, smb, tftp, krb5 ntp are all enabled by default (new config only)
- VXLAN decoder enabled by default. To disable, set `decoder.vxlan.enabled` to `false`.

- HTTP LZMA support enabled by default. To disable, set `lzma-enabled` to `false` in each of the `libhtp` configurations in use.
- classification.config updated. ET 5.0 ruleset will use this.
- decoder event counters use 'decoder.event' as prefix now. This can be controlled using the `stats.decoder-events-prefix` setting.

### 4.2.2 Removals

- `dns-log`, the text dns log. Use EVE.dns instead.
- `file-log`, the non-EVE JSON file log. Use EVE.files instead.

See https://suricata-ids.org/about/deprecation-policy/

CHAPTER 5

# Command Line Options

Suricata's command line options:

**-h**
Display a brief usage overview.

**-V**
Displays the version of Suricata.

**-c** <path>
Path to configuration file.

**-T**
Test configuration.

**-v**
Increase the verbosity of the Suricata application logging by increasing the log level from the default. This option can be passed multiple times to further increase the verbosity.

- -v: INFO
- -vv: PERF
- -vvv: CONFIG
- -vvvv: DEBUG

This option will not decrease the log level set in the configuration file if it is already more verbose than the level requested with this option.

**-r** <path>
Run in pcap offline mode (replay mode) reading files from pcap file. If <path> specifies a directory, all files in that directory will be processed in order of modified time maintaining flow state between files.

**--pcap-file-continuous**
Used with the -r option to indicate that the mode should stay alive until interrupted. This is useful with directories to add new files and not reset flow state between files.

**--pcap-file-delete**
Used with the -r option to indicate that the mode should delete pcap files after they have been processed. This is

useful with pcap-file-continuous to continuously feed files to a directory and have them cleaned up when done. If this option is not set, pcap files will not be deleted after processing.

**-i** <interface>
After the -i option you can enter the interface card you would like to use to sniff packets from. This option will try to use the best capture method available. Can be used several times to sniff packets from several interfaces.

**--pcap[**=<device>]
Run in PCAP mode. If no device is provided the interfaces provided in the *pcap* section of the configuration file will be used.

**--af-packet[**=<device>]
Enable capture of packet using AF_PACKET on Linux. If no device is supplied, the list of devices from the af-packet section in the yaml is used.

**-q** <queue id>
Run inline of the NFQUEUE queue ID provided. May be provided multiple times.

**-s** <filename.rules>
With the -s option you can set a file with signatures, which will be loaded together with the rules set in the yaml.

**-S** <filename.rules>
With the -S option you can set a file with signatures, which will be loaded exclusively, regardless of the rules set in the yaml.

**-l** <directory>
With the -l option you can set the default log directory. If you already have the default-log-dir set in yaml, it will not be used by Suricata if you use the -l option. It will use the log dir that is set with the -l option. If you do not set a directory with the -l option, Suricata will use the directory that is set in yaml.

**-D**
Normally if you run Suricata on your console, it keeps your console occupied. You can not use it for other purposes, and when you close the window, Suricata stops running. If you run Suricata as daemon (using the -D option), it runs at the background and you will be able to use the console for other tasks without disturbing the engine running.

**--runmode** <runmode>
With the *–runmode* option you can set the runmode that you would like to use. This command line option can override the yaml runmode option.

Runmodes are: *workers*, *autofp* and *single*.

For more information about runmodes see *Runmodes* in the user guide.

**-F** <bpf filter file>
Use BPF filter from file.

**-k** [all|none]
Force (all) the checksum check or disable (none) all checksum checks.

**--user**=<user>
Set the process user after initialization. Overrides the user provided in the *run-as* section of the configuration file.

**--group**=<group>
Set the process group to group after initialization. Overrides the group provided in the *run-as* section of the configuration file.

**--pidfile** <file>
Write the process ID to file. Overrides the *pid-file* option in the configuration file and forces the file to be written when not running as a daemon.

**--init-errors-fatal**
Exit with a failure when errors are encountered loading signatures.

**--disable-detection**
Disable the detection engine.

**--dump-config**
Dump the configuration loaded from the configuration file to the terminal and exit.

**--build-info**
Display the build information the Suricata was built with.

**--list-app-layer-protos**
List all supported application layer protocols.

**--list-keywords**=[all|csv|<kword>]
List all supported rule keywords.

**--list-runmodes**
List all supported run modes.

**--set** <key>=<value>
Set a configuration value. Useful for overriding basic configuration parameters in the configuration. For example, to change the default log directory:

```
--set default-log-dir=/var/tmp
```

**--engine-analysis**
Print reports on analysis of different sections in the engine and exit. Please have a look at the conf parameter engine-analysis on what reports can be printed

**--unix-socket**=<file>
Use file as the Suricata unix control socket. Overrides the *filename* provided in the *unix-command* section of the configuration file.

**--pcap-buffer-size**=<size>
Set the size of the PCAP buffer (0 - 2147483647).

**--netmap[**=<device>]
Enable capture of packet using NETMAP on FreeBSD or Linux. If no device is supplied, the list of devices from the netmap section in the yaml is used.

**--pfring[**=<device>]
Enable PF_RING packet capture. If no device provided, the devices in the Suricata configuration will be used.

**--pfring-cluster-id** <id>
Set the PF_RING cluster ID.

**--pfring-cluster-type** <type>
Set the PF_RING cluster type (cluster_round_robin, cluster_flow).

**-d** <divert-port>
Run inline using IPFW divert mode.

**--dag** <device>
Enable packet capture off a DAG card. If capturing off a specific stream the stream can be select using a device name like “dag0:4”. This option may be provided multiple times read off multiple devices and/or streams.

**--napatech**
Enable packet capture using the Napatech Streams API.

**--erf-in**=<file>
Run in offline mode reading the specific ERF file (Endace extensible record format).

**--simulate-ips**
Simulate IPS mode when running in a non-IPS mode.

## 5.1 Unit Tests

Builtin unittests are only available if Suricata has been built with –enable-unittests.

Running unittests does not take a configuration file. Use -l to supply an output directory.

**-u**
Run the unit tests and exit. Requires that Suricata be compiled with *–enable-unittests*.

**-U, --unittest-filter**=REGEX
With the -U option you can select which of the unit tests you want to run. This option uses REGEX. Example of use: suricata -u -U http

**--list-unittests**
List all unit tests.

**--fatal-unittests**
Enables fatal failure on a unit test error. Suricata will exit instead of continuing more tests.

**--unittests-coverage**
Display unit test coverage report.

CHAPTER 6

# Suricata Rules

## 6.1 Rules Format

Signatures play a very important role in Suricata. In most occasions people are using existing rulesets.

The official way to install rulesets is described in *Rule Management with Suricata-Update*.

This Suricata Rules document explains all about signatures; how to read, adjust and create them.

A rule/signature consists of the following:

- The **action**, that determines what happens when the signature matches
- The **header**, defining the protocol, IP addresses, ports and direction of the rule.
- The **rule options**, defining the specifics of the rule.

An example of a rule is as follows:

drop tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"ET TROJAN Likely Bot Nick in IRC (USA +..)"; flow:established,to_server; flowbits:isset,is_proto_irc; content:"NICK "; pcre:"/NICK .*USA.*[0-9]{3,}/i"; reference:url,doc.emergingthreats.net/2008124; classtype:trojan-activity; sid:2008124; rev:2;)

In this example, red is the action, green is the header and blue are the options.

We will be using the above signature as an example throughout this section, highlighting the different parts of the signature. It is a signature taken from the database of Emerging Threats, an open database featuring lots of rules that you can freely download and use in your Suricata instance.

### 6.1.1 Action

drop tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"ET TROJAN Likely Bot Nick in IRC (USA +..)"; flow:established,to_server; flowbits:isset,is_proto_irc; content:"NICK "; pcre:"/NICK .*USA.*[0-9]{3,}/i"; reference:url,doc.emergingthreats.net/2008124; classtype:trojan-activity; sid:2008124; rev:2;)

For more information see *Action-order*.

### 6.1.2 Protocol

drop tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"ET TROJAN Likely Bot Nick in IRC (USA +..)"; flow:established,to_server; flowbits:isset,is_proto_irc; content:"NICK "; pcre:"/NICK .*USA.*[0-9]{3,}/i"; reference:url,doc.emergingthreats.net/2008124; classtype:trojan-activity; sid:2008124; rev:2;)

This keyword in a signature tells Suricata which protocol it concerns. You can choose between four basic protocols:

- tcp (for tcp-traffic)
- udp
- icmp
- ip (ip stands for 'all' or 'any')

There are also a few so-called application layer protocols, or layer 7 protocols you can pick from. These are:

- http
- ftp
- tls (this includes ssl)
- smb
- dns
- dcerpc
- ssh
- smtp
- imap
- modbus (disabled by default)
- dnp3 (disabled by default)
- enip (disabled by default)
- nfs (depends on rust availability)
- ikev2 (depends on rust availability)
- krb5 (depends on rust availability)
- ntp (depends on rust availability)
- dhcp (depends on rust availability)

The availability of these protocols depends on whether the protocol is enabled in the configuration file suricata.yaml.

If you have a signature with for instance a http protocol, Suricata makes sure the signature can only match if it concerns http-traffic.

### 6.1.3 Source and destination

drop tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"ET TROJAN Likely Bot Nick in IRC (USA +..)"; flow:established,to_server; flowbits:isset,is_proto_irc; content:"NICK "; pcre:"/NICK .*USA.*[0-9]{3,}/i"; reference:url,doc.emergingthreats.net/2008124; classtype:trojan-activity; sid:2008124; rev:2;)

*The first emphasized part is the source, the second is the destination (note the direction of the directional arrow).*

With source and destination, you specify the source of the traffic and the destination of the traffic, respectively. You can assign IP addresses, (both IPv4 and IPv6 are supported) and IP ranges. These can be combined with operators:

| Operator | Description |
|---|---|
| ../.. | IP ranges (CIDR notation) |
| ! | exception/negation |
| [.., ..] | grouping |

Normally, you would also make use of variables, such as `$HOME_NET` and `$EXTERNAL_NET`. The configuration file specifies the IP addresses these concern, and these settings will be used in place of the variables in you rules. See *Rule-vars* for more information.

For example:

| Example | Meaning |
|---|---|
| ! 1.1.1.1 | Every IP address but 1.1.1.1 |
| ![1.1.1.1, 1.1.1.2] | Every IP address but 1.1.1.1 and 1.1.1.2 |
| $HOME_NET | Your setting of HOME_NET in yaml |
| [$EXTERNAL_NET, !$HOME_NET] | EXTERNAL_NET and not HOME_NET |
| [10.0.0.0/24, !10.0.0.5] | 10.0.0.0/24 except for 10.0.0.5 |
| [. . . , [. . . .]] | |
| [. . . , ![. . . ..]] | |

**Warning:** If you set your configuration to something like this:

```
HOME_NET: any
EXTERNAL_NET: ! $HOME_NET
```

You can not write a signature using `$EXTERNAL_NET` because it stands for 'not any'. This is an invalid setting.

### 6.1.4 Ports (source and destination)

drop tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"ET TROJAN Likely Bot Nick in IRC (USA +..)"; flow:established,to_server; flowbits:isset,is_proto_irc; content:"NICK "; pcre:"/NICK .*USA.*[0-9]{3,}/i"; reference:url,doc.emergingthreats.net/2008124; classtype:trojan-activity; sid:2008124; rev:2;)

*The first emphasized part is the source, the second is the destination (note the direction of the directional arrow).*

Traffic comes in and goes out through ports. Different ports have different port numbers. For example, the default port for HTTP is 80 while 443 is typically the port for HTTPS. Note, however, that the port does not dictate which protocol is used in the communication. Rather, it determines which application is receiving the data.

The ports mentioned above are typically the destination ports. Source ports, i.e. the application that sent the packet, typically get assigned a random port by the operating system. When writing a rule for your own HTTP service, you would typically write `any -> 80`, since that would mean any packet from any source port to your HTTP application (running on port 80) is matched.

In setting ports you can make use of special operators as well, like described above. Signs like:

| Operator | Description |
|---|---|
| : | port ranges |
| ! | exception/negation |
| [.., ..] | grouping |

For example:

| Example | Meaning |
|---|---|
| [80, 81, 82] | port 80, 81 and 82 |
| [80: 82] | Range from 80 till 82 |
| [1024: ] | From 1024 till the highest port-number |
| !80 | Every port but 80 |
| [80:100,!99] | Range from 80 till 100 but 99 excluded |
| [1:80,![2,4]] | Range from 1-80, except ports 2 and 4 |
| [.., [..,..]] | |

### 6.1.5 Direction

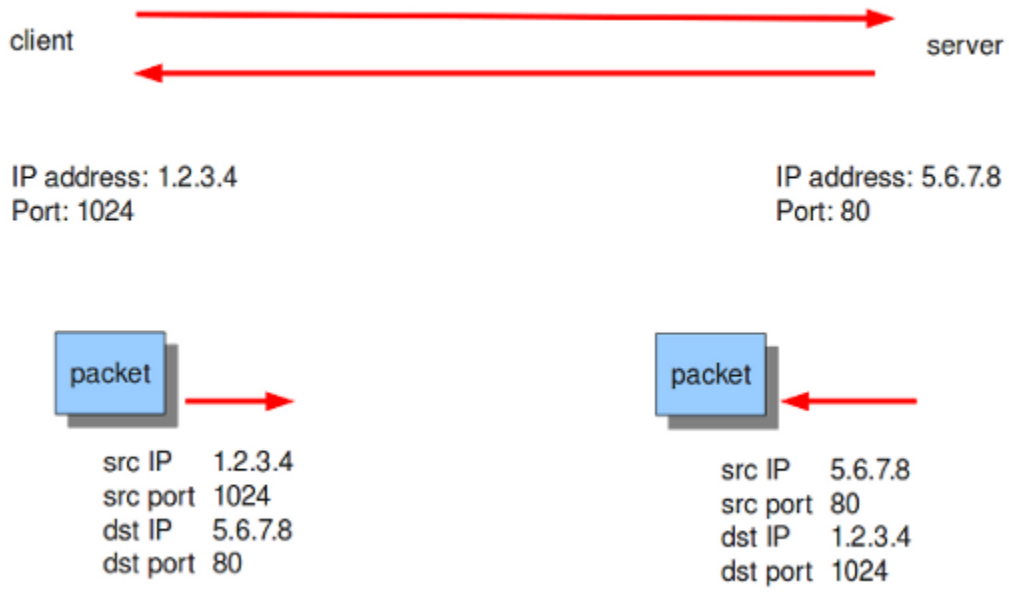
drop tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"ET TROJAN Likely Bot Nick in IRC (USA +..)"; flow:established,to_server; flowbits:isset,is_proto_irc; content:"NICK "; pcre:"/NICK .*USA.*[0-9]{3,}/i"; reference:url,doc.emergingthreats.net/2008124; classtype:trojan-activity; sid:2008124; rev:2;)

The direction tells in which way the signature has to match. Nearly every signature has an arrow to the right (->). This means that only packets with the same direction can match. However, it is also possible to have a rule match both ways (<>):

```
source -> destination
source <> destination  (both directions)
```

**Warning:** There is no 'reverse' style direction, i.e. there is no <-.

The following example illustrates this. Say, there is a client with IP address 1.2.3.4 and port 1024, and a server with IP address 5.6.7.8, listening on port 80 (typically HTTP). The client sends a message to the server, and the server replies with its answer.



Now, let's say we have a rule with the following header:

```
alert tcp 1.2.3.4 1024 -> 5.6.7.8 80
```

Only the first packet will be matched by this rule, as the direction specifies that we do not match on the response packet.

### 6.1.6 Rule options

The rest of the rule consists of options. These are enclosed by parenthesis and separated by semicolons. Some options have settings (such as `msg`), which are specified by the keyword of the option, followed by a colon, followed by the settings. Others have no settings, and are simply the keyword (such as `nocase`):

```
<keyword>: <settings>;
<keyword>;
```

Rule options have a specific ordering and changing their order would change the meaning of the rule.

---

**Note:** The characters `;` and `"` have special meaning in the Suricata rule language and must be escaped when used in a rule option value. For example:

```
msg:"Message with semicolon\;";
```

As a consequence, you must also escape the backslash, as it functions as an escape character.

---

The rest of this chapter in the documentation documents the use of the various keywords.

Some generic details about keywords follow.

#### Modifier Keywords

Some keywords function act as modifiers. There are two types of modifiers.

- The older style **'content modifiers'** look back in the rule, e.g.:

  ```
  alert http any any -> any any (content:"index.php"; http_uri; sid:1;)
  ```

  In the above example the pattern 'index.php' is modified to inspect the HTTP uri buffer.

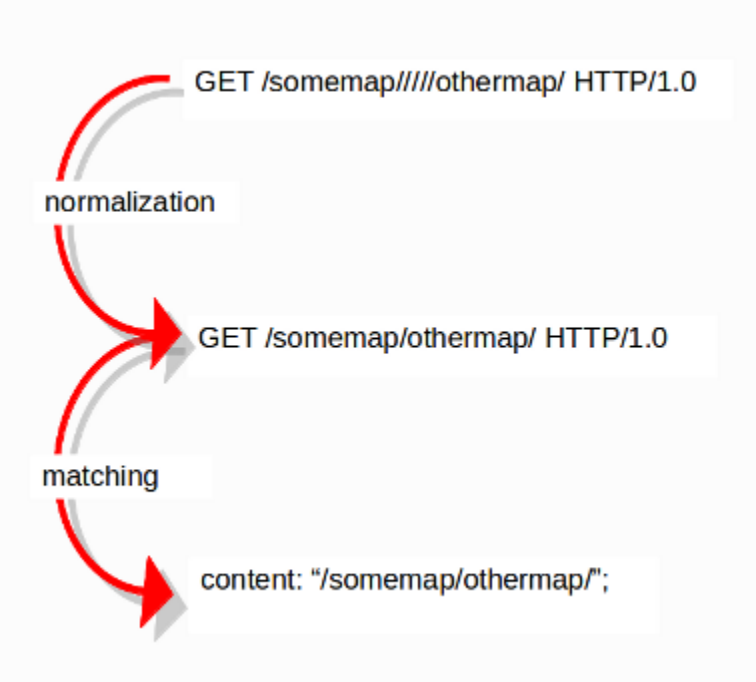
- The more recent type is called the **'sticky buffer'**. It places the buffer name first and all keywords following it apply to that buffer, for instance:

  ```
  alert http any any -> any any (http_response_line; content:"403 Forbidden"; sid:1;
  ↪)
  ```

  In the above example the pattern '403 Forbidden' is inspected against the HTTP response line because it follows the `http_response_line` keyword.

#### Normalized Buffers

A packet consists of raw data. HTTP and reassembly make a copy of those kinds of packets data. They erase anomalous content, combine packets etcetera. What remains is a called the 'normalized buffer':




Because the data is being normalized, it is not what it used to be; it is an interpretation. Normalized buffers are: all HTTP-keywords, reassembled streams, TLS-, SSL-, SSH-, FTP- and dcerpc-buffers.

Note that there are some exceptions, e.g. the `http_raw_uri` keyword. See *http.uri and http.uri.raw* for more information.

## 6.2 Meta Keywords

Meta-settings have no effect on Suricata's inspection; they do have an effect on the way Suricata reports events.

### 6.2.1 msg (message)

The keyword msg gives textual information about the signature and the possible alert.

The format of msg is:

```
msg: "some description";
```

Examples:

```
msg:"ATTACK-RESPONSES 403 Forbidden";
msg:"ET EXPLOIT SMB-DS DCERPC PnP bind attempt";
```

To continue the example of the previous chapter, this is the keyword in action in an actual rule:

drop tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"ET TROJAN Likely Bot Nick in IRC (USA +..)"; flow:established,to_server; flowbits:isset,is_proto_irc; content:"NICK "; pcre:"/NICK .*USA.*[0-9]{3,}/i"; reference:url,doc.emergingthreats.net/2008124; classtype:trojan-activity; sid:2008124; rev:2;)

**Tip:** It is convention to make the first part of the signature uppercase and show the class of the signature.

It is also convention that `msg` is made the first keyword in the signature.

**Note:** The following characters must be escaped inside the msg: `;` `\` `"`

### 6.2.2 sid (signature ID)

The keyword sid gives every signature its own id. This id is stated with a number. The format of sid is:

```
sid:123;
```

Example of sid in a signature:

drop tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"ET TROJAN Likely Bot Nick in IRC (USA +..)"; flow:established,to_server; flowbits:isset,is_proto_irc; content:"NICK "; pcre:"/NICK .*USA.*[0-9]{3,}/i"; reference:url,doc.emergingthreats.net/2008124; classtype:trojan-activity; sid:2008124; rev:2;)

**Tip:** It is convention that the signature `sid` is provided as the last keyword (or second-to-last if there is a `rev`) of the signature.

### 6.2.3 rev (revision)

The sid keyword is almost every time accompanied by rev. Rev represents the version of the signature. If a signature is modified, the number of rev will be incremented by the signature writers. The format of rev is:

```
rev:123;
```

Example of rev in a signature:

drop tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"ET TROJAN Likely Bot Nick in IRC (USA +..)"; flow:established,to_server; flowbits:isset,is_proto_irc; content:"NICK "; pcre:"/NICK .*USA.*[0-9]{3,}/i"; reference:url,doc.emergingthreats.net/2008124; classtype:trojan-activity; sid:2008124; rev:2;)

**Tip:** It is a convention that sid comes before rev, and both are the last of all keywords.

### 6.2.4 gid (group ID)

The gid keyword can be used to give different groups of signatures another id value (like in sid). Suricata uses by default gid 1. It is possible to modify this. It is not usual that it will be changed, and changing it has no technical implications. You can only notice it in the alert.

Example of gid in an alert of fast.log. In the part [1:2008124:2], 1 is the gid (2008124 is the sid and 2 the rev).

10/15/09-03:30:10.219671 [**] [1:2008124:2] ET TROJAN Likely Bot Nick in IRC (USA +..) [**] [Classification: A Network Trojan was Detected] [Priority: 3] {TCP} 192.168.1.42:1028 -> 72.184.196.31:6667

### 6.2.5 classtype

The classtype keyword gives information about the classification of rules and alerts. It consists of a short name, a long name and a priority. It can tell for example whether a rule is just informational or is about a hack etcetera. For each classtype, the classification.config has a priority which will be used in the rule.

Example classtype definition:

```
config classification: web-application-attack,Web Application Attack,1
config classification: not-suspicious,Not Suspicious Traffic,3
```

Now when we have defined this in the configuration, we can use the classtypes in our rules. A rule with classtype web-application-attack will be assigned a priority of 1 and the alert will contain 'Web Application Attack':

| classtype | Alert | Priority |
|---|---|---|
| web-application-attack | Web Application Attack | 1 |
| not-suspicious | Not Suspicious Traffic | 3 |

Our continuing example has also a classtype, this one of trojan-activity:

drop tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"ET TROJAN Likely Bot Nick in IRC (USA +..)"; flow:established,to_server; flowbits:isset,is_proto_irc; content:"NICK "; pcre:"/NICK .*USA.*[0-9]{3,}/i"; reference:url,doc.emergingthreats.net/2008124; classtype:trojan-activity; sid:2008124; rev:2;)

**Tip:** It is a convention that classtype comes before sid and rev and after the rest of the keywords.

### 6.2.6 reference

The reference keywords direct to places where information about the signature and about the problem the signature tries to address, can be found. The reference keyword can appear multiple times in a signature. This keyword is meant for signature-writers and analysts who investigate why a signature has matched. It has the following format:

```
reference: type, reference
```

A typical reference to www.info.com would be:

```
reference: url, www.info.com
```

However, there are also several systems that can be used as a reference. A commonly known example is the CVE-database, that assigns numbers to vulnerabilities. To prevent you from typing the same URL over and over again, you can use something like this:

```
reference: cve, CVE-2014-1234
```

This would make a reference to http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-1234. All reference types are defined in the reference.config configuration file.

Our continuing example also has a reference:

drop tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"ET TROJAN Likely Bot Nick in IRC (USA +..)"; flow:established,to_server; flowbits:isset,is_proto_irc; content:"NICK "; pcre:"/NICK .*USA.*[0-9]{3,}/i"; reference:url,doc.emergingthreats.net/2008124; classtype:trojan-activity; sid:2008124; rev:2;)

### 6.2.7 priority

The priority keyword comes with a mandatory numeric value which can range from 1 till 255. The numbers 1 to 4 are most often used. Signatures with a higher priority will be examined first. The highest priority is 1. Normally signatures have already a priority through classtype. This can be overruled with the keyword priority. The format of priority is:

```
priority:1;
```

### 6.2.8 metadata

The metadata keyword allows additional, non-functional information to be added to the signature. While the format is free-form, it is recommended to stick to key, value pairs as Suricata can include these in eve alerts. The format is:

```
metadata: key value;
metadata: key value, key value;
```

### 6.2.9 target

The target keyword allows the rules writer to specify which side of the alert is the target of the attack. If specified, the alert event is enhanced to contain information about source and target.

The format is:

```
target:[src_ip|dest_ip]
```

If the value is src_ip then the source IP in the generated event (src_ip field in JSON) is the target of the attack. If target is set to dest_ip then the target is the destination IP in the generated event.

## 6.3 IP Keywords

### 6.3.1 ttl

The ttl keyword is used to check for a specific IP time-to-live value in the header of a packet. The format is:

```
ttl:<number>
```

For example:

```
ttl:10;
```

At the end of the ttl keyword you can enter the value on which you want to match. The Time-to-live value determines the maximal amount of time a packet can be in the Internet-system. If this field is set to 0, then the packet has to be destroyed. The time-to-live is based on hop count. Each hop/router the packet passes subtracts one of the packet TTL counter. The purpose of this mechanism is to limit the existence of packets so that packets can not end up in infinite routing loops.

Example of the ttl keyword in a rule:

alert ip $EXTERNAL_NET any -> $HOME_NET any (msg:”GPL MISC 0 ttl”; ttl:0; reference:url,support.microsoft.com/default.aspx?scid=kb#-#-EN-US#-#-q138268; reference:url,www.isi.edu/in-notes/rfc1122.txt; classtype:misc-activity; sid:2101321; rev:9;)

### 6.3.2 ipopts

With the ipopts keyword you can check if a specific IP option is set. Ipopts has to be used at the beginning of a rule. You can only match on one option per rule. There are several options on which can be matched. These are:

| IP Option | Description |
|---|---|
| rr | Record Route |
| eol | End of List |
| nop | No Op |
| ts | Time Stamp |
| sec | IP Security |
| esec | IP Extended Security |
| lsrr | Loose Source Routing |
| ssrr | Strict Source Routing |
| satid | Stream Identifier |
| any | any IP options are set |

Format of the ipopts keyword:

```
ipopts: <name>
```

For example:

```
ipopts: lsrr;
```

Example of ipopts in a rule:

alert ip $EXTERNAL_NET any -> $HOME_NET any (msg:"GPL MISC source route ssrr"; ipopts:ssrr; reference:arachnids,422; classtype:bad-unknown; sid:2100502; rev:3;)

### 6.3.3 sameip

Every packet has a source IP-address and a destination IP-address. It can be that the source IP is the same as the destination IP. With the sameip keyword you can check if the IP address of the source is the same as the IP address of the destination. The format of the sameip keyword is:

```
sameip;
```

Example of sameip in a rule:

alert ip any any -> any any (msg:"GPL SCAN same SRC/DST"; sameip; reference:bugtraq,2666; reference:cve,1999-0016; reference:url,www.cert.org/advisories/CA-1997-28.html; classtype:bad-unknown; sid:2100527; rev:9;)

### 6.3.4 ip_proto

With the ip_proto keyword you can match on the IP protocol in the packet-header. You can use the name or the number of the protocol. You can match for example on the following protocols:

```
 1     ICMP        Internet Control Message
 6     TCP         Transmission Control Protocol
17     UDP         User Datagram
47     GRE         General Routing Encapsulation
50     ESP         Encap Security Payload for IPv6
```

(continues on next page)

(continued from previous page)

```
51      AH          Authentication Header for Ipv6
58      IPv6-ICMP   ICMP for Ipv6
```

For the complete list of protocols and their numbers see http://en.wikipedia.org/wiki/List_of_IP_protocol_numbers

Example of ip_proto in a rule:

alert ip any any -> any any (msg:"GPL MISC IP Proto 103 PIM"; ip_proto:103; reference:bugtraq,8211; reference:cve,2003-0567; classtype:non-standard-protocol; sid:2102189; rev:4;)

The named variant of that example would be:

```
ip_proto:PIM
```

### 6.3.5 id

With the id keyword, you can match on a specific IP ID value. The ID identifies each packet sent by a host and increments usually with one with each packet that is being send. The IP ID is used as a fragment identification number. Each packet has an IP ID, and when the packet becomes fragmented, all fragments of this packet have the same ID. In this way, the receiver of the packet knows which fragments belong to the same packet. (IP ID does not take care of the order, in that case offset is used. It clarifies the order of the fragments.)

Format of id:

```
id:<number>;
```

Example of id in a rule:

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"ET DELETED F5 BIG-IP 3DNS TCP Probe 1"; id: 1; dsize: 24; flags: S,12; content:"|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|"; window: 2048; reference:url,www.f5.com/f5products/v9intro/index.html; reference:url,doc.emergingthreats.net/2001609; classtype:misc-activity; sid:2001609; rev:13;)

### 6.3.6 geoip

The geoip keyword enables (you) to match on the source, destination or source and destination IPv4 addresses of network traffic, and to see to which country it belongs. To be able to do this, Suricata uses the GeoIP2 API of MaxMind.

The syntax of geoip:

```
geoip: src,RU;
geoip: both,CN,RU;
geoip: dst,CN,RU,IR;
geoip: both,US,CA,UK;
geoip: any,CN,IR;
```

So, you can see you can use the following to make clear on which direction you would like to match

| Option | Description |
|---|---|
| both | Both directions have to match with the given geoip(s) |
| any | One of the directions has to match with the given geoip(s). |
| dest | If the destination matches with the given geoip. |
| src | The source matches with the given geoip. |

The keyword only supports IPv4. As it uses the GeoIP2 API of MaxMind, libmaxminddb must be compiled in. You must download and install the GeoIP2 or GeoLite2 database editions desired. Visit the MaxMind site at https://dev.maxmind.com/geoip/geoip2/geolite2/ for details.

You must also supply the location of the GeoIP2 or GeoLite2 database file on the local system in the YAML-file configuration (for example):

```
geoip-database: /usr/local/share/GeoIP/GeoLite2-Country.mmdb
```

### 6.3.7 fragbits (IP fragmentation)

With the fragbits keyword, you can check if the fragmentation and reserved bits are set in the IP header. The fragbits keyword should be placed at the beginning of a rule. Fragbits is used to modify the fragmentation mechanism. During routing of messages from one Internet module to the other, it can occur that a packet is bigger than the maximal packet size a network can process. In that case, a packet can be send in fragments. This maximum of the packet size is called Maximal Transmit Unit (MTU).

You can match on the following bits:

```
M - More Fragments
D - Do not Fragment
R - Reserved Bit
```

Matching on this bits can be more specified with the following modifiers:

```
+         match on the specified bits, plus any others
*         match if any of the specified bits are set
!         match if the specified bits are not set
```

Format:

```
fragbits:[*+!]<[MDR]>;
```

Example of fragbits in a rule:

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"ET EXPLOIT Invalid non-fragmented packet with fragment offset>0"; fragbits: M; fragoffset: >0; reference:url,doc.emergingthreats.net/bin/view/Main/2001022; classtype:bad-unknown; sid:2001022; rev:5; metadata:created_at 2010_07_30, updated_at 2010_07_30;)

### 6.3.8 fragoffset

With the fragoffset keyword you can match on specific decimal values of the IP fragment offset field. If you would like to check the first fragments of a session, you have to combine fragoffset 0 with the More Fragment option. The fragmentation offset field is convenient for reassembly. The id is used to determine which fragments belong to which packet and the fragmentation offset field clarifies the order of the fragments.

You can use the following modifiers:

```
<       match if the value is smaller than the specified value
>       match if the value is greater than the specified value
!       match if the specified value is not present
```

Format of fragoffset:

```
fragoffset:[!|<|>]<number>;
```

Example of fragoffset in a rule:

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"ET EXPLOIT Invalid non-fragmented packet with fragment offset>0"; fragbits: M; fragoffset: >0; reference:url,doc.emergingthreats.net/bin/view/Main/2001022; classtype:bad-unknown; sid:2001022; rev:5; metadata:created_at 2010_07_30, updated_at 2010_07_30;)

### 6.3.9 tos

The tos keyword can match on specific decimal values of the IP header TOS field. The tos keyword can be have a value from 0 - 255. This field of the IP header has been updated by rfc2474 to include functionality for Differentiated services.

Format of tos:

```
tos:[!]<number>;
```

Example of tos in a rule:

alert ip any any -> any any (msg:"Differentiated Services Codepoint: Class Selector 1 (8)"; flow:established; tos:8; classtype:not-suspicious; sid:2600115; rev:1;)

Example of tos with negated values:

alert ip any any -> any any (msg:"TGI HUNT non-DiffServ aware TOS setting"; flow:established,to_server; tos:!0; tos:!8; tos:!16; tos:!24; tos:!32; tos:!40; tos:!48; tos:!56; threshold:type limit, track by_src, seconds 60, count 1; classtype:bad-unknown; sid:2600124; rev:1;)

## 6.4 TCP keywords

### 6.4.1 seq

The seq keyword can be used in a signature to check for a specific TCP sequence number. A sequence number is a number that is generated practically at random by both endpoints of a TCP-connection. The client and the server both create a sequence number, which increases with one with every byte that they send. So this sequence number is different for both sides. This sequence number has to be acknowledged by both sides of the connection. Through sequence numbers, TCP handles acknowledgement, order and retransmission. Its number increases with every data-byte the sender has send. The seq helps keeping track of to what place in a data-stream a byte belongs. If the SYN flag is set at 1, than the sequence number of the first byte of the data is this number plus 1 (so, 2).

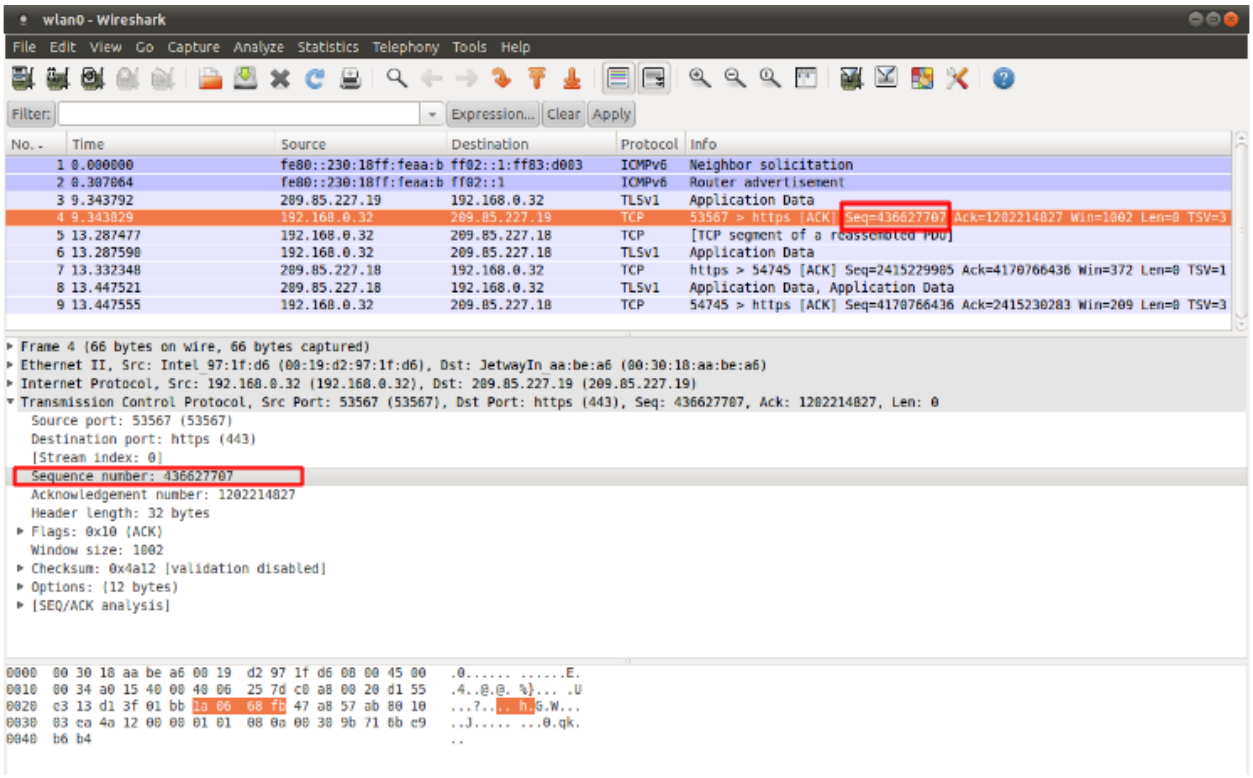
Example:

```
seq:0;
```

Example of seq in a signature:

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"GPL SCAN NULL"; flow:stateless; ack:0; flags:0; seq:0; reference:arachnids,4; classtype:attempted-recon; sid:2100623; rev:7;)

Example of seq in a packet (Wireshark):




### 6.4.2 ack

The ack is the acknowledgement of the receipt of all previous (data)-bytes send by the other side of the TCP-connection. In most occasions every packet of a TCP connection has an ACK flag after the first SYN and a ack-number which increases with the receipt of every new data-byte. The ack keyword can be used in a signature to check for a specific TCP acknowledgement number.

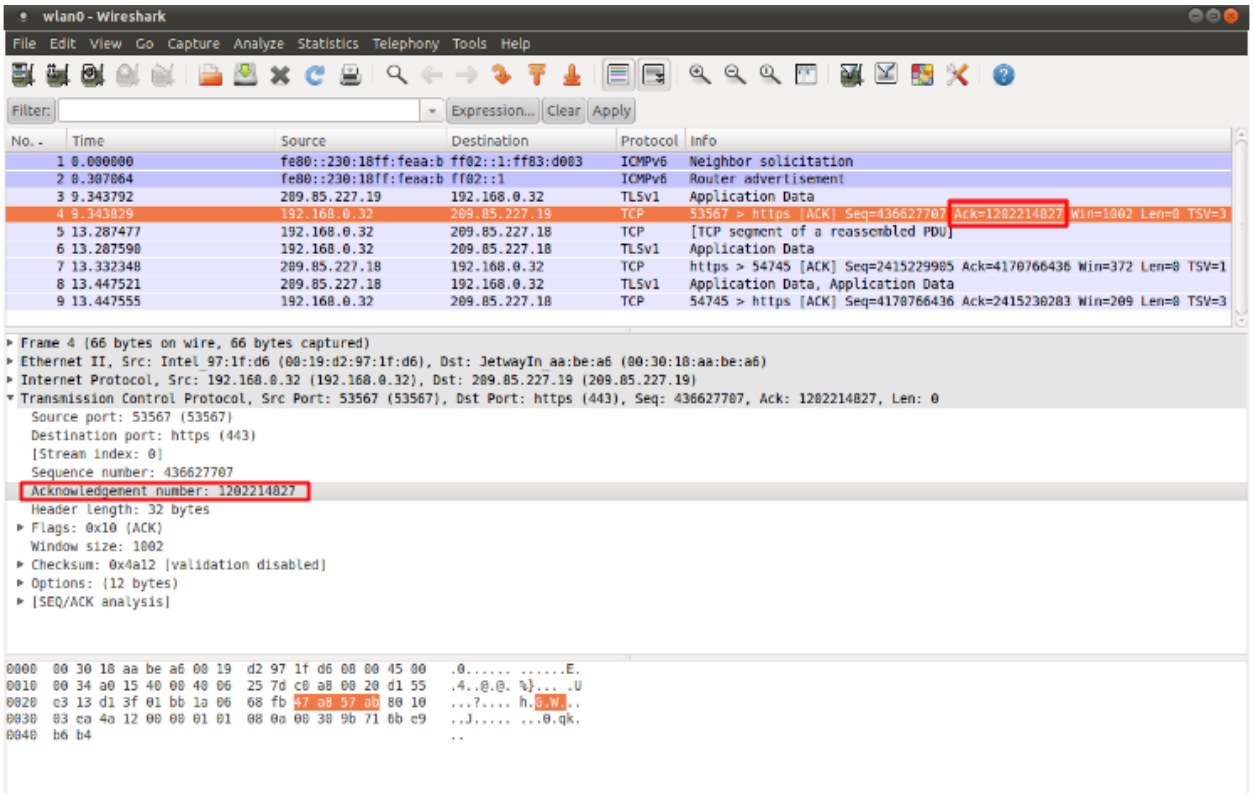
Format of ack:

```
ack:1;
```

Example of ack in a signature:

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"GPL SCAN NULL"; flow:stateless; ack:0; flags:0; seq:0; reference:arachnids,4; classtype:attempted-recon; sid:2100623; rev:7;)

Example of ack in a packet (Wireshark):




### 6.4.3 window

The window keyword is used to check for a specific TCP window size. The TCP window size is a mechanism that has control of the data-flow. The window is set by the receiver (receiver advertised window size) and indicates the amount of bytes that can be received. This amount of data has to be acknowledged by the receiver first, before the sender can send the same amount of new data. This mechanism is used to prevent the receiver from being overflowed by data. The value of the window size is limited and can be 2 to 65.535 bytes. To make more use of your bandwidth you can use a bigger TCP-window.

The format of the window keyword:

```
window:[!]<number>;
```

Example of window in a rule:

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"GPL DELETED typot trojan traffic"; flow:stateless; flags:S,12; window:55808; reference:mcafee,100406; classtype:trojan-activity; sid:2182; rev:8;)

### 6.4.4 tcp.mss

Match on the TCP MSS option value. Will not match if the option is not present.

The format of the keyword:

```
tcp.mss:<min>-<max>;
tcp.mss:[<|>]<number>;
tcp.mss:<value>;
```

Example rule:

alert tcp $EXTERNAL_NET any -> $HOME_NET any (flow:stateless; flags:S,12; tcp.mss:<536; sid:1234; rev:5;)

### 6.4.5 tcp.hdr

Sticky buffer to match on the whole TCP header.

Example rule:

alert tcp $EXTERNAL_NET any -> $HOME_NET any (flags:S,12; tcp.hdr; content:"|02 04|"; offset:20; byte_test:2,<,536,0,big,relative; sid:1234; rev:5;)

This example starts looking after the fixed portion of the header, so into the variable sized options. There it will look for the MSS option (type 2, option len 4) and using a byte_test determine if the value of the option is lower than 536. The *tcp.mss* option will be more efficient, so this keyword is meant to be used in cases where no specific keyword is available.

## 6.5 UDP keywords

### 6.5.1 udp.hdr

Sticky buffer to match on the whole UDP header.

Example rule:

alert udp any any -> any any (udp.hdr; content:"|00 08|"; offset:4; depth:2; sid:1234; rev:5;)

This example matches on the length field of the UDP header. In this case the length of 8 means that there is no payload. This can also be matched using *dsize:0;*.

## 6.6 ICMP keywords

ICMP (Internet Control Message Protocol) is a part of IP. IP at itself is not reliable when it comes to delivering data (datagram). ICMP gives feedback in case problems occur. It does not prevent problems from happening, but helps in understanding what went wrong and where. If reliability is necessary, protocols that use IP have to take care of reliability themselves. In different situations ICMP messages will be send. For instance when the destination is unreachable, if there is not enough buffer-capacity to forward the data, or when a datagram is send fragmented when it should not be, etcetera. More can be found in the list with message-types.

There are four important contents of a ICMP message on which can be matched with corresponding ICMP-keywords. These are: the type, the code, the id and the sequence of a message.

### 6.6.1 itype

The itype keyword is for matching on a specific ICMP type (number). ICMP has several kinds of messages and uses codes to clarify those messages. The different messages are distinct by different names, but more important by numeric values. For more information see the table with message-types and codes.

The format of the itype keyword:

```
itype:min<>max;
itype:[<|>]<number>;
```

Example This example looks for an ICMP type greater than 10:

```
itype:>10;
```

Example of the itype keyword in a signature:

alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"GPL SCAN Broadscan Smurf Scanner"; dsize:4; icmp_id:0; icmp_seq:0; itype:8; classtype:attempted-recon; sid:2100478; rev:4;)

The following lists all ICMP types known at the time of writing. A recent table can be found at the website of IANA

| ICMP Type | Name |
|---|---|
| 0 | Echo Reply |
| 3 | Destination Unreachable |
| 4 | Source Quench |
| 5 | Redirect |
| 6 | Alternate Host Address |
| 8 | Echo |
| 9 | Router Advertisement |
| 10 | Router Solicitation |
| 11 | Time Exceeded |
| 12 | Parameter Problem |
| 13 | Timestamp |
| 14 | Timestamp Reply |
| 15 | Information Request |
| 16 | Information Reply |
| 17 | Address Mask Request |
| 18 | Address Mask Reply |
| 30 | Traceroute |
| 31 | Datagram Conversion Error |
| 32 | Mobile Host Redirect |
| 33 | IPv6 Where-Are-You |
| 34 | IPv6 I-Am-Here |
| 35 | Mobile Registration Request |
| 36 | Mobile Registration Reply |
| 37 | Domain Name Request |
| 38 | Domain Name Reply |
| 39 | SKIP |
| 40 | Photuris |
| 41 | Experimental mobility protocols such as Seamoby |

### 6.6.2 icode

With the icode keyword you can match on a specific ICMP code. The code of a ICMP message clarifies the message. Together with the ICMP-type it indicates with what kind of problem you are dealing with. A code has a different purpose with every ICMP-type.

The format of the icode keyword:

```
icode:min<>max;
icode:[<|>]<number>;
```

Example: This example looks for an ICMP code greater than 5:

```
icode:>5;
```

Example of the icode keyword in a rule:

alert icmp $HOME_NET any -> $EXTERNAL_NET any (msg:"GPL MISC Time-To-Live Exceeded in Transit"; icode:0; itype:11; classtype:misc-activity; sid:2100449; rev:7;)

The following lists the meaning of all ICMP types. When a code is not listed, only type 0 is defined and has the meaning of the ICMP code, in the table above. A recent table can be found at the website of IANA

| ICMP Code | ICMP Type | Description |
|---|---|---|
| 3 | 0 | Net Unreachable |
| | 1 | Host Unreachable |
| | 2 | Protocol Unreachable |
| | 3 | Port Unreachable |
| | 4 | Fragmentation Needed and Don't Fragment was Set |
| | 5 | Source Route Failed |
| | 6 | Destination Network Unknown |
| | 7 | Destination Host Unknown |
| | 8 | Source Host Isolated |
| | 9 | Communication with Destination Network is Administratively Prohibited |
| | 10 | Communication with Destination Host is Administratively Prohibited |
| | 11 | Destination Network Unreachable for Type of Service |
| | 12 | Destination Host Unreachable for Type of Service |
| | 13 | Communication Administratively Prohibited |
| | 14 | Host Precedence Violation |
| | 15 | Precedence cutoff in effect |
| 5 | 0 | Redirect Datagram for the Network (or subnet) |
| | 1 | Redirect Datagram for the Host |
| | 2 | Redirect Datagram for the Type of Service and Network |
| | 3 | Redirect Datagram for the Type of Service and Host |
| 9 | 0 | Normal router advertisement |
| | 16 | Doesn't route common traffic |
| 11 | 0 | Time to Live exceeded in Transit |
| | 1 | Fragment Reassembly Time Exceeded |
| 12 | 0 | Pointer indicates the error |
| | 1 | Missing a Required Option |
| | 2 | Bad Length |
| 40 | 0 | Bad SPI |
| | 1 | Authentication Failed |
| | 2 | Decompression Failed |
| | 3 | Decryption Failed |
| | 4 | Need Authentication |
| | 5 | Need Authorization |

### 6.6.3 icmp_id

With the icmp_id keyword you can match on specific ICMP id-values. Every ICMP-packet gets an id when it is being send. At the moment the receiver has received the packet, it will send a reply using the same id so the sender will recognize it and connects it with the correct ICMP-request.

Format of the icmp_id keyword:

```
icmp_id:<number>;
```

Example: This example looks for an ICMP ID of 0:

```
icmp_id:0;
```

Example of the icmp_id keyword in a rule:

alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"GPL SCAN Broadscan Smurf Scanner"; dsize:4; icmp_id:0; icmp_seq:0; itype:8; classtype:attempted-recon; sid:2100478; rev:4;)

### 6.6.4 icmp_seq

You can use the icmp_seq keyword to check for a ICMP sequence number. ICMP messages all have sequence numbers. This can be useful (together with the id) for checking which reply message belongs to which request message.

Format of the icmp_seq keyword:

```
icmp_seq:<number>;
```

Example: This example looks for an ICMP Sequence of 0:

```
icmp_seq:0;
```

Example of icmp_seq in a rule:

alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"GPL SCAN Broadscan Smurf Scanner"; dsize:4; icmp_id:0; icmp_seq:0; itype:8; classtype:attempted-recon; sid:2100478; rev:4;)

## 6.7 Payload Keywords

Payload keywords inspect the content of the payload of a packet or stream.

### 6.7.1 content

The content keyword is very important in signatures. Between the quotation marks you can write on what you would like the signature to match. The most simple format of content is:

```
content: ".....";
```

It is possible to use several contents in a signature.

Contents match on bytes. There are 256 different values of a byte (0-255). You can match on all characters; from a till z, upper case and lower case and also on all special signs. But not all of the bytes are printable characters. For these bytes heximal notations are used. Many programming languages use 0x00 as a notation, where 0x means it concerns a binary value, however the rule language uses `|00|` as a notation. This kind of notation can also be used for printable characters.

Example:

```
|61| is a
|61 61| is aa
|41| is A
```

(continues on next page)

(continued from previous page)

```
|21| is !
|0D| is carriage return
|0A| is line feed
```

There are characters you can not use in the content because they are already important in the signature. For matching on these characters you should use the heximal notation. These are:

```
"      |22|
;      |3B|
:      |3A|
|      |7C|
```

It is a convention to write the heximal notation in upper case characters.

To write for instance `http://` in the content of a signature, you should write it like this: `content: "http|3A|//";` If you use a heximal notation in a signature, make sure you always place it between pipes. Otherwise the notation will be taken literally as part of the content.

A few examples:

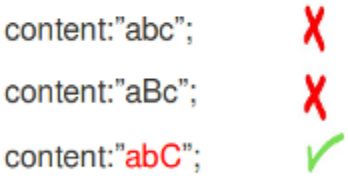
```
content:"a|0D|bc";
content:"|61 0D 62 63|";
content:"a|0D|b|63|";
```

It is possible to let a signature check the whole payload for a match with the content or to let it check specific parts of the payload. We come to that later. If you add nothing special to the signature, it will try to find a match in all the bytes of the payload.

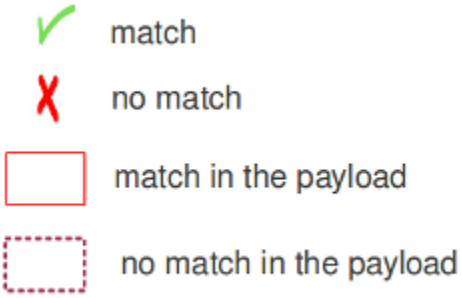
drop tcp $HOME_NET any -> $EXTERNAL_NET any (msg:”ET TROJAN Likely Bot Nick in IRC (USA +..)”; flow:established,to_server; flowbits:isset,is_proto_irc; content:”NICK “; pcre:”/NICK .*USA.*[0-9]{3,}/i”; reference:url,doc.emergingthreats.net/2008124; classtype:trojan-activity; sid:2008124; rev:2;)

By default the pattern-matching is case sensitive. The content has to be accurate, otherwise there will not be a match.




Legend:




It is possible to use the ! for exceptions in contents as well.

For example:

```
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"Outdated Firefox on
Windows"; content:"User-Agent|3A| Mozilla/5.0 |28|Windows|3B| ";
content:"Firefox/3."; distance:0; content:!"Firefox/3.6.13";
distance:-10; sid:9000000; rev:1;)
```

You see `content:!"Firefox/3.6.13";`. This means an alert will be generated if the used version of Firefox is not 3.6.13.

**Note:** The following characters must be escaped inside the content: `;` `\` `"`

### 6.7.2 nocase

If you do not want to make a distinction between uppercase and lowercase characters, you can use nocase. The keyword nocase is a content modifier.

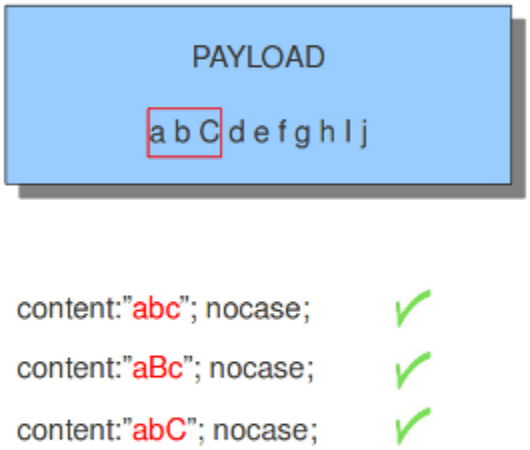
The format of this keyword is:

```
nocase;
```

You have to place it after the content you want to modify, like:

```
content: "abc"; nocase;
```

Example nocase:




It has no influence on other contents in the signature.

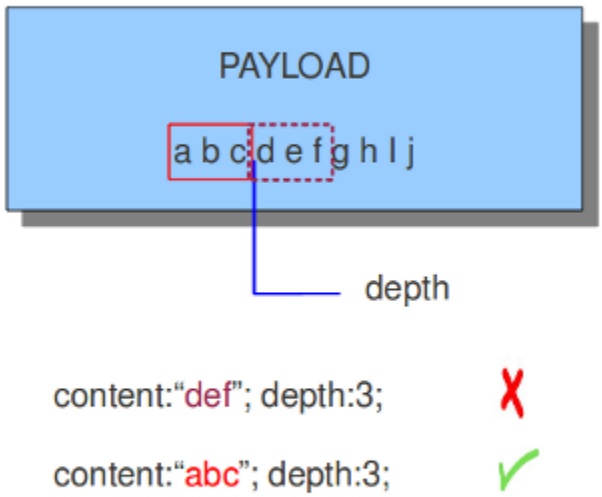
### 6.7.3 depth

The depth keyword is a absolute content modifier. It comes after the content. The depth content modifier comes with a mandatory numeric value, like:

```
depth:12;
```

The number after depth designates how many bytes from the beginning of the payload will be checked.

Example:




### 6.7.4 startswith

The `startswith` keyword is similar to `depth`. It takes no arguments and must follow a `content` keyword. It modifies the `content` to match exactly at the start of a buffer.

Example:

```
content:"GET|20|"; startswith;
```

`startswith` is a short hand notation for:

```
content:"GET|20|"; depth:4; offset:0;
```

`startswith` cannot be mixed with `depth`, `offset`, `within` or `distance` for the same pattern.

### 6.7.5 endswith

The `endswith` keyword is similar to `isdataat:!1,relative;`. It takes no arguments and must follow a `content` keyword. It modifies the `content` to match exactly at the end of a buffer.

Example:

```
content:".php"; endswith;
```

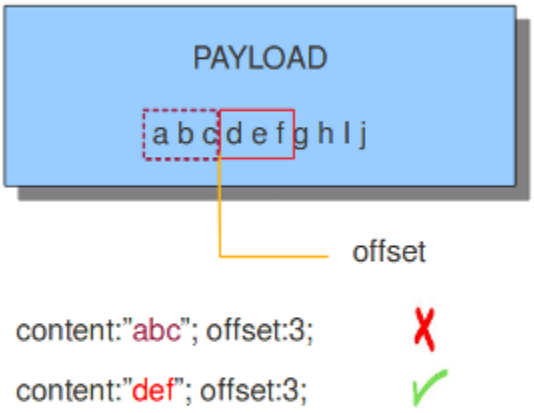
`endswith` is a short hand notation for:

```
content:".php"; isdatat:!1,relative;
```

`endswith` cannot be mixed with `offset`, `within` or `distance` for the same pattern.

### 6.7.6 offset

The offset keyword designates from which byte in the payload will be checked to find a match. For instance offset:3; checks the fourth byte and further.

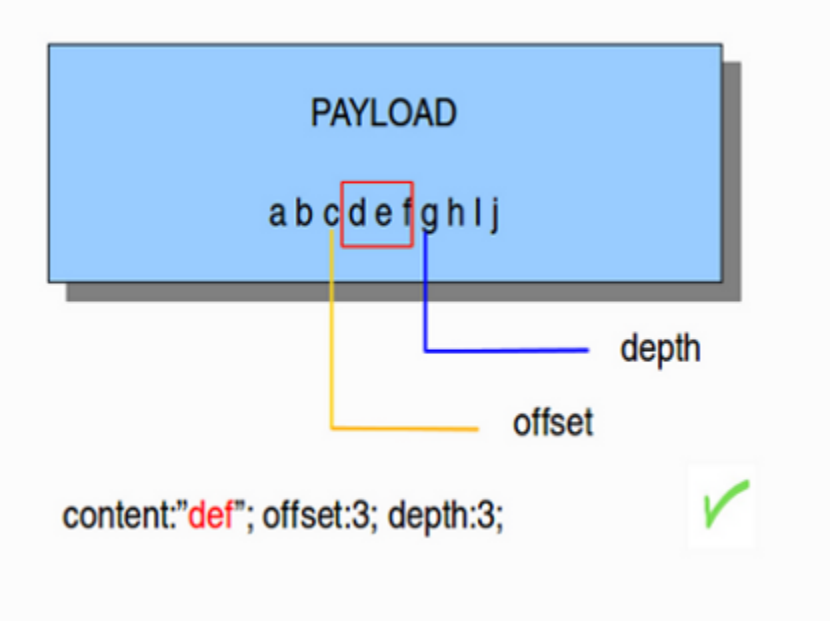



The keywords offset and depth can be combined and are often used together.

For example:

```
content:"def"; offset:3; depth:3;
```

If this was used in a signature, it would check the payload from the third byte till the sixth byte.



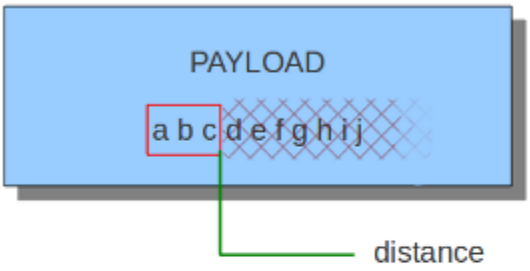

### 6.7.7 distance

The keyword distance is a relative content modifier. This means it indicates a relation between this content keyword and the content preceding it. Distance has its influence after the preceding match. The keyword distance comes with a mandatory numeric value. The value you give distance, determines the byte in the payload from which will be checked for a match relative to the previous match. Distance only determines where Suricata will start looking for a pattern. So, distance:5; means the pattern can be anywhere after the previous match + 5 bytes. For limiting how far after the last match Suricata needs to look, use 'within'.

Examples of distance:



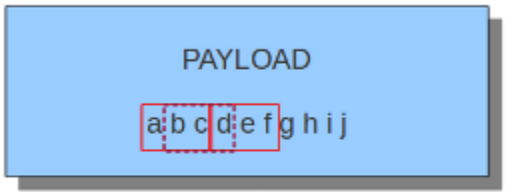

The distance (3), tells how the second (2) content relates to the first (1) content.




content:"abc"; content:"klm"; distance: 0;

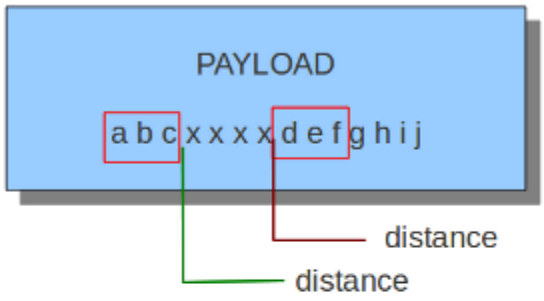


checked area using 'distance'




content:"abc"; content:"def"; distance:0;

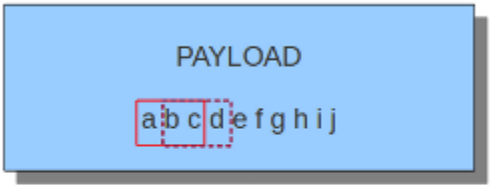
content:"abc"; content:"bcd"; distance:0;




content:"abc"; content:"def"; distance:0;

content:"abc"; content:"def"; distance:4;

Distance can also be a negative number. It can be used to check for matches with partly the same content (see example) or for a content even completely before it. This is not very often used though. It is possible to attain the same results with other keywords.




content:"abc"; content:"bcd"; distance:-2;

### 6.7.8 within

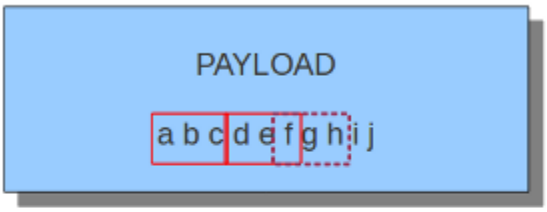
The keyword within is relative to the preceding match. The keyword within comes with a mandatory numeric value. Using within makes sure there will only be a match if the content matches with the payload within the set amount of bytes. Within can not be 0 (zero)

Example:




The keyword within (3), tells how the second (2) content relates to the first (1) content.

Example of matching with within:

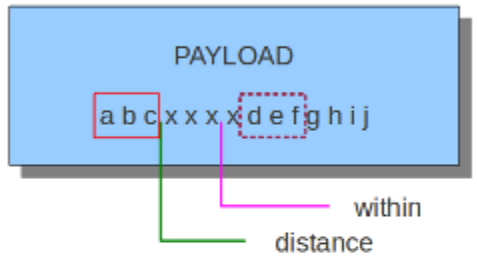



content:"abc"; content:"def"; within:3; ✓

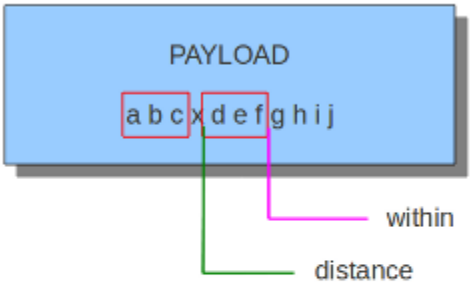
content:"abc"; content:"fgh"; within:3; ✗

The second content has to fall/come 'within 3 ' from the first content.

As mentioned before, distance and within can be very well combined in a signature. If you want Suricata to check a specific part of the payload for a match, use within.




content:"abc"; content:"def"; distance:0; within:3; ✗




content:"abc"; content:"def"; distance:1; within:4; ✓

### 6.7.9 isdataat

The purpose of the isdataat keyword is to look if there is still data at a specific part of the payload. The keyword starts with a number (the position) and then optional followed by 'relative' separated by a comma and the option rawbytes. You use the word 'relative' to know if there is still data at a specific part of the payload relative to the last match.

So you can use both examples:

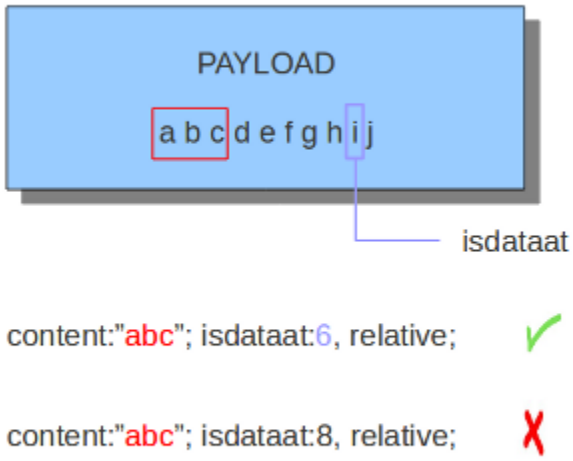
```
isdataat:512;

isdataat:50, relative;
```

The first example illustrates a signature which searches for byte 512 of the payload. The second example illustrates a

signature searching for byte 50 after the last match.

You can also use the negation (!) before isdataat.




### 6.7.10 bsize

With the bsize keyword, you can match on the length of the buffer. This adds precision to the content match, previously this could have been done with isdataat.

Format:

```
bsize:<number>;
```

Example of bsize in a rule:

alert dns any any -> any any (msg:"test bsize rule"; dns.query; content:"google.com"; bsize:10; sid:123; rev:1;)

### 6.7.11 dsize

With the dsize keyword, you can match on the size of the packet payload. You can use the keyword for example to look for abnormal sizes of payloads. This may be convenient in detecting buffer overflows.

Format:

```
dsize:<number>;
```

Example of dsize in a rule:

alert udp $EXTERNAL_NET any -> $HOME_NET 65535 (msg:"GPL DELETED EXPLOIT LANDesk Management Suite Alerting Service buffer overflow"; dsize:>268; reference: bugtraq,23483; reference: cve,2007-1674; classtype: attempted-admin; sid:100000928; rev:1;)

### 6.7.12 byte_test

The `byte_test` keyword extracts `<num of bytes>` and performs an operation selected with `<operator>` against the value in `<test value>` at a particular `<offset>`.

Format:

```
byte_test:<num of bytes>, [!]<operator>, <test value>, <offset> [,relative] \
[,<endian>][, string, <num type>][, dce][, bitmask <bitmask value>];
```

| | |
|---|---|
| <num of bytes> | The number of bytes selected from the packet to be converted |
| <operator> | • [!] Negation can prefix other operators<br>• < less than<br>• > greater than<br>• = equal<br>• <= less than or equal<br>• >= greater than or equal<br>• & bitwise AND<br>• ^ bitwise OR |
| <value> | Value to test the converted value against [hex or decimal accepted] |
| <offset> | Number of bytes into the payload |
| [relative] | Offset relative to last content match |
| [endian] | Type of number being read: - big (Most significant byte at lowest address) - little (Most significant byte at the highest address) |
| [string] <num> | • hex - Converted string represented in hex<br>• dec - Converted string represented in dedimal<br>• oct - Converted string represented in octal |
| [dce] | Allow the DCE module determine the byte order |
| [bitmask] | Applies the AND operator on the bytes converted |

Example:

```
alert tcp any any -> any any \
       (msg:"Byte_Test Example - Num = Value"; \
       content:"|00 01 00 02|"; byte_test:2,=,0x01;)

alert tcp any any -> any any \
       (msg:"Byte_Test Example - Num = Value relative to content"; \
       content:"|00 01 00 02|"; byte_test:2,=,0x03,relative;)

alert tcp any any -> any any \
       (msg:"Byte_Test Example - Num != Value"; content:"|00 01 00 02|"; \
       byte_test:2,!=,0x06;)

alert tcp any any -> any any \
       (msg:"Byte_Test Example - Detect Large Values"; content:"|00 01 00 02|"; \
       byte_test:2,>,1000,relavtive;)
```

(continues on next page)

(continued from previous page)

```
alert tcp any any -> any any \
       (msg:"Byte_Test Example - Lowest bit is set"; \
       content:"|00 01 00 02|"; byte_test:2,&,0x01,relative;)

alert tcp any any -> any any (msg:"Byte_Test Example - Compare to String"; \
       content:"foobar"; byte_test:4,=,1337,1,relative,string,dec;)
```

### 6.7.13 byte_jump

The `byte_jump` keyword allows for the ability to select a `<num of bytes>` from an `<offset>` and moves the detection pointer to that position. Content matches will then be based off the new position.

Format:

```
byte_jump:<num of bytes>, <offset> [, relative][, multiplier <mult_value>] \
      [, <endian>][, string, <num_type>][, align][, from_beginning][, from_end] \
      [, post_offset <value>][, dce][, bitmask <value>];
```

| | |
|---|---|
| <num of bytes> | The number of bytes selected from the packet to be converted |
| <offset> | Number of bytes into the payload |
| [relative] | Offset relative to last content match |
| [multiplier] <value> | Multiple the converted byte by the <value> |
| [endian] | • big (Most significant byte at lowest address)<br>• little (Most significant byte at the highest address) |
| [string] <num_type> | • hex Converted data is represented in hex<br>• dec Converted data is represented in decimal<br>• oct Converted data is represented as octal |
| [align] | Rounds the number up to the next 32bit boundary |
| [from_beginning] | Jumps forward from the beginning of the packet, instead of where the detection pointer is set |
| [from_end] | Jump will begin at the end of the payload, instead of where the detection point is set |
| [post_offset] <value> | After the jump operation has been performed, it will jump an additional number of bytes specified by <value> |
| [dce] | Allow the DCE module determine the byte order |
| [bitmask] <value> | The AND operator will be applied by <value> and the converted bytes, then jump operation is performed |

Example:

```
alert tcp any any -> any any \
      (msg:"Byte_Jump Example"; \
      content:"Alice"; byte_jump:2,0; content:"Bob";)

alert tcp any any -> any any \
      (msg:"Byte_Jump Multiple Jumps"; \
```

(continues on next page)

(continued from previous page)

```
      byte_jump:2,0; byte_jump:2,0,relative; content:"foobar"; distance:0; within:6;)

alert tcp any any -> any any \
      (msg:"Byte_Jump From the End -8 Bytes"; \
      byte_jump:0,0, from_end, post_offset -8; \
      content:"|6c 33 33 74|"; distance:0 within:4;)
```

### 6.7.14 byte_extract

The `byte_extract` keyword extracts `<num of bytes>` at a particular `<offset>` and stores it in `<var_name>`. The value in `<var_name>` can be used in any modifier that takes a number as an option and in the case of `byte_test` it can be used as a value.

Format:

```
byte_extract:<num of bytes>, <offset>, <var_name>, [, relative];
```

| Keyword | Modifier |
|---|---|
| content | offset,depth,distance,within |
| byte_test | offset,value |
| byte_jump | offset |
| isdataat | offset |

Example:

```
alert tcp any any -> any any \
       (msg:"Byte_Extract Example Using distance"; \
       content:"Alice"; byte_extract:2,0,size; content:"Bob"; distance:size; within:3;
↪ sid:1;)
alert tcp any any -> any any \
       (msg:"Byte_Extract Example Using within"; \
       flow:established,to_server; content:"|00 FF|"; \
       byte_extract:1,0,len,relative; content:"|5c 00|"; distance:2; within:len;␣
↪sid:2;)
alert tcp any any -> any any \
       (msg:"Byte_Extract Example Comparing Bytes"; \
       flow:established,to_server; content:"|00 FF|"; \
       byte_extract:2,0,cmp_ver,relative; content:"FooBar"; distance:0; byte_test:2,=,
↪cmp_ver,0; sid:3;)
```

### 6.7.15 rpc

The rpc keyword can be used to match in the SUNRPC CALL on the RPC procedure numbers and the RPC version.

You can modify the keyword by using a wild-card, defined with * With this wild-card you can match on all version and/or procedure numbers.

RPC (Remote Procedure Call) is an application that allows a computer program to execute a procedure on another computer (or address space). It is used for inter-process communication. See http://en.wikipedia.org/wiki/Inter-process_communication

Format:

```
rpc:<application number>, [<version number>|*], [<procedure number>|*]>;
```

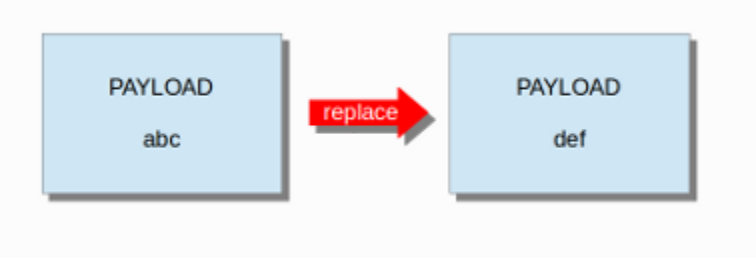
Example of the rpc keyword in a rule:

alert udp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request yppasswdd"; rpc:100009,*,*; reference:bugtraq,2763; classtype:rpc-portmap-decode; sid:1296; rev:4;)

### 6.7.16 replace

The replace content modifier can only be used in ips. It adjusts network traffic. It changes the content it follows ('abc') into another ('def'), see example:




The replace modifier has to contain as many characters as the content it replaces. It can only be used with individual packets. It will not work for *Normalized Buffers* like HTTP uri or a content match in the reassembled stream.

The checksums will be recalculated by Suricata and changed after the replace keyword is being used.

### 6.7.17 pcre (Perl Compatible Regular Expressions)

The keyword pcre matches specific on regular expressions. More information about regular expressions can be found here http://en.wikipedia.org/wiki/Regular_expression.

The complexity of pcre comes with a high price though: it has a negative influence on performance. So, to mitigate Suricata from having to check pcre often, pcre is mostly combined with 'content'. In that case, the content has to match first, before pcre will be checked.

Format of pcre:

```
pcre:"/<regex>/opts";
```

Example of pcre. In this example there will be a match if the payload contains six numbers following:

```
pcre:"/[0-9]{6}/";
```

Example of pcre in a signature:

drop tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"ET TROJAN Likely Bot Nick in IRC (USA +..)"; flow:established,to_server; flowbits:isset,is_proto_irc; content:"NICK "; pcre:"/NICK .*USA.*[0-9]{3,}/i"; reference:url,doc.emergingthreats.net/2008124; classtype:trojan-activity; sid:2008124; rev:2;)

There are a few qualities of pcre which can be modified:

- By default pcre is case-sensitive.

- The . (dot) is a part of regex. It matches on every byte except for newline characters.
- By default the payload will be inspected as one line.

These qualities can be modified with the following characters:

```
i    pcre is case insensitive
s    pcre does check newline characters
m    can make one line (of the payload) count as two lines
```

These options are perl compatible modifiers. To use these modifiers, you should add them to pcre, behind regex. Like this:

```
pcre: "/<regex>/i";
```

*Pcre compatible modifiers*

There are a few pcre compatible modifiers which can change the qualities of pcre as well. These are:

- `A`: A pattern has to match at the beginning of a buffer. (In pcre ^ is similar to A.)
- `E`: Ignores newline characters at the end of the buffer/payload.
- `G`: Inverts the greediness.

---

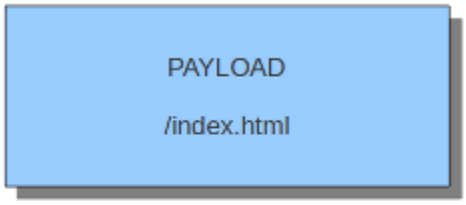
**Note:** The following characters must be escaped inside the content: `;` `\` `"`

---

### Suricata's modifiers

Suricata has its own specific pcre modifiers. These are:

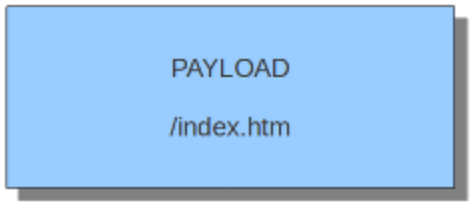
- `R`: Match relative to the last pattern match. It is similar to distance:0;
- `U`: Makes pcre match on the normalized uri. It matches on the uri_buffer just like uricontent and content combined with http_uri.U can be combined with /R. Note that R is relative to the previous match so both matches have to be in the HTTP-uri buffer. Read more about *HTTP URI Normalization*.




content:"/index."; http_uri; content:"htm"; http_uri; distance:0; ✓

content:"index."; http_uri; pcre:"/html?$/UR"; ✓

content:"index."; http_uri; pcre:"/^/index\.html?/$/U"; ✓




content:"/index."; http_uri; content:"htm"; http_uri; distance:0; ✓

content:"index."; http_uri; pcre:"/html?$/UR"; ✓

content:"index."; http_uri; pcre:"/^/index\.html?/$/U"; ✓




content:"/index."; http_uri; content:"htm"; http_uri; distance:0; ✓

content:"index."; http_uri; pcre:"/html?$/UR"; ✗

content:"index."; http_uri; pcre:"/^/index\.html?/$/U"; ✗




content:"/index."; http_uri; content:"htm"; http_uri; distance:0; ✗

content:"index."; http_uri; pcre:"/html?$/UR"; ✓

content:"index."; http_uri; pcre:"/^/index\.html?/$/U"; ✗

- `I`: Makes pcre match on the HTTP-raw-uri. It matches on the same buffer as http_raw_uri. I can be combined with /R. Note that R is relative to the previous match so both matches have to be in the HTTP-raw-uri buffer. Read more about *HTTP URI Normalization*.
- `P`: Makes pcre match on the HTTP- request-body. So, it matches on the same buffer as http_client_body. P can be combined with /R. Note that R is relative to the previous match so both matches have to be in the HTTP-request body.

- `Q`: Makes pcre match on the HTTP- response-body. So, it matches on the same buffer as http_server_body. Q can be combined with /R. Note that R is relative to the previous match so both matches have to be in the HTTP-response body.
- `H`: Makes pcre match on the HTTP-header. H can be combined with /R. Note that R is relative to the previous match so both matches have to be in the HTTP-header body.
- `D`: Makes pcre match on the unnormalized header. So, it matches on the same buffer as http_raw_header. D can be combined with /R. Note that R is relative to the previous match so both matches have to be in the HTTP-raw-header.
- `M`: Makes pcre match on the request-method. So, it matches on the same buffer as http_method. M can be combined with /R. Note that R is relative to the previous match so both matches have to be in the HTTP-method buffer.
- `C`: Makes pcre match on the HTTP-cookie. So, it matches on the same buffer as http_cookie. C can be combined with /R. Note that R is relative to the previous match so both matches have to be in the HTTP-cookie buffer.
- `S`: Makes pcre match on the HTTP-stat-code. So, it matches on the same buffer as http_stat_code. S can be combined with /R. Note that R is relative to the previous match so both matches have to be in the HTTP-stat-code buffer.
- `Y`: Makes pcre match on the HTTP-stat-msg. So, it matches on the same buffer as http_stat_msg. Y can be combined with /R. Note that R is relative to the previous match so both matches have to be in the HTTP-stat-msg buffer.
- `B`: You can encounter B in signatures but this is just for compatibility. So, Suricata does not use B but supports it so it does not cause errors.
- `O`: Overrides the configures pcre match limit.
- `V`: Makes pcre match on the HTTP-User-Agent. So, it matches on the same buffer as http_user_agent. V can be combined with /R. Note that R is relative to the previous match so both matches have to be in the HTTP-User-Agent buffer.
- `W`: Makes pcre match on the HTTP-Host. So, it matches on the same buffer as http_host. W can be combined with /R. Note that R is relative to the previous match so both matches have to be in the HTTP-Host buffer.

## 6.8 Transformations

Transformation keywords turn the data at a sticky buffer into something else.

Example:

```
alert http any any -> any any (file_data; strip_whitespace; \
    content:"window.navigate("; sid:1;)
```

This example will match on traffic even if there are one or more spaces between the `navigate` and `(`.

The transforms can be chained. They are processed in the order in which they appear in a rule. Each transforms output acts as input for the next one.

Example:

```
alert http any any -> any any (http_request_line; compress_whitespace; to_sha256; \
    content:"|54A9 7A8A B09C 1B81 3725 2214 51D3 F997 F015 9DD7 049E E5AD CED3 945A␣
↪FC79 7401|"; sid:1;)
```

**Note:** not all sticky buffers support transformations yet

### 6.8.1 dotprefix

Takes the buffer, and prepends a `.` character to help facilitate concise domain checks. For example, an input string of `hello.google.com` would be modified and become `.hello.google.com`. Additionally, adding the dot allows `google.com` to match against `content:".google.com"`

Example:

```
alert dns any any -> any any (dns.query; dotprefix; \
    content:".microsoft.com"; sid:1;)
```

This example will match on `windows.update.microsoft.com` and `maps.microsoft.com.au` but not `windows.update.fakemicrosoft.com`.

This rule can be used to match on the domain only; example:

```
alert dns any any -> any any (dns.query; dotprefix; \
    content:".microsoft.com"; endswith; sid:1;)
```

This example will match on `windows.update.microsoft.com` but not `windows.update.microsoft.com.au`.

Finally, this rule can be used to match on the TLD only; example:

```
alert dns any any -> any any (dns.query; dotprefix; \
    content:".co.uk"; endswith; sid:1;)
```

This example will match on `maps.google.co.uk` but not `maps.google.co.nl`.

### 6.8.2 strip_whitespace

Strips all whitespace as considered by the `isspace()` call in C.

Example:

```
alert http any any -> any any (file_data; strip_whitespace; \
    content:"window.navigate("; sid:1;)
```

### 6.8.3 compress_whitespace

Compresses all consecutive whitespace into a single space.

### 6.8.4 to_md5

Takes the buffer, calculates the MD5 hash and passes the raw hash value on.

Example:

```
alert http any any -> any any (http_request_line; to_md5; \
    content:"|54 A9 7A 8A B0 9C 1B 81 37 25 22 14 51 D3 F9 97|"; sid:1;)
```

**Note:** depends on libnss being compiled into Suricata

### 6.8.5 to_sha1

Takes the buffer, calculates the SHA-1 hash and passes the raw hash value on.

Example:

```
alert http any any -> any any (http_request_line; to_sha1; \
    content:"|54A9 7A8A B09C 1B81 3725 2214 51D3 F997 F015 9DD7|"; sid:1;)
```

**Note:** depends on libnss being compiled into Suricata

### 6.8.6 to_sha256

Takes the buffer, calculates the SHA-256 hash and passes the raw hash value on.

Example:

```
alert http any any -> any any (http_request_line; to_sha256; \
    content:"|54A9 7A8A B09C 1B81 3725 2214 51D3 F997 F015 9DD7 049E E5AD CED3 945A␣
↪FC79 7401|"; sid:1;)
```

**Note:** depends on libnss being compiled into Suricata

## 6.9 Prefiltering Keywords

### 6.9.1 fast_pattern

#### Suricata Fast Pattern Determination Explained

If the 'fast_pattern' keyword is explicitly set in a rule, Suricata will use that as the fast pattern match. The 'fast_pattern' keyword can only be set once per rule. If 'fast_pattern' is not set, Suricata automatically determines the content to use as the fast pattern match.

The following explains the logic Suricata uses to automatically determine the fast pattern match to use.

Be aware that if there are positive (i.e. non-negated) content matches, then negated content matches are ignored for fast pattern determination. Otherwise, negated content matches are considered.

The fast_pattern selection criteria are as follows:

1. Suricata first identifies all content matches that have the highest "priority" that are used in the signature. The priority is based off of the buffer being matched on and generally 'http_*' buffers have a higher priority (lower number is higher priority). See *Appendix B* for details on which buffers have what priority.
2. Within the content matches identified in step 1 (the highest priority content matches), the longest (in terms of character/byte length) content match is used as the fast pattern match.

3. If multiple content matches have the same highest priority and qualify for the longest length, the one with the highest character/byte diversity score ("Pattern Strength") is used as the fast pattern match. See *Appendix C* for details on the algorithm used to determine Pattern Strength.
4. If multiple content matches have the same highest priority, qualify for the longest length, and the same highest Pattern Strength, the buffer ("list_id") that was *registered last* is used as the fast pattern match. See *Appendix B* for the registration order of the different buffers/lists.
5. If multiple content matches have the same highest priority, qualify for the longest length, the same highest Pattern Strength, and have the same list_id (i.e. are looking in the same buffer), then the one that comes first (from left-to-right) in the rule is used as the fast pattern match.

It is worth noting that for content matches that have the same priority, length, and Pattern Strength, 'http_stat_msg', 'http_stat_code', and 'http_method' take precedence over regular 'content' matches.

## Appendices

### Appendix A - Buffers, list_id values, and Registration Order for Suricata 1.3.4

This should be pretty much the same for Suricata 1.1.x - 1.4.x.

| list_id | Content Modifier Keyword | Buffer Name | Registration Order |
|---|---|---|---|
| 1 | <none> (regular content match) | DETECT_SM_LIST_PMATCH | 1 (first) |
| 2 | http_uri | DETECT_SM_LIST_UMATCH | 2 |
| 6 | http_client_body | DETECT_SM_LIST_HCBDMATCH | 3 |
| 7 | http_server_body | DETECT_SM_LIST_HSBDMATCH | 4 |
| 8 | http_header | DETECT_SM_LIST_HHDMATCH | 5 |
| 9 | http_raw_header | DETECT_SM_LIST_HRHDMATCH | 6 |
| 10 | http_method | DETECT_SM_LIST_HMDMATCH | 7 |
| 11 | http_cookie | DETECT_SM_LIST_HCDMATCH | 8 |
| 12 | http_raw_uri | DETECT_SM_LIST_HRUDMATCH | 9 |
| 13 | http_stat_msg | DETECT_SM_LIST_HSMDMATCH | 10 |
| 14 | http_stat_code | DETECT_SM_LIST_HSCDMATCH | 11 |
| 15 | http_user_agent | DETECT_SM_LIST_HUADMATCH | 12 (last) |

Note: registration order doesn't matter when it comes to determining the fast pattern match for Suricata 1.3.4 but list_id value does.

### Appendix B - Buffers, list_id values, Priorities, and Registration Order for Suricata 2.0.7

This should be pretty much the same for Suricata 2.0.x.

| Priority (lower number is higher priority) | Registration Order | Content Modifier Keyword | Buffer Name | list_id |
|---|---|---|---|---|
| 3 | 11 | <none> (regular content match) | DETECT_SM_LIST_PMATCH | 1 |
| 3 | 12 | http_method | DETECT_SM_LIST_HMDMATCH | 12 |
| 3 | 13 | http_stat_code | DETECT_SM_LIST_HSCDMATCH | 9 |
| 3 | 14 | http_stat_msg | DETECT_SM_LIST_HSMDMATCH | 8 |
| 2 | 1 (first) | http_client_body | DETECT_SM_LIST_HCBDMATCH | 4 |
| 2 | 2 | http_server_body | DETECT_SM_LIST_HSBDMATCH | 5 |
| 2 | 3 | http_header | DETECT_SM_LIST_HHDMATCH | 6 |
| 2 | 4 | http_raw_header | DETECT_SM_LIST_HRHDMATCH | 7 |
| 2 | 5 | http_uri | DETECT_SM_LIST_UMATCH | 2 |
| 2 | 6 | http_raw_uri | DETECT_SM_LIST_HRUDMATCH | 3 |
| 2 | 7 | http_host | DETECT_SM_LIST_HHHDMATCH | 10 |
| 2 | 8 | http_raw_host | DETECT_SM_LIST_HRHHDMATCH | 11 |
| 2 | 9 | http_cookie | DETECT_SM_LIST_HCDMATCH | 13 |
| 2 | 10 | http_user_agent | DETECT_SM_LIST_HUADMATCH | 14 |
| 2 | 15 (last) | dns_query | DETECT_SM_LIST_DNSQUERY_MATCH | 20 |

Note: list_id value doesn't matter when it comes to determining the fast pattern match for Suricata 2.0.7 but registration order does.

### Appendix C - Pattern Strength Algorithm

From detect-engine-mpm.c. Basically the Pattern Strength "score" starts at zero and looks at each character/byte in the passed in byte array from left to right. If the character/byte has not been seen before in the array, it adds 3 to the score if it is an alpha character; else it adds 4 to the score if it is a printable character, 0x00, 0x01, or 0xFF; else it adds 6 to the score. If the character/byte has been seen before it adds 1 to the score. The final score is returned.

```
/** \brief Predict a strength value for patterns
 *
 *  Patterns with high character diversity score higher.
 *  Alpha chars score not so high
 *  Other printable + a few common codes a little higher
 *  Everything else highest.
 *  Longer patterns score better than short patters.
 *
 *  \param pat pattern
```

(continues on next page)

(continued from previous page)

```
 *  \param patlen length of the pattern
 *
 *  \retval s pattern score
 */
uint32_t PatternStrength(uint8_t *pat, uint16_t patlen) {
    uint8_t a[256];
    memset(&a, 0 ,sizeof(a));
    uint32_t s = 0;
    uint16_t u = 0;
    for (u = 0; u < patlen; u++) {
        if (a[pat[u]] == 0) {
            if (isalpha(pat[u]))
                s += 3;
            else if (isprint(pat[u]) || pat[u] == 0x00 || pat[u] == 0x01 || pat[u]
↪== 0xFF)
                s += 4;
            else
                s += 6;
            a[pat[u]] = 1;
        } else {
            s++;
        }
    }
    return s;
}
```

Only one content of a signature will be used in the Multi Pattern Matcher (MPM). If there are multiple contents, then Suricata uses the 'strongest' content. This means a combination of length, how varied a content is, and what buffer it is looking in. Generally, the longer and more varied the better. For full details on how Suricata determines the fast pattern match, see *Suricata Fast Pattern Determination Explained*.

Sometimes a signature writer concludes he wants Suricata to use another content than it does by default.

For instance:

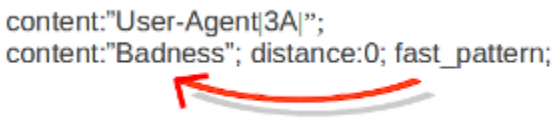
```
User-agent: Mozilla/5.0 Badness;

content:"User-Agent|3A|";
content:"Badness"; distance:0;
```

In this example you see the first content is longer and more varied than the second one, so you know Suricata will use this content for the MPM. Because 'User-Agent:' will be a match very often, and 'Badness' appears less often in network traffic, you can make Suricata use the second content by using 'fast_pattern'.

```
content:"User-Agent|3A|";
content:"Badness"; distance:0; fast_pattern;
```

The keyword fast_pattern modifies the content previous to it.




Fast-pattern can also be combined with all previous mentioned keywords, and all mentioned HTTP-modifiers.

#### fast_pattern:only

Sometimes a signature contains only one content. In that case it is not necessary Suricata will check it any further after a match has been found in MPM. If there is only one content, the whole signature matches. Suricata notices this automatically. In some signatures this is still indicated with 'fast_pattern:only;'. Although Suricata does not need fast_pattern:only, it does support it.

#### fast_pattern:'chop'

If you do not want the MPM to use the whole content, you can use fast_pattern 'chop'.

For example:

```
content: "aaaaaaaaabc"; fast_pattern:8,4;
```

This way, MPM uses only the last four characters.

### 6.9.2 prefilter

The prefilter engines for other non-MPM keywords can be enabled in specific rules by using the 'prefilter' keyword.

In the following rule the TTL test will be used in prefiltering instead of the single byte pattern:

```
alert ip any any -> any any (ttl:123; prefilter; content:"a"; sid:1;)
```

For more information on how to configure the prefilter engines, see *Prefilter Engines*

## 6.10 Flow Keywords

### 6.10.1 flowbits

Flowbits consists of two parts. The first part describes the action it is going to perform, the second part is the name of the flowbit.

There are multiple packets that belong to one flow. Suricata keeps those flows in memory. For more information see *Flow Settings*. Flowbits can make sure an alert will be generated when for example two different packets match. An alert will only be generated when both packets match. So, when the second packet matches, Suricata has to know if the first packet was a match too. Flowbits marks the flow if a packet matches so Suricata 'knows' it should generate an alert when the second packet matches as well.

Flowbits have different actions. These are:

**flowbits: set, name** Will set the condition/'name', if present, in the flow.

**flowbits: isset, name** Can be used in the rule to make sure it generates an alert when the rule matches and the condition is set in the flow.

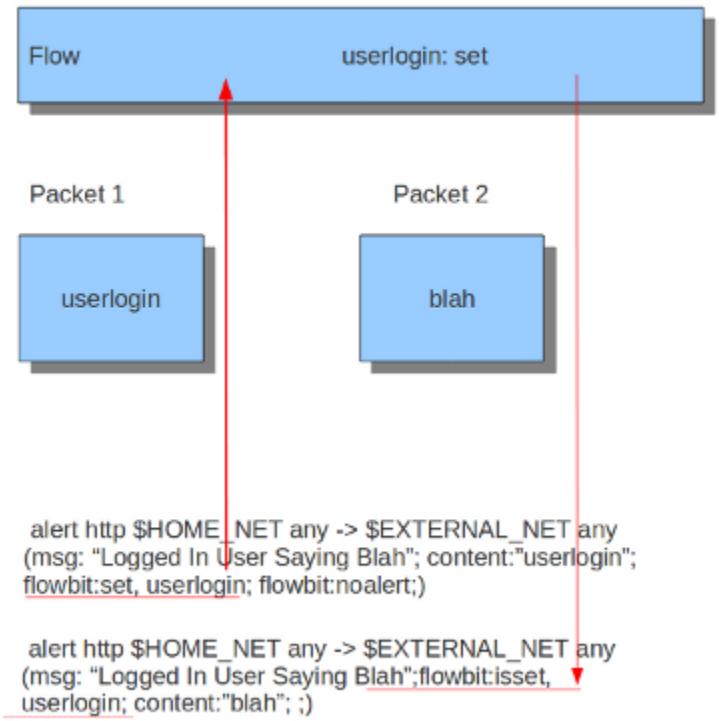
**flowbits: toggle, name** Reverses the present setting. So for example if a condition is set, it will be unset and vice-versa.

**flowbits: unset, name** Can be used to unset the condition in the flow.

**flowbits: isnotset, name** Can be used in the rule to make sure it generates an alert when it matches and the condition is not set in the flow.

**flowbits: noalert** No alert will be generated by this rule.

Example:




When you take a look at the first rule you will notice it would generate an alert if it would match, if it were not for the 'flowbits: noalert' at the end of that rule. The purpose of this rule is to check for a match on 'userlogin' and mark that in the flow. So, there is no need for generating an alert. The second rule has no effect without the first rule. If the first rule matches, the flowbits sets that specific condition to be present in the flow. Now with the second rule there can be checked whether or not the previous packet fulfills the first condition. If at that point the second rule matches, an alert will be generated.

It is possible to use flowbits several times in a rule and combine the different functions.

### 6.10.2 flow

The flow keyword can be used to match on direction of the flow, so to/from client or to/from server. It can also match if the flow is established or not. The flow keyword can also be use to say the signature has to match on stream only (only_stream) or on packet only (no_stream).

So with the flow keyword you can match on:

**to_client** Match on packets from server to client.

**to_server** Match on packets from client to server.

**from_client** Match on packets from client to server (same as to_server).

**from_server** Match on packets from server to client (same as to_client).

**established** Match on established connections.

**not_established** Match on packets that are not part of an established connection.

**stateless** Match on packets that are and are not part of an established connection.

**only_stream** Match on packets that have been reassembled by the stream engine.

**no_stream** Match on packets that have not been reassembled by the stream engine. Will not match packets that have been reassembled.

**only_frag** Match packets that have been reassembled from fragments.

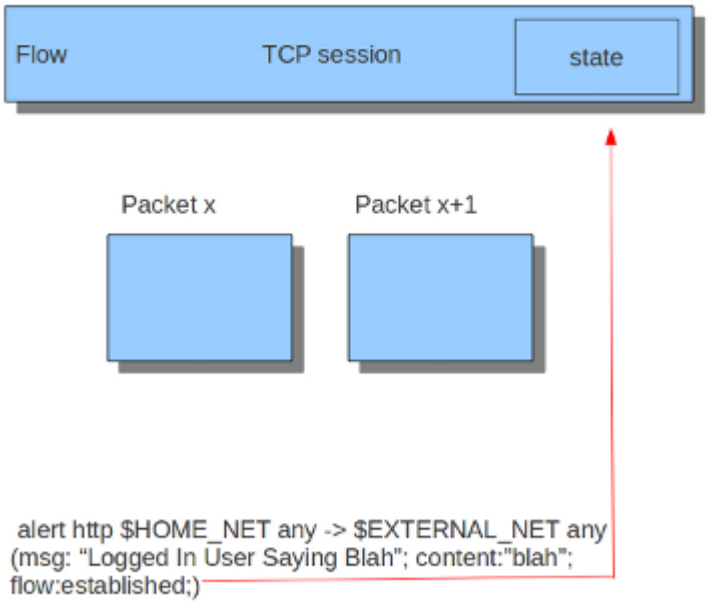
**no_frag** Match packets that have not been reassembled from fragments.

Multiple flow options can be combined, for example:

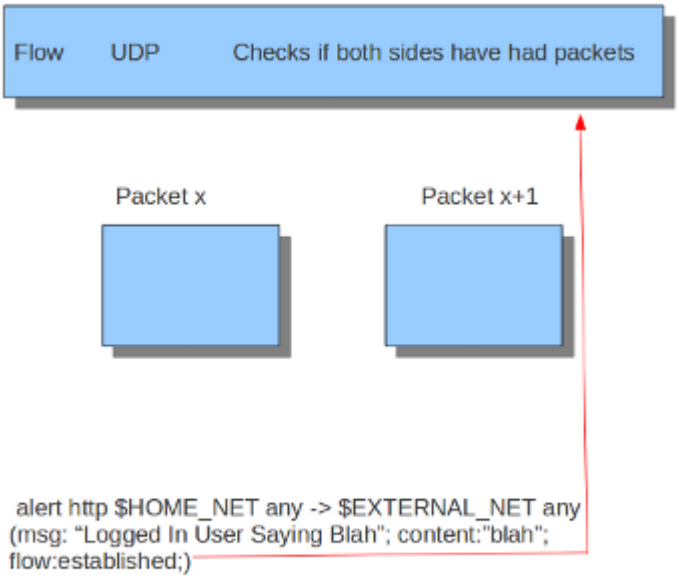
```
flow:to_client, established
flow:to_server, established, only_stream
flow:to_server, not_established, no_frag
```

The determination of *established* depends on the protocol:

- For TCP a connection will be established after a three way handshake.




- For other protocols (for example UDP), the connection will be considered established after seeing traffic from both sides of the connection.




### 6.10.3 flowint

Flowint allows storage and mathematical operations using variables. It operates much like flowbits but with the addition of mathematical capabilities and the fact that an integer can be stored and manipulated, not just a flag set. We can use this for a number of very useful things, such as counting occurrences, adding or subtracting occurrences, or doing thresholding within a stream in relation to multiple factors. This will be expanded to a global context very soon, so users can perform these operations between streams.

The syntax is as follows:

```
flowint: name, modifier[, value];
```

Define a var (not required), or check that one is set or not set.

```
flowint: name, < +,-,=,>,<,>=,<=,==, != >, value;
flowint: name, (isset|isnotset);
```

Compare or alter a var. Add, subtract, compare greater than or less than, greater than or equal to, and less than or equal to are available. The item to compare with can be an integer or another variable.

---

For example, if you want to count how many times a username is seen in a particular stream and alert if it is over 5.

```
alert tcp any any -> any any (msg:"Counting Usernames"; content:"jonkman"; \
      flowint: usernamecount, +, 1; noalert;)
```

This will count each occurrence and increment the var usernamecount and not generate an alert for each.

Now say we want to generate an alert if there are more than five hits in the stream.

```
alert tcp any any -> any any (msg:"More than Five Usernames!"; content:"jonkman"; \
      flowint: usernamecount, +, 1; flowint:usernamecount, >, 5;)
```

So we'll get an alert ONLY if usernamecount is over five.

So now let's say we want to get an alert as above but NOT if there have been more occurrences of that username logging out. Assuming this particular protocol indicates a log out with "jonkman logout", let's try:

```
alert tcp any any -> any any (msg:"Username Logged out"; content:"logout jonkman"; \
      flowint: usernamecount, -, 1; flowint:usernamecount, >, 5;)
```

So now we'll get an alert ONLY if there are more than five active logins for this particular username.

This is a rather simplistic example, but I believe it shows the power of what such a simple function can do for rule writing. I see a lot of applications in things like login tracking, IRC state machines, malware tracking, and brute force login detection.

Let's say we're tracking a protocol that normally allows five login fails per connection, but we have vulnerability where an attacker can continue to login after that five attempts and we need to know about it.

```
alert tcp any any -> any any (msg:"Start a login count"; content:"login failed"; \
      flowint:loginfail, notset; flowint:loginfail, =, 1; noalert;)
```

So we detect the initial fail if the variable is not yet set and set it to 1 if so. Our first hit.

```
alert tcp any any -> any any (msg:"Counting Logins"; content:"login failed"; \
      flowint:loginfail, isset; flowint:loginfail, +, 1; noalert;)
```

We are now incrementing the counter if it's set.

```
alert tcp any any -> any any (msg:"More than Five login fails in a Stream"; \
      content:"login failed"; flowint:loginfail, isset; flowint:loginfail, >, 5;)
```

Now we'll generate an alert if we cross five login fails in the same stream.

But let's also say we also need alert if there are two successful logins and a failed login after that.

```
alert tcp any any -> any any (msg:"Counting Good Logins";              \
      content:"login successful"; flowint:loginsuccess, +, 1; noalert;)
```

Here we're counting good logins, so now we'll count good logins relevant to fails:

```
alert tcp any any -> any any (msg:"Login fail after two successes";    \
      content:"login failed"; flowint:loginsuccess, isset;             \
      flowint:loginsuccess, =, 2;)
```

Here are some other general examples:

```
alert tcp any any -> any any (msg:"Setting a flowint counter"; content:"GET"; \
      flowint:myvar, notset; flowint:maxvar,notset;                           \
      flowint:myvar,=,1; flowint: maxvar,=,6;)
```

```
alert tcp any any -> any any (msg:"Adding to flowint counter";                \
      content:"Unauthorized"; flowint:myvar,isset; flowint: myvar,+,2;)
```

```
alert tcp any any -> any any (msg:"when flowint counter is 3 create new counter"; \
      content:"Unauthorized"; flowint:myvar, isset; flowint:myvar,==,3; \
      flowint:cntpackets,notset; flowint:cntpackets, =, 0;)
```

```
alert tcp any any -> any any (msg:"count the rest without generating alerts"; \
      flowint:cntpackets,isset; flowint:cntpackets, +, 1; noalert;)
```

```
alert tcp any any -> any any (msg:"fire this when it reach 6";                \
      flowint: cntpackets, isset;                                             \
      flowint: maxvar,isset; flowint: cntpackets, ==, maxvar;)
```

### 6.10.4 stream_size

The stream size option matches on traffic according to the registered amount of bytes by the sequence numbers. There are several modifiers to this keyword:

```
>       greater than
<       less than
=       equal
!=      not equal
>=     greater than or equal
<=     less than or equal
```

Format

```
stream_size:<server|client|both|either>, <modifier>, <number>;
```

Example of the stream-size keyword in a rule:

```
alert tcp any any -> any any (stream_size:both, >, 5000; sid:1;)
```

## 6.11 Bypass Keyword

Suricata has a `bypass` keyword that can be used in signatures to exclude traffic from further evaluation.

The `bypass` keyword is useful in cases where there is a large flow expected (e.g. Netflix, Spotify, Youtube).

The `bypass` keyword is considered a post-match keyword.

### 6.11.1 bypass

Bypass a flow on matching http traffic.

Example:

```
alert http any any -> any any (content:"suricata-ids.org"; \
    http_host; bypass; sid:10001; rev:1;)
```

## 6.12 HTTP Keywords

There are additional content modifiers that can provide protocol-specific capabilities at the application layer. More information can be found at *Payload Keywords* These keywords make sure the signature checks only specific parts of the network traffic. For instance, to check specifically on the request URI, cookies, or the HTTP request or response body, etc.

All HTTP keywords are modifiers. Note the difference between content modifiers and sticky buffers. See *Modifier Keywords* for more information. As a refresher:

- **'sticky buffers'** are placed first and all keywords following it apply to that buffer, for instance:

```
alert http any any -> any any (http.response_line; content:"403 Forbidden"; sid:1;
↪)
```

- **'content modifiers'** look back in the rule, e.g.:

```
alert http any any -> any any (content:"index.php"; http_uri; sid:1;)
```

The following **request** keywords are available:

| Keyword | Sticky or Modifier | Direction |
|---|---|---|
| http.uri | Sticky Buffer | Request |
| http.uri.raw | Sticky Buffer | Request |
| http.method | Sticky Buffer | Request |
| http.request_line | Sticky Buffer | Request |
| http.request_body | Sticky Buffer | Request |
| http.header | Sticky Buffer | Both |
| http.header.raw | Sticky Buffer | Both |
| http.cookie | Sticky Buffer | Both |
| http.user_agent | Sticky Buffer | Request |
| http.host | Sticky Buffer | Request |
| http.host.raw | Sticky Buffer | Request |
| http.accept | Sticky Buffer | Request |
| http.accept_lang | Sticky Buffer | Request |
| http.accept_enc | Sticky Buffer | Request |
| http.referer | Sticky Buffer | Request |
| http.connection | Sticky Buffer | Request |
| http.content_type | Sticky Buffer | Both |
| http.content_len | Sticky Buffer | Both |
| http.start | Sticky Buffer | Both |
| http.protocol | Sticky Buffer | Both |
| http.header_names | Sticky Buffer | Both |

The following **response** keywords are available:

| Keyword | Sticky or Modifier | Direction |
|---|---|---|
| http.stat_msg | Sticky Buffer | Response |
| http.stat_code | Sticky Buffer | Response |
| http.response_line | Sticky Buffer | Response |
| http.header | Sticky Buffer | Both |
| http.header.raw | Sticky Buffer | Both |
| http.cookie | Sticky Buffer | Both |
| http.server_body | Sticky Buffer | Response |
| http.server | Sticky Buffer | Response |
| http.location | Sticky Buffer | Response |
| file_data | Sticky Buffer | Response |
| http.content_type | Sticky Buffer | Both |
| http.content_len | Sticky Buffer | Both |
| http.start | Sticky Buffer | Both |
| http.protocol | Sticky Buffer | Both |
| http.header_names | Sticky Buffer | Both |

### 6.12.1 HTTP Primer

It is important to understand the structure of HTTP requests and responses. A simple example of a HTTP request and response follows:

**HTTP request**

```
GET /index.html HTTP/1.0\r\n
```

GET is a request **method**. Examples of methods are: GET, POST, PUT, HEAD, etc. The URI path is `/index.html` and the HTTP version is `HTTP/1.0`. Several HTTP versions have been used over the years; of the versions 0.9, 1.0 and 1.1, 1.0 and 1.1 are the most commonly used today.

**HTTP response**

```
HTTP/1.0 200 OK\r\n
<html>
<title> some page </title>
</HTML>
```

In this example, HTTP/1.0 is the HTTP version, 200 the response status code and OK the response status message.

Another more detailed example:

Request:

| | |
|---|---|
| GET / HTTP/1.1 | HTTP-method, keyword: http_method<br>HTTP-uri, keywords: http_uri or http_raw_uri<br>HTTP-version |
| Host: www.google.com<br>Connection: keep-alive<br>User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US) AppleWebKit/534.16 (KHTML, like Gecko) Ubuntu/10.10 Chromium/10.0.618.0 Chrome/10.0.618.0 Safari/534.16<br>Accept-Encoding: gzip,deflate,sdch<br>Accept-Language: en-US,en;q=0.8<br>Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3 | HTTP-header, keywords: http_header, http_raw_header<br><br>User-Agent (part of HTTP-header), keyword: http_user_agent |
| Cookie: PREF=ID=efe36c63a3bfa6a4:U=aa0cf39996084d7e:TM=1252314621:LM=1292956821:GM=1:S=dYtecyNBioerA47b | HTTP-cookie, keyword: http_cookie |

Response:

| | |
|---|---|
| HTTP/1.1 302 Found | HTTP-version<br>HTTP-response code, keyword: http_stat_code<br>HTTP-response message, keyword: http_stat_msg |
| Location: http://www.google.nl/<br>Cache-Control: private<br>Content-Type: text/html; charset=UTF-8<br>Set-Cookie: PREF=ID=efe36c63a3bfa6a4:FF=0:TM=1252314621:LM=1293104406:GM=1:S=xeKyIaBhZkPrZEkN; expires=Sat, 22-Dec-2012 11:40:06 GMT; path=/; domain=.google.com<br>Date: Thu, 23 Dec 2010 11:40:06 GMT<br>Server: gws<br>Content-Length: 218<br>X-XSS-Protection: 1; mode=block | HTTP-header, keywords: http_header, http_raw_header |
| <HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8"><br><TITLE>302 Moved</TITLE></HEAD><BODY><br><H1>302 Moved</H1><br>The document has moved<br><A HREF="http://www.google.nl/">here</A>.<br></BODY></HTML> | HTTP-response body, keywords: file_data, http_server_body |

Request:

| | |
|---|---|
| POST / HTTP/1.0 | HTTP-method, keyword: http_method<br>HTTP-uri, keywords: http_uri or http_raw_uri<br>HTTP-version |
| Accept: */*<br>Accept-Language: en-US<br>x-flash-version: 9,0,115,0<br>Content-Type: application/x-www-form-urlencoded<br>Content-Length: 31<br>Accept-Encoding: bbbbbbbbblate<br>User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)<br>Host: nowhereasdfasdf.com<br>Connection: Keep-Alive<br>Cache-Control: no-cache | HTTP-header, keywords: http_header, http_raw_header |
| type=playerStart&position=tidal | HTTP-client body, keyword: http_client_body |

Although cookies are sent in an HTTP header, you can not match on them with the `http.header` keyword. Cookies are matched with their own keyword, namely `http.cookie`.

Each part of the table belongs to a so-called *buffer*. The HTTP method belongs to the method buffer, HTTP headers to the header buffer etc. A buffer is a specific portion of the request or response that Suricata extracts in memory for inspection.

All previous described keywords can be used in combination with a buffer in a signature. The keywords `distance` and `within` are relative modifiers, so they may only be used within the same buffer. You can not relate content matches against different buffers with relative modifiers.

### 6.12.2 http.method

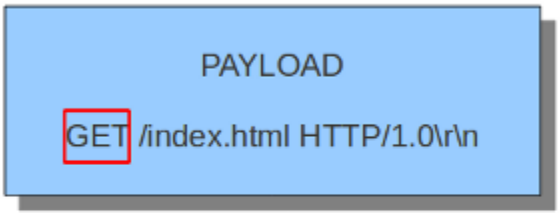
With the `http.method` content modifier, it is possible to match specifically and only on the HTTP method buffer. The keyword can be used in combination with all previously mentioned content modifiers such as: `depth`, `distance`, `offset`, `nocase` and `within`.

Examples of methods are: **GET**, **POST**, **PUT**, **HEAD**, **DELETE**, **TRACE**, **OPTIONS**, **CONNECT** and **PATCH**.

Example of a method in a HTTP request:

```
GET / HTTP/1.1
Host: www.google.com
Connection: keep-alive
Accept:
application/xml,application/xhtml+xml,text/html;q=0.9,text/
plain;q=0.8,image/png,*/*;q=0.5
```

Example of the purpose of method:




content:"GET"; ✓

content:"GET"; http_method ✓

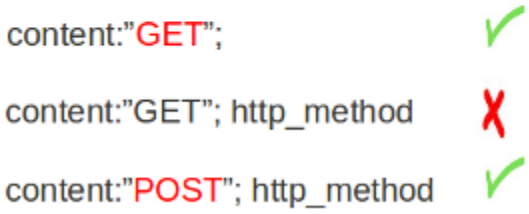
✓ match

✗ no match

match in the payload

no match in the payload




### 6.12.3 http.uri and http.uri.raw

With the `http.uri` and the `http.uri.raw` content modifiers, it is possible to match specifically and only on the request URI buffer. The keyword can be used in combination with all previously mentioned content modifiers like `depth`, `distance`, `offset`, `nocase` and `within`.

The uri has two appearances in Suricata: the uri.raw and the normalized uri. The space for example can be indicated with the heximal notation %20. To convert this notation in a space, means normalizing it. It is possible though to match specific on the characters %20 in a uri. This means matching on the uri.raw. The uri.raw and the normalized uri are separate buffers. So, the uri.raw inspects the uri.raw buffer and can not inspect the normalized buffer.

Example of the URI in a HTTP request:

```
GET /index.html HTTP/1.0\r\n
```

Example of the purpose of `http.uri`:




content: "/index.html"; http_uri; ✓

content: "GET"; http_uri; ✗

content: "/index"; http_uri; content: ".html"; http_uri; within:5; ✓

content: "/index"; http_uri; depth:6; ✓

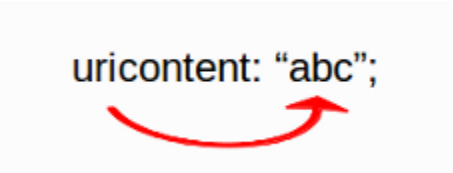
### 6.12.4 uricontent

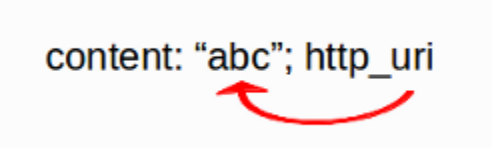
The `uricontent` keyword has the exact same effect as the `http.uri` content modifier. `uricontent` is a deprecated (although still supported) way to match specifically and only on the request URI buffer.

Example of `uricontent`:

alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"ET TROJAN Possible Vundo Trojan Variant reporting to Controller"; flow:established,to_server; content:"POST "; depth:5; uricontent:"/frame.html?"; urilen: > 80; classtype:trojan-activity; reference:url,doc.emergingthreats.net/2009173; reference:url,www.emergingthreats.net/cgi-bin/cvsweb.cgi/sigs/VIRUS/TROJAN_Vundo; sid:2009173; rev:2;)

The difference between `http.uri` and `uricontent` is the syntax:




When authoring new rules, it is recommended that the `http.uri` content sticky buffer be used rather than the deprecated `uricontent` keyword.

### 6.12.5 urilen

The `urilen` keyword is used to match on the length of the request URI. It is possible to use the < and > operators, which indicate respectively *smaller than* and *larger than*.

The format of `urilen` is:

```
urilen:3;
```

Other possibilities are:

```
urilen:1;
urilen:>1;
urilen:<10;
urilen:10<>20;          (bigger than 10, smaller than 20)
```

Example:







Example of `urilen` in a signature:

alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"ET TROJAN Possible Vundo Trojan Variant reporting to Controller"; flow:established,to_server; content:"POST "; depth:5; uricontent:"/frame.html?"; urilen: > 80; classtype:trojan-activity; reference:url,doc.emergingthreats.net/2009173; reference:url,www.emergingthreats.net/cgi-bin/cvsweb.cgi/sigs/VIRUS/TROJAN_Vundo; sid:2009173; rev:2;)

You can also append `norm` or `raw` to define what sort of buffer you want to use (normalized or raw buffer).

### 6.12.6 http.protocol

The `http.protocol` inspects the protocol field from the HTTP request or response line. If the request line is 'GET / HTTP/1.0rn', then this buffer will contain 'HTTP/1.0'.

Example:

```
alert http any any -> any any (flow:to_server; http.protocol; content:"HTTP/1.0";␣
↪sid:1;)
```

`http.protocol` replaces the previous keyword name: `http_protocol. You may continue +to use the previous name, but it's recommended that rules be converted to use +the new name.

Example:

```
alert http any any -> any any (flow:to_server; http.protocol; content:"HTTP/1.0";␣
↪sid:1;)
```

### 6.12.7 http.request_line

The `http.request_line` forces the whole HTTP request line to be inspected.

Example:

```
alert http any any -> any any (http.request_line; content:"GET / HTTP/1.0"; sid:1;)
```

### 6.12.8 http.header and http.header.raw

With the `http.header` content modifier, it is possible to match specifically and only on the HTTP header buffer. This contains all of the extracted headers in a single buffer, except for those indicated in the documentation that are not able to match by this buffer and have their own content modifier (e.g. `http.cookie`). The modifier can be used in combination with all previously mentioned content modifiers, like `depth`, `distance`, `offset`, `nocase` and `within`.

> **Note**: the header buffer is *normalized*. Any trailing whitespace and tab characters are removed. See: https://lists.openinfosecfoundation.org/pipermail/oisf-users/2011-October/000935.html. To avoid that, use the `http.header.raw` keyword.

Example of a header in a HTTP request:

```
GET / HTTP/1.1
Host: www.google.com
Connection: keep-alive
Accept:
application/xml,application/xhtml+xml,text/html;q=0.9,
text/plain;q=0.8,image/png,*/*;q=0.5
```

Example of the purpose of `http.header`:




### 6.12.9 http.cookie

With the `http.cookie` content modifier, it is possible to match specifically and only on the cookie buffer. The keyword can be used in combination with all previously mentioned content modifiers like `depth`, `distance`, `offset`, `nocase` and `within`.

Note that cookies are passed in HTTP headers, but are extracted to a dedicated buffer and matched using their own specific content modifier.

Example of a cookie in a HTTP request:

```
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US)
AppleWebKit/534.16
(KHTML, like Gecko) Ubuntu/10.10 Chromium/10.0.618.0
Chrome/10.0.618.0
Safari/534.16
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
Cookie:
PREF=ID=efe36c63a3bfa6a4:U=aa0cf39996084d7e:TM
=1252314621:LM=1292956821:GM=1:S=dYtecyNBioer
A47b
```

Example of the purpose of `http.cookie`:




### 6.12.10 http.user_agent

The `http.user_agent` content modifier is part of the HTTP request header. It makes it possible to match specifically on the value of the User-Agent header. It is normalized in the sense that it does not include the _"User-Agent: "_ header name and separator, nor does it contain the trailing carriage return and line feed (CRLF). The keyword can be used in combination with all previously mentioned content modifiers like `depth`, `distance`, `offset`, `nocase` and `within`. Note that the `pcre` keyword can also inspect this buffer when using the `/V` modifier.

Normalization: leading spaces **are not** part of this buffer. So "User-Agent: rn" will result in an empty `http.user_agent` buffer.

Example of the User-Agent header in a HTTP request:

```
GET / HTTP/1.1
Host: www.google.com
Connection: keep-alive
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US)
AppleWebKit/534.16
(KHTML, like Gecko) Ubuntu/10.10
Chromium/10.0.618.0 Chrome/10.0.618.0
Safari/534.16
```

Example of the purpose of `http.user_agent`:




### Notes

- The `http.user_agent` buffer will NOT include the header name, colon, or leading whitespace. i.e. it will not include "User-Agent: ".
- The `http.user_agent` buffer does not include a CRLF (0x0D 0x0A) at the end. If you want to match the end of the buffer, use a relative `isdataat` or a PCRE (although PCRE will be worse on performance).
- If a request contains multiple "User-Agent" headers, the values will be concatenated in the `http.user_agent` buffer, in the order seen from top to bottom, with a comma and space (", ") between each of them.

  Example request:

```
GET /test.html HTTP/1.1
User-Agent: SuriTester/0.8
User-Agent: GGGG
```

  `http.user_agent` buffer contents:

```
SuriTester/0.8, GGGG
```

- Corresponding PCRE modifier: `V`
- Using the `http.user_agent` buffer is more efficient when it comes to performance than using the `http.header` buffer (~10% better).
- https://blog.inliniac.net/2012/07/09/suricata-http_user_agent-vs-http_header/

### 6.12.11 http.accept

Sticky buffer to match on the HTTP Accept header. Only contains the header value. The \r\n after the header are not part of the buffer.

Example:

```
alert http any any -> any any (http.accept; content:"image/gif"; sid:1;)
```

### 6.12.12 http.accept_enc

Sticky buffer to match on the HTTP Accept-Encoding header. Only contains the header value. The \r\n after the header are not part of the buffer.

Example:

```
alert http any any -> any any (http.accept_enc; content:"gzip"; sid:1;)
```

### 6.12.13 http.accept_lang

Sticky buffer to match on the HTTP Accept-Language header. Only contains the header value. The \r\n after the header are not part of the buffer.

Example:

```
alert http any any -> any any (http.accept_lang; content:"en-us"; sid:1;)
```

### 6.12.14 http.connection

Sticky buffer to match on the HTTP Connection header. Only contains the header value. The \r\n after the header are not part of the buffer.

Example:

```
alert http any any -> any any (http.connection; content:"keep-alive"; sid:1;)
```

### 6.12.15 http.content_type

Sticky buffer to match on the HTTP Content-Type headers. Only contains the header value. The \r\n after the header are not part of the buffer.

Use flow:to_server or flow:to_client to force inspection of request or response.

Examples:

```
alert http any any -> any any (flow:to_server; \
        http.content_type; content:"x-www-form-urlencoded"; sid:1;)

alert http any any -> any any (flow:to_client; \
        http.content_type; content:"text/javascript"; sid:2;)
```

### 6.12.16 http.content_len

Sticky buffer to match on the HTTP Content-Length headers. Only contains the header value. The \r\n after the header are not part of the buffer.

Use flow:to_server or flow:to_client to force inspection of request or response.

Examples:

```
alert http any any -> any any (flow:to_server; \
        http.content_len; content:"666"; sid:1;)

alert http any any -> any any (flow:to_client; \
        http.content_len; content:"555"; sid:2;)
```

To do a numeric inspection of the content length, `byte_test` can be used.

Example, match if C-L is equal to or bigger than 8079:

```
alert http any any -> any any (flow:to_client; \
        http.content_len; byte_test:0,>=,8079,0,string,dec; sid:3;)
```

### 6.12.17 http.referer

Sticky buffer to match on the HTTP Referer header. Only contains the header value. The \r\n after the header are not part of the buffer.

Example:

```
alert http any any -> any any (http.referer; content:".php"; sid:1;)
```

### 6.12.18 http.start

Inspect the start of a HTTP request or response. This will contain the request/response line plus the request/response headers. Use flow:to_server or flow:to_client to force inspection of request or response.

Example:

```
alert http any any -> any any (http.start; content:"HTTP/1.1|0d 0a|User-Agent"; sid:1;
↪)
```

The buffer contains the normalized headers and is terminated by an extra \r\n to indicate the end of the headers.

### 6.12.19 http.header_names

Inspect a buffer only containing the names of the HTTP headers. Useful for making sure a header is not present or testing for a certain order of headers.

Buffer starts with a \r\n and ends with an extra \r\n.

Example buffer:

```
\\r\\nHost\\r\\n\\r\\n
```

Example rule:

```
alert http any any -> any any (http.header_names; content:"|0d 0a|Host|0d 0a|"; sid:1;
↪)
```

Example to make sure *only* Host is present:

```
alert http any any -> any any (http.header_names; \
        content:"|0d 0a|Host|0d 0a 0d 0a|"; sid:1;)
```

Example to make sure *User-Agent* is directly after *Host*:

```
alert http any any -> any any (http.header_names; \
        content:"|0d 0a|Host|0d 0a|User-Agent|0d 0a|"; sid:1;)
```

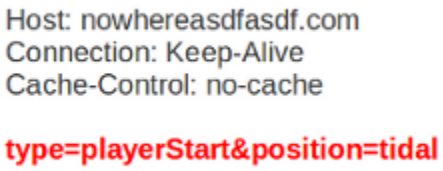
Example to make sure *User-Agent* is after *Host*, but not necessarily directly after:

```
alert http any any -> any any (http.header_names; \
        content:"|0d 0a|Host|0d 0a|"; content:"|0a 0d|User-Agent|0d 0a|"; \
        distance:-2; sid:1;)
```

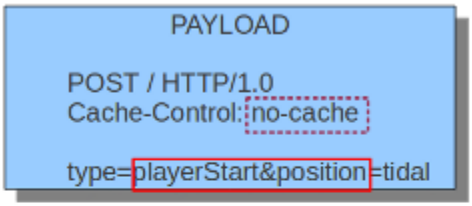
### 6.12.20 http.request_body

With the `http.request_body` content modifier, it is possible to match specifically and only on the HTTP request body. The keyword can be used in combination with all previously mentioned content modifiers like `distance`, `offset`, `nocase`, `within`, etc.

Example of `http.request_body` in a HTTP request:




Example of the purpose of `http.client_body`:




content:"playerStart&position"; http_client_body; ✓

content:"no-cache"; http_client_body; ✗

content:"playerStart"; depth: 16; http_client_body; ✓

content:"playerStart"; http_client_body;
content:"&position"; distance:0; within:9 ✓

Note: how much of the request/client body is inspected is controlled in the *libhtp configuration section* via the `request-body-limit` setting.

`http.request_body` replaces the previous keyword name: `` `http_client_body ``. You may continue +to use the previous name, but it's recommended that rules be converted to use +the new name.

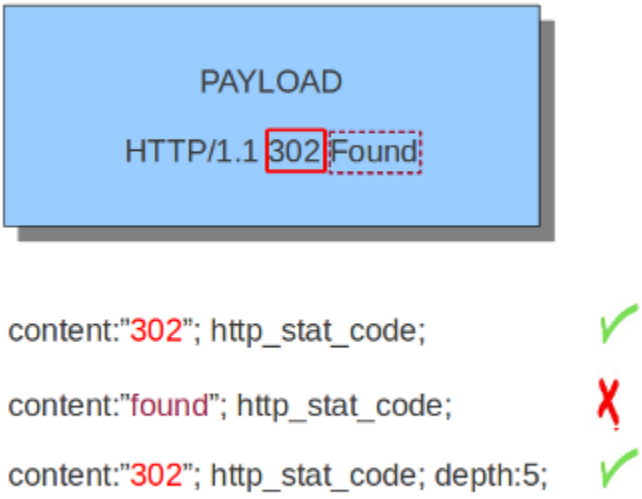
### 6.12.21 http.stat_code

With the `http.stat_code` content modifier, it is possible to match specifically and only on the HTTP status code buffer. The keyword can be used in combination with all previously mentioned content modifiers like `distance`, `offset`, `nocase`, `within`, etc.

Example of `http.stat_code` in a HTTP response:

```
HTTP/1.1 302 Found
```

Example of the purpose of `http.stat_code`:



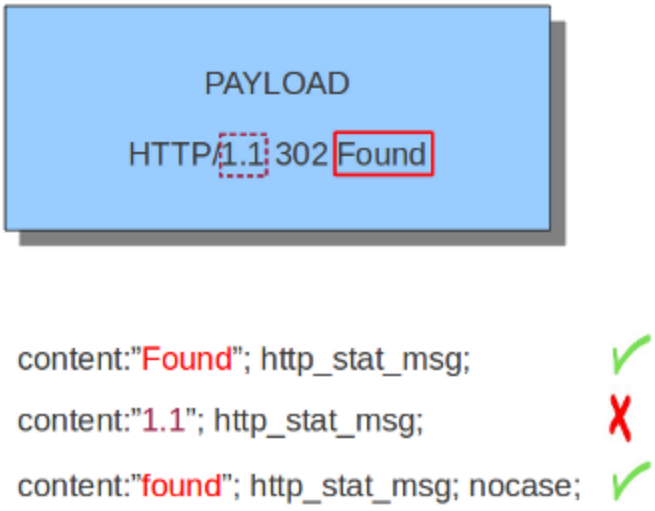

### 6.12.22 http.stat_msg

With the `http.stat_msg` content modifier, it is possible to match specifically and only on the HTTP status message buffer. The keyword can be used in combination with all previously mentioned content modifiers like `depth`, `distance`, `offset`, `nocase` and `within`.

Example of `http.stat_msg` in a HTTP response:

```
HTTP/1.1 302 Found
```

Example of the purpose of `http.stat_msg`:



### 6.12.23 http.response_line

The `http.response_line` forces the whole HTTP response line to be inspected.

Example:

```
alert http any any -> any any (http.response_line; content:"HTTP/1.0 200 OK"; sid:1;)
```

### 6.12.24 http.response_body

With the `http.response_body` content modifier, it is possible to match specifically and only on the HTTP response body. The keyword can be used in combination with all previously mentioned content modifiers like `distance`, `offset`, `nocase`, `within`, etc.

Note: how much of the response/server body is inspected is controlled in your *libhtp configuration section* via the `response-body-limit` setting.

#### Notes

- Using `http.response_body` is similar to having content matches that come after `file_data` except that it doesn't permanently (unless reset) set the detection pointer to the beginning of the server response body. i.e. it is not a sticky buffer.
- `http.response_body` will match on gzip decoded data just like `file_data` does.
- Since `http.response_body` matches on a server response, it can't be used with the `to_server` or `from_client` flow directives.
- Corresponding PCRE modifier: `Q`
- further notes at the `file_data` section below.

`http.response_body` replaces the previous keyword name: `` `http_server_body ``. You may continue +to use the previous name, but it's recommended that rules be converted to use +the new name.

### 6.12.25 http.server

Sticky buffer to match on the HTTP Server headers. Only contains the header value. The \r\n after the header are not part of the buffer.

Example:

```
alert http any any -> any any (flow:to_client; \
        http.server; content:"Microsoft-IIS/6.0"; sid:1;)
```

### 6.12.26 http.location

Sticky buffer to match on the HTTP Location headers. Only contains the header value. The \r\n after the header are not part of the buffer.

Example:

```
alert http any any -> any any (flow:to_client; \
        http.location; content:"http://www.google.com"; sid:1;)
```

### 6.12.27 http.host and http.host.raw

With the `http.host` content modifier, it is possible to match specifically and only the normalized hostname. The `http.host.raw` inspects the raw hostname.

The keyword can be used in combination with most of the content modifiers like `distance`, `offset`, `within`, etc.

The `nocase` keyword is not allowed anymore. Keep in mind that you need to specify a lowercase pattern.

#### Notes

- The `http.host` and `http.host.raw` buffers are populated from either the URI (if the full URI is present in the request like in a proxy request) or the HTTP Host header. If both are present, the URI is used.
- The `http.host` and `http.host.raw` buffers will NOT include the header name, colon, or leading whitespace if populated from the Host header. i.e. they will not include "Host: ".
- The `http.host` and `http.host.raw` buffers do not include a CRLF (0x0D 0x0A) at the end. If you want to match the end of the buffer, use a relative 'isdataat' or a PCRE (although PCRE will be worse on performance).
- The `http.host` buffer is normalized to be all lower case.
- The content match that `http.host` applies to must be all lower case or have the `nocase` flag set.
- `http.host.raw` matches the unnormalized buffer so matching will be case-sensitive (unless `nocase` is set).
- If a request contains multiple "Host" headers, the values will be concatenated in the `http.host` and `http.host.raw` buffers, in the order seen from top to bottom, with a comma and space (", ") between each of them.

  Example request:

```
GET /test.html HTTP/1.1
Host: ABC.com
Accept: */*
Host: efg.net
```

  `http.host` buffer contents:

```
abc.com, efg.net
```

  `http.host.raw` buffer contents:

```
ABC.com, efg.net
```

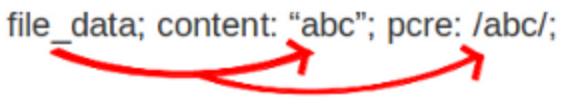
- Corresponding PCRE modifier (`http_host`): `W`
- Corresponding PCRE modifier (`http_raw_host`): `Z`

### 6.12.28 file_data

With `file_data`, the HTTP response body is inspected, just like with `http.response_body`. The `file_data` keyword is a sticky buffer.

Example:

```
alert http any any -> any any (file_data; content:"abc"; content:"xyz";)
```




The `file_data` keyword affects all following content matches, until the `pkt_data` keyword is encountered or it reaches the end of the rule. This makes it a useful shortcut for applying many content matches to the HTTP response body, eliminating the need to modify each content match individually.

As the body of a HTTP response can be very large, it is inspected in smaller chunks.

How much of the response/server body is inspected is controlled in your *libhtp configuration section* via the `response-body-limit` setting.

If the HTTP body is a flash file compressed with ‘deflate’ or ‘lzma’, it can be decompressed and `file_data` can match on the decompress data. Flash decompression must be enabled under `libhtp` configuration:

```
# Decompress SWF files.
# 2 types: 'deflate', 'lzma', 'both' will decompress deflate and lzma
# compress-depth:
# Specifies the maximum amount of data to decompress,
# set 0 for unlimited.
# decompress-depth:
# Specifies the maximum amount of decompressed data to obtain,
# set 0 for unlimited.
swf-decompression:
  enabled: yes
  type: both
  compress-depth: 0
  decompress-depth: 0
```

#### Notes

- If a HTTP body is using gzip or deflate, `file_data` will match on the decompressed data.
- Negated matching is affected by the chunked inspection. E.g. ‘content:!”<html”;’ could not match on the first chunk, but would then possibly match on the 2nd. To avoid this, use a depth setting. The depth setting takes the body size into account. Assuming that the `response-body-minimal-inspect-size` is bigger than 1k, ‘content:!”<html”; depth:1024;’ can only match if the pattern ‘<html’ is absent from the first inspected chunk.
- `file_data` can also be used with SMTP

## 6.13 File Keywords

Suricata comes with several rule keywords to match on various file properties. They depend on properly configured *File Extraction*.

### 6.13.1 filename

Matches on the file name.

Syntax:

```
filename:<string>;
```

Example:

```
filename:"secret";
```

### 6.13.2 fileext

Matches on the extension of a file name.

Syntax:

```
fileext:<string>;
```

Example:

```
fileext:"jpg";
```

### 6.13.3 filemagic

Matches on the information libmagic returns about a file.

Syntax:

```
filemagic:<string>;
```

Example:

```
filemagic:"executable for MS Windows";
```

Note: as libmagic versions differ between installations, the returned information may also slightly change. See also #437.

### 6.13.4 filestore

Stores files to disk if the signature matched.

Syntax:

```
filestore:<direction>,<scope>;
```

direction can be:

- request/to_server: store a file in the request / to_server direction
- response/to_client: store a file in the response / to_client direction
- both: store both directions

scope can be:

- file: only store the matching file (for filename,fileext,filemagic matches)
- tx: store all files from the matching HTTP transaction
- ssn/flow: store all files from the TCP session/flow.

If direction and scope are omitted, the direction will be the same as the rule and the scope will be per file.

### 6.13.5 filemd5

Match file *MD5* against list of MD5 checksums.

Syntax:

```
filemd5:[!]filename;
```

The filename is expanded to include the rule dir. In the default case it will become /etc/suricata/rules/filename. Use the exclamation mark to get a negated match. This allows for white listing.

Examples:

```
filemd5:md5-blacklist;
filemd5:!md5-whitelist;
```

*File format*

The file format is simple. It's a text file with a single md5 per line, at the start of the line, in hex notation. If there is extra info on the line it is ignored.

Output from md5sum is fine:

```
2f8d0355f0032c3e6311c6408d7c2dc2  util-path.c
b9cf5cf347a70e02fde975fc4e117760  util-pidfile.c
02aaa6c3f4dbae65f5889eeb8f2bbb8d  util-pool.c
dd5fc1ee7f2f96b5f12d1a854007a818  util-print.c
```

Just MD5's are good as well:

```
2f8d0355f0032c3e6311c6408d7c2dc2
b9cf5cf347a70e02fde975fc4e117760
02aaa6c3f4dbae65f5889eeb8f2bbb8d
dd5fc1ee7f2f96b5f12d1a854007a818
```

*Memory requirements*

Each MD5 uses 16 bytes of memory. 20 Million MD5's use about 310 MiB of memory.

See also: https://blog.inliniac.net/2012/06/09/suricata-md5-blacklisting/

### 6.13.6 filesha1

Match file SHA1 against list of SHA1 checksums.

Syntax:

```
filesha1:[!]filename;
```

The filename is expanded to include the rule dir. In the default case it will become /etc/suricata/rules/filename. Use the exclamation mark to get a negated match. This allows for white listing.

Examples:

```
filesha1:sha1-blacklist;
filesha1:!sha1-whitelist;
```

*File format*

Same as md5 file format.

### 6.13.7 filesha256

Match file SHA256 against list of SHA256 checksums.

Syntax:

```
filesha256:[!]filename;
```

The filename is expanded to include the rule dir. In the default case it will become /etc/suricata/rules/filename. Use the exclamation mark to get a negated match. This allows for white listing.

Examples:

```
filesha256:sha256-blacklist;
filesha256:!sha256-whitelist;
```

*File format*

Same as md5 file format.

### 6.13.8 filesize

Match on the size of the file as it is being transferred.

Syntax:

```
filesize:<value>;
```

Possible units are KB, MB and GB, without any unit the default is bytes.

Examples:

```
filesize:100; # exactly 100 bytes
filesize:100<>200; # greater than 100 and smaller than 200
filesize:>100MB; # greater than 100 megabytes
filesize:<100MB; # smaller than 100 megabytes
```

**Note**: For files that are not completely tracked because of packet loss or stream.reassembly.depth being reached on the "greater than" is checked. This is because Suricata can know a file is bigger than a value (it has seen some of it already), but it can't know if the final size would have been within a range, an exact value or smaller than a value.

## 6.14 DNS Keywords

There are some more content modifiers (If you are unfamiliar with content modifiers, please visit the page *Payload Keywords* These ones make sure the signature checks a specific part of the network-traffic.

### 6.14.1 dns.opcode

This keyword matches on the **opcode** found in the DNS header flags.

#### Syntax

```
dns.opcode:[!]<number>
```

#### Examples

Match on DNS requests and responses with **opcode** 4:

```
dns.opcode:4;
```

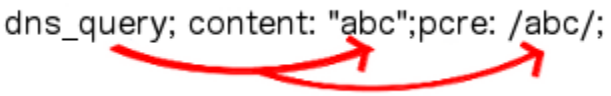
Match on DNS requests whre the **opcode** is NOT 0:

```
dns.opcode:!0;
```

### 6.14.2 dns.query

With **dns.query** the DNS request queries are inspected. The dns.query keyword works a bit different from the normal content modifiers. When used in a rule all contents following it are affected by it. Example:

> alert dns any any -> any any (msg:"Test dns.query option"; dns.query; content:"google"; nocase; sid:1;)




The **dns.query** keyword affects all following contents, until pkt_data is used or it reaches the end of the rule.

**Note:** **dns.query** is equivalent to the older **dns_query**.

#### Normalized Buffer

Buffer contains literal domain name

- <length> values (as seen in a raw DNS request) are literal '.' characters
- no leading <length> value
- No terminating NULL (0x00) byte (use a negated relative `isdataat` to match the end)

Example DNS request for "mail.google.com" (for readability, hex values are encoded between pipes):

DNS query on the wire (snippet):

```
|04|mail|06|google|03|com|00|
```

`dns.query` buffer:

```
mail.google.com
```

## 6.15 SSL/TLS Keywords

Suricata comes with several rule keywords to match on various properties of TLS/SSL handshake. Matches are string inclusion matches.

### 6.15.1 tls.cert_subject

Match TLS/SSL certificate Subject field.

Examples:

```
tls.cert_subject; content:"CN=*.googleusercontent.com"; isdataat:!1,relative;
tls.cert_subject; content:"google.com"; nocase; pcre:"/google.com$/";
```

`tls.cert_subject` is a 'sticky buffer'.

`tls.cert_subject` can be used as `fast_pattern`.

`tls.cert_subject` replaces the previous keyword name: `tls_cert_subject`. You may continue to use the previous name, but it's recommended that rules be converted to use the new name.

### 6.15.2 tls.cert_issuer

Match TLS/SSL certificate Issuer field.

Examples:

```
tls.cert_issuer; content:"WoSign"; nocase; isdataat:!1,relative;
tls.cert_issuer; content:"StartCom"; nocase; pcre:"/StartCom$/";
```

`tls.cert_issuer` is a 'sticky buffer'.

`tls.cert_issuer` can be used as `fast_pattern`.

`tls.cert_issuer` replaces the previous keyword name: `tls_cert_issuer`. You may continue to use the previous name, but it's recommended that rules be converted to use the new name.

### 6.15.3 tls.cert_serial

Match on the serial number in a certificate.

Example:

```
alert tls any any -> any any (msg:"match cert serial"; \
  tls.cert_serial; content:"5C:19:B7:B1:32:3B:1C:A1"; sid:200012;)
```

`tls.cert_serial` is a 'sticky buffer'.

`tls.cert_serial` can be used as `fast_pattern`.

`tls.cert_serial` replaces the previous keyword name: `tls_cert_serial`. You may continue to use the previous name, but it's recommended that rules be converted to use the new name.

### 6.15.4 tls.cert_fingerprint

Match on the SHA-1 fingerprint of the certificate.

Example:

```
alert tls any any -> any any (msg:"match cert fingerprint"; \
  tls.cert_fingerprint; \
  content:"4a:a3:66:76:82:cb:6b:23:bb:c3:58:47:23:a4:63:a7:78:a4:a1:18"; \
  sid:200023;)
```

`tls.cert_fingerprint` is a 'sticky buffer'.

`tls.cert_fingerprint` can be used as `fast_pattern`.

`tls.cert_fingerprint` replaces the previous keyword name: `tls_cert_fingerprint` may continue to use the previous name, but it's recommended that rules be converted to use the new name.

### 6.15.5 tls.sni

Match TLS/SSL Server Name Indication field.

Examples:

```
tls.sni; content:"oisf.net"; nocase; isdataat:!1,relative;
tls.sni; content:"oisf.net"; nocase; pcre:"/oisf.net$/";
```

`tls.sni` is a 'sticky buffer'.

`tls.sni` can be used as `fast_pattern`.

`tls.sni` replaces the previous keyword name: `tls_sni`. You may continue to use the previous name, but it's recommended that rules be converted to use the new name.

### 6.15.6 tls_cert_notbefore

Match on the NotBefore field in a certificate.

Example:

```
alert tls any any -> any any (msg:"match cert NotBefore"; \
  tls_cert_notbefore:1998-05-01<>2008-05-01; sid:200005;)
```

### 6.15.7 tls_cert_notafter

Match on the NotAfter field in a certificate.

Example:

```
alert tls any any -> any any (msg:"match cert NotAfter"; \
  tls_cert_notafter:>2015; sid:200006;)
```

### 6.15.8 tls_cert_expired

Match returns true if certificate is expired. It evaluates the validity date from the certificate.

Usage:

```
tls_cert_expired;
```

### 6.15.9 tls_cert_valid

Match returns true if certificate is not expired. It only evaluates the validity date. It does *not* do cert chain validation. It is the opposite of `tls_cert_expired`.

Usage:

```
tls_cert_valid;
```

### 6.15.10 tls.certs

Do a "raw" match on each of the certificates in the TLS certificate chain.

Example:

```
alert tls any any -> any any (msg:"match bytes in TLS cert"; tls.certs; \
  content:"|06 09 2a 86|"; sid:200070;)
```

`tls.certs` is a 'sticky buffer'.

`tls.certs` can be used as `fast_pattern`.

### 6.15.11 tls.version

Match on negotiated TLS/SSL version.

Supported values: "1.0", "1.1", "1.2", "1.3"

It is also possible to match versions using a hex string.

Examples:

```
tls.version:1.2;
tls.version:0x7f12;
```

The first example matches TLSv1.2, whilst the last example matches TLSv1.3 draft 16.

### 6.15.12 ssl_version

Match version of SSL/TLS record.

Supported values "sslv2", "sslv3", "tls1.0", "tls1.1", "tls1.2", "tls1.3"

Example:

```
alert tls any any -> any any (msg:"match TLSv1.2"; \
  ssl_version:tls1.2; sid:200030;)
```

It is also possible to match on several versions at the same time.

Example:

```
alert tls any any -> any any (msg:"match SSLv2 and SSLv3"; \
  ssl_version:sslv2,sslv3; sid:200031;)
```

### 6.15.13 tls.subject

Match TLS/SSL certificate Subject field.

example:

```
tls.subject:"CN=*.googleusercontent.com"
```

Case sensitive, can't use 'nocase'.

Legacy keyword. `tls.cert_subject` is the replacement.

### 6.15.14 tls.issuerdn

match TLS/SSL certificate IssuerDN field

example:

```
tls.issuerdn:!"CN=Google-Internet-Authority"
```

Case sensitive, can't use 'nocase'.

Legacy keyword. `tls.cert_issuer` is the replacement.

### 6.15.15 tls.fingerprint

match TLS/SSL certificate SHA1 fingerprint

example:

```
tls.fingerprint:!"f3:40:21:48:70:2c:31:bc:b5:aa:22:ad:63:d6:bc:2e:b3:46:e2:5a"
```

Case sensitive, can't use 'nocase'.

The tls.fingerprint buffer is lower case so you must use lower case letters for this to match.

### 6.15.16 tls.store

store TLS/SSL certificate on disk

### 6.15.17 ssl_state

The `ssl_state` keyword matches the state of the SSL connection. The possible states are `client_hello`, `server_hello`, `client_keyx`, `server_keyx` and `unknown`. You can specify several states with `|` (OR) to check for any of the specified states.

Negation support is not available yet, see https://redmine.openinfosecfoundation.org/issues/1231

## 6.16 SSH Keywords

Suricata has several rule keywords to match on different elements of SSH connections.

### 6.16.1 ssh.proto

Match on the version of the SSH protocol used. `ssh.proto` is a sticky buffer, and can be used as a fast pattern. `ssh.proto` replaces the previous buffer name: `ssh_proto`. You may continue to use the previous name, but it's recommended that existing rules be converted to use the new name.

Format:

```
ssh.proto;
```

Example:

alert ssh any any -> any any (msg:"match SSH protocol version"; ssh.proto; content:"2.0"; sid:1000010;)

The example above matches on SSH connections with SSH version 2.0.

### 6.16.2 ssh.software

Match on the software string from the SSH banner. `ssh.software` is a sticky buffer, and can be used as fast pattern.

`ssh.software` replaces the previous keyword names: `ssh_software` & `ssh.softwareversion`. You may continue to use the previous name, but it's recommended that rules be converted to use the new name.

Format:

```
ssh.software;
```

Example:

alert ssh any any -> any any (msg:"match SSH software string"; ssh.software; content:"openssh"; nocase; sid:1000020;)

The example above matches on SSH connections where the software string contains "openssh".

### 6.16.3 ssh.protoversion

Matches on the version of the SSH protocol used. A value of `2_compat` includes SSH version 1.99.

Format:

```
ssh.protoversion:[0-9](\.[0-9])?|2_compat;
```

Example:

alert ssh any any -> any any (msg:"SSH v2 compatible"; ssh.protoversion:2_compat; sid:1;)

The example above matches on SSH connections with SSH version 2 or 1.99.

alert ssh any any -> any any (msg:"SSH v1.10"; ssh.protoversion:1.10; sid:1;)

The example above matches on SSH connections with SSH version 1.10 only.

### 6.16.4 ssh.softwareversion

This keyword has been deprecated. Please use `ssh.software` instead. Matches on the software string from the SSH banner.

Example:

alert ssh any any -> any any (msg:"match SSH software string"; ssh.softwareversion:"OpenSSH"; sid:10000040;)

## 6.17 JA3 Keywords

Suricata comes with a JA3 integration (https://github.com/salesforce/ja3). JA3 is used to fingerprint TLS clients.

JA3 must be enabled in the Suricata config file (set 'app-layer.protocols.tls.ja3-fingerprints' to 'yes').

JA3 also requires Suricata to be built with LibNSS support.

### 6.17.1 ja3.hash

Match on JA3 hash (md5).

Example:

```
alert tls any any -> any any (msg:"match JA3 hash"; \
    ja3.hash; content:"e7eca2baf4458d095b7f45da28c16c34"; \
    sid:100001;)
```

`ja3.hash` is a 'sticky buffer'.

`ja3.hash` can be used as `fast_pattern`.

`ja3.hash` replaces the previous keyword name: `ja3_hash`. You may continue to use the previous name, but it's recommended that rules be converted to use the new name.

### 6.17.2 ja3.string

Match on JA3 string.

Example:

```
alert tls any any -> any any (msg:"match JA3 string"; \
    ja3.string; content:"19-20-21-22"; \
    sid:100002;)
```

`ja3.string` is a 'sticky buffer'.

`ja3.string` can be used as `fast_pattern`.

`ja3.string` replaces the previous keyword name: `ja3_string`. You may continue to use the previous name, but it's recommended that rules be converted to use the new name.

### 6.17.3 ja3s.hash

Match on JA3S hash (md5).

Example:

```
alert tls any any -> any any (msg:"match JA3S hash"; \
    ja3s.hash; content:"b26c652e0a402a24b5ca2a660e84f9d5"; \
    sid:100003;)
```

`ja3s.hash` is a 'sticky buffer'.

`ja3s.hash` can be used as `fast_pattern`.

### 6.17.4 ja3s.string

Match on JA3S string.

Example:

```
alert tls any any -> any any (msg:"match on JA3S string"; \
    ja3s.string; content:"771,23-35"; sid:100004;)
```

`ja3s.string` is a 'sticky buffer'.

`ja3s.string` can be used as `fast_pattern`.

## 6.18 Modbus Keyword

The modbus keyword can be used for matching on various properties of Modbus requests.

There are three ways of using this keyword:

- matching on functions properties with the setting "function";
- matching on directly on data access with the setting "access";
- matching on unit identifier with the setting "unit" only or with the previous setting "function" or "access".

With the setting **function**, you can match on:

- an action based on a function code field and a sub-function code when applicable;
- one of three categories of Modbus functions;
- public functions that are publicly defined (setting "public")
- user-defined functions (setting "user")
- reserved functions that are dedicated to proprietary extensions of Modbus (keyword "reserved")
- one of the two sub-groups of public functions:
  - assigned functions whose definition is already given in the Modbus specification (keyword "assigned");
  - unassigned functions, which are reserved for future use (keyword "unassigned").

Syntax:

```
modbus: function <value>
modbus: function <value>, subfunction <value>
modbus: function [!] <assigned | unassigned | public | user | reserved | all>
```

Sign '!' is negation

Examples:

```
modbus: function 21                # Write File record function
modbus: function 4, subfunction 4  # Force Listen Only Mode (Diagnostics) function
modbus: function assigned          # defined by Modbus Application Protocol␣
↪Specification V1.1b3
```

(continues on next page)

(continued from previous page)

```
modbus: function public             # validated by the Modbus.org community
modbus: function user               # internal use and not supported by the␣
↪specification
modbus: function reserved           # used by some companies for legacy products and␣
↪not available for public use
modbus: function !reserved          # every function but reserved function
```

With the **access** setting, you can match on:

- a type of data access (read or write);
- one of primary tables access (Discretes Input, Coils, Input Registers and Holding Registers);
- a range of addresses access;
- a written value.

Syntax:

```
modbus: access <read | write>
modbus: access read <discretes | coils | input | holding>
modbus: access read <discretes | coils | input | holding>, address <value>
modbus: access write < coils | holding>
modbus: access write < coils | holding>, address <value>
modbus: access write < coils | holding>, address <value>, value <value>
```

With _<value>_ setting matches on the address or value as it is being accessed or written as follows:

```
address 100      # exactly address 100
address 100<>200 # greater than address 100 and smaller than address 200
address >100     # greater than address 100
address <100     # smaller than address 100
```

Examples:

```
modbus: access read                                     # Read access
modbus: access write                                    # Write access
modbus: access read input                               # Read access to Discretes␣
↪Input table
modbus: access write coils                              # Write access to Coils table
modbus: access read discretes, address <100             # Read access at address␣
↪smaller than 100 of Discretes Input table
modbus: access write holding, address 500, value >200   # Write value greater than 200␣
↪at address 500 of Holding Registers table
```

With the setting **unit**, you can match on:

- a MODBUS slave address of a remote device connected on the sub-network behind a bridge or a gateway. The destination IP address identifies the bridge itself and the bridge uses the MODBUS unit identifier to forward the request to the right slave device.

Syntax:

```
modbus: unit <value>
modbus: unit <value>, function <value>
modbus: unit <value>, function <value>, subfunction <value>
modbus: unit <value>, function [!] <assigned | unassigned | public | user | reserved␣
↪| all>
modbus: unit <value>, access <read | write>
```

(continues on next page)

(continued from previous page)

```
modbus: unit <value>, access read <discretes | coils | input | holding>
modbus: unit <value>, access read <discretes | coils | input | holding>, address
↪<value>
modbus: unit <value>, access write < coils | holding>
modbus: unit <value>, access write < coils | holding>, address <value>
modbus: unit <value>, access write < coils | holding>, address <value>, value <value>
```

With _<value>_ setting matches on the address or value as it is being accessed or written as follows:

```
unit 10     # exactly unit identifier 10
unit 10<>20 # greater than unit identifier 10 and smaller than unit identifier 20
unit >10    # greater than unit identifier 10
unit <10    # smaller than unit identifier 10
```

Examples:

```
modbus: unit 10                                                    # Unit␣
↪identifier 10
modbus: unit 10, function 21                                       # Unit␣
↪identifier 10 and write File record function
modbus: unit 10, function 4, subfunction 4                         # Unit␣
↪identifier 10 and force Listen Only Mode (Diagnostics) function
modbus: unit 10, function assigned                                 # Unit␣
↪identifier 10 and assigned function
modbus: unit 10, function !reserved                                # Unit␣
↪identifier 10 and every function but reserved function
modbus: unit 10, access read                                       # Unit␣
↪identifier 10 and Read access
modbus: unit 10, access write coils                                # Unit␣
↪identifier 10 and Write access to Coils table
modbus: unit >10, access read discretes, address <100              # Greater than␣
↪unit identifier 10 and Read access at address smaller than 100 of Discretes Input␣
↪table
modbus: unit 10<>20, access write holding, address 500, value >200 # Greater than␣
↪unit identifier 10 and smaller than unit identifier 20 and Write value greater than␣
↪200 at address 500 of Holding Registers table
```

(cf. http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf)

**Note:** Address of read and write are starting at 1. So if your system is using a start at 0, you need to add 1 the address values.

**Note:** According to MODBUS Messaging on TCP/IP Implementation Guide V1.0b, it is recommended to keep the TCP connection opened with a remote device and not to open and close it for each MODBUS/TCP transaction. In that case, it is important to set the depth of the stream reassembling as unlimited (stream.reassembly.depth: 0)

**Note:** According to MODBUS Messaging on TCP/IP Implementation Guide V1.0b, the MODBUS slave device addresses on serial line are assigned from 1 to 247 (decimal). Address 0 is used as broadcast address.

(cf. http://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf)

Paper and presentation (in french) on Modbus support are available : http://www.ssi.gouv.fr/agence/publication/detection-dintrusion-dans-les-systemes-industriels-suricata-et-le-cas-modbus/

## 6.19 DNP3 Keywords

The DNP3 keywords can be used to match on fields in decoded DNP3 messages. The keywords are based on Snort's DNP3 keywords and aim to be 100% compatible.

### 6.19.1 dnp3_func

This keyword will match on the application function code found in DNP3 request and responses. It can be specified as the integer value or the symbolic name of the function code.

#### Syntax

```
dnp3_func:<value>;
```

Where value is one of:

- An integer value between 0 and 255 inclusive.
- Function code name:
  - confirm
  - read
  - write
  - select
  - operate
  - direct_operate
  - direct_operate_nr
  - immed_freeze
  - immed_freeze_nr
  - freeze_clear
  - freeze_clear_nr
  - freeze_at_time
  - freeze_at_time_nr
  - cold_restart
  - warm_restart
  - initialize_data
  - initialize_appl
  - start_appl
  - stop_appl
  - save_config
  - enable_unsolicited
  - disable_unsolicited

- assign_class
- delay_measure
- record_current_time
- open_file
- close_file
- delete_file
- get_file_info
- authenticate_file
- abort_file
- activate_config
- authenticate_req
- authenticate_err
- response
- unsolicited_response
- authenticate_resp

### 6.19.2 dnp3_ind

This keyword matches on the DNP3 internal indicator flags in the response application header.

#### Syntax

```
dnp3_ind:<flag>{,<flag>...}
```

Where flag is the name of the internal indicator:

- all_stations
- class_1_events
- class_2_events
- class_3_events
- need_time
- local_control
- device_trouble
- device_restart
- no_func_code_support
- object_unknown
- parameter_error
- event_buffer_overflow
- already_executing

- config_corrupt
- reserved_2
- reserved_1

This keyword will match of any of the flags listed are set. To match on multiple flags (AND type match), use dnp3_ind for each flag that must be set.

#### Examples

```
dnp3_ind:all_stations;
```

```
dnp3_ind:class_1_events,class_2_events;
```

### 6.19.3 dnp3_obj

This keyword matches on the DNP3 application data objects.

#### Syntax

```
dnp3_obj:<group>,<variation>
```

Where <group> and <variation> are integer values between 0 and 255 inclusive.

### 6.19.4 dnp3_data

This keyword will cause the following content options to match on the re-assembled application buffer. The reassembled application buffer is a DNP3 fragment with CRCs removed (which occur every 16 bytes), and will be the complete fragment, possibly reassembled from multiple DNP3 link layer frames.

#### Syntax

```
dnp3_data;
```

#### Example

```
dnp3_data; content:"|c3 06|";
```

## 6.20 ENIP/CIP Keywords

The enip_command and cip_service keywords can be used for matching on various properties of ENIP requests.

There are three ways of using this keyword:

- matching on ENIP command with the setting “enip_command”;
- matching on CIP Service with the setting “cip_service”.

- matching both the ENIP command and the CIP Service with "enip_command" and "cip_service" together

For the ENIP command, we are matching against the command field found in the ENIP encapsulation.

For the CIP Service, we use a maximum of 3 comma separated values representing the Service, Class and Attribute. These values are described in the CIP specification. CIP Classes are associated with their Service, and CIP Attributes are associated with their Service. If you only need to match up until the Service, then only provide the Service value. If you want to match to the CIP Attribute, then you must provide all 3 values.

Syntax:

```
enip_command:<value>
cip_service:<value(s)>
enip_command:<value>, cip_service:<value(s)>
```

Examples:

```
enip_command:99
cip_service:75
cip_service:16,246,6
enip_command:111, cip_service:5
```

(cf. http://read.pudn.com/downloads166/ebook/763211/EIP-CIP-V1-1.0.pdf)

Information on the protocol can be found here: http://literature.rockwellautomation.com/idc/groups/literature/documents/wp/enet-wp001_-en-p.pdf

## 6.21 FTP/FTP-DATA Keywords

### 6.21.1 ftpdata_command

Filter ftp-data channel based on command used on the FTP command channel. Currently supported commands are RETR (get on a file) and STOR (put on a file).

Syntax:

```
ftpdata_command:(retr|stor)
```

Examples:

```
ftpdata_command:retr
ftpdata_command:stor
```

Signature example:

```
alert ftp-data any any -> any any (msg:"FTP store password"; filestore; filename:
↪"password"; ftpdata_command:stor; sid:3; rev:1;)
```

### 6.21.2 ftpbounce

Detect FTP bounce attacks.

Syntax:

```
ftpbounce
```

## 6.22 Kerberos Keywords

### 6.22.1 krb5_msg_type

Kerberos message type (integer).

Values are defined in RFC4120. Common values are

- 10 (AS-REQ)
- 11 (AS-REP)
- 12 (TGS-REQ)
- 13 (TGS-REP)
- 14 (AP-REQ)
- 15 (AP-REP)
- 30 (ERROR)

Syntax:

```
krb5_msg_type:<number>
```

Signature example:

```
alert krb5 any any -> any any (msg:"Kerberos 5 AS-REQ message"; krb5_msg_type:10;␣
↪sid:3; rev:1;)
```

### 6.22.2 krb5_cname

Kerberos client name, provided in the ticket (for AS-REQ and TGS-REQ messages).

If the client name from the Kerberos message is composed of several parts, the name is compared to each part and the match will succeed if any is identical.

Comparison is case-sensitive.

Syntax:

```
krb5_cname; content:"name";
```

Signature example:

```
alert krb5 any any -> any any (msg:"Kerberos 5 des server name"; krb5_cname; content:
↪"des"; sid:4; rev:1;)
```

`krb5_cname` is a 'sticky buffer'.

`krb5_cname` can be used as `fast_pattern`.

### 6.22.3 krb5_sname

Kerberos server name, provided in the ticket (for AS-REQ and TGS-REQ messages) or in the error message.

If the server name from the Kerberos message is composed of several parts, the name is compared to each part and the match will succeed if any is identical.

Comparison is case-sensitive.

Syntax:

```
krb5_sname; content:"name";
```

Signature example:

```
alert krb5 any any -> any any (msg:"Kerberos 5 krbtgt server name"; krb5_sname;␣
↪content:"krbtgt"; sid:5; rev:1;)
```

`krb5_sname` is a 'sticky buffer'.

`krb5_sname` can be used as `fast_pattern`.

### 6.22.4 krb5_err_code

Kerberos error code (integer). This field is matched in Kerberos error messages only.

For a list of error codes, refer to RFC4120 section 7.5.9.

Syntax:

```
krb5_err_code:<number>
```

Signature example:

```
alert krb5 any any -> any any (msg:"Kerberos 5 error C_PRINCIPAL_UNKNOWN"; krb5_err_
↪code:6; sid:6; rev:1;)
```

### 6.22.5 krb5.weak_encryption (event)

Event raised if the encryption parameters selected by the server are weak or deprecated. For example, using a key size smaller than 128, or using deprecated ciphers like DES.

Syntax:

```
app-layer-event:krb5.weak_encryption
```

Signature example:

```
alert krb5 any any -> any any (msg:"SURICATA Kerberos 5 weak encryption parameters";␣
↪flow:to_client; app-layer-event:krb5.weak_encryption; classtype:protocol-command-
↪decode; sid:2226001; rev:1;)
```

### 6.22.6 krb5.malformed_data (event)

Event raised in case of a protocol decoding error.

Syntax:

```
app-layer-event:krb5.malformed_data
```

Signature example:

```
alert krb5 any any -> any any (msg:"SURICATA Kerberos 5 malformed request data";␣
↪flow:to_server; app-layer-event:krb5.malformed_data; classtype:protocol-command-
↪decode; sid:2226000; rev:1;)
```

## 6.23 SNMP keywords

### 6.23.1 snmp.version

SNMP protocol version (integer). Expected values are 1, 2 (for version 2c) or 3.

Syntax:

```
snmp.version:[op]<number>
```

The version can be matched exactly, or compared using the _op_ setting:

```
snmp.version:3     # exactly 3
snmp.version:<3    # smaller than 3
snmp.version:>=2   # greater or equal than 2
```

Signature example:

```
alert snmp any any -> any any (msg:"old SNMP version (<3)"; snmp.version:<3; sid:1;␣
↪rev:1;)
```

### 6.23.2 snmp.community

SNMP community strings are like passwords for SNMP messages in version 1 and 2c. In version 3, the community string is likely to be encrypted. This keyword will not match if the value is not accessible.

The default value for the read-only community string is often “public”, and “private” for the read-write community string.

Comparison is case-sensitive.

Syntax:

```
snmp.community; content:"private";
```

Signature example:

```
alert snmp any any -> any any (msg:"SNMP community private"; snmp.community; content:
↪"private"; sid:2; rev:1;)
```

`snmp.community` is a ‘sticky buffer’.

`snmp.community` can be used as `fast_pattern`.

### 6.23.3 snmp.pdu_type

SNMP PDU type (integer).

Common values are:

- 0: GetRequest
- 1: GetNextRequest
- 2: Response
- 3: SetRequest
- 4: TrapV1 (obsolete, was the old Trap-PDU in SNMPv1)
- 5: GetBulkRequest
- 6: InformRequest
- 7: TrapV2
- 8: Report

This keyword will not match if the value is not accessible within (for ex, an encrypted SNMP v3 message).

Syntax:

```
snmp.pdu_type:<number>
```

Signature example:

```
alert snmp any any -> any any (msg:"SNMP response"; snmp.pdu_type:2; sid:3; rev:1;)
```

## 6.24 Base64 keywords

Suricata supports decoding base64 encoded data from buffers and matching on the decoded data.

This is achieved by using two keywords, `base64_decode` and `base64_data`. Both keywords must be used in order to generate an alert.

### 6.24.1 base64_decode

Decodes base64 data from a buffer and makes it available for the base64_data function.

Syntax:

```
base64_decode:bytes <value>, offset <value>, relative;
```

The `bytes` option specifies how many bytes Suricata should decode and make available for base64_data. The decoding will stop at the end of the buffer.

The `offset` option specifies how many bytes Suricata should skip before decoding. Bytes are skipped relative to the start of the payload buffer if the `relative` is not set.

The `relative` option makes the decoding start relative to the previous content match. Default behavior is to start at the beginning of the buffer. This option makes `offset` skip bytes relative to the previous match.

---

**Note:** Regarding `relative` and `base64_decode`:

The content match that you want to decode relative to must be the first match in the stream.

---

### 6.24.2 base64_data

base64_data is a `sticky buffer`.

Enables content matching on the data previously decoded by base64_decode.

### 6.24.3 Example

Here is an example of a rule matching on the base64 encoded string "test" that is found inside the http_uri buffer.

It starts decoding relative to the known string "somestring" with the known offset of 1. This must be the first occurrence of "somestring" in the buffer.

Example:

```
Buffer content:
http_uri = "GET /en/somestring&dGVzdAo=&not_base64"

Rule:
alert http any any -> any any (msg:"Example"; http.uri; content:"somestring"; \
     base64_decode:bytes 8, offset 1, relative; \
     base64_data; content:"test"; sid:10001; rev:1;)

Buffer content:
http_uri = "GET /en/somestring&dGVzdAo=&not_base64"

Rule:
alert http any any -> any any (msg:"Example"; content:"somestring"; http_uri; \
     base64_decode:bytes 8, offset 1, relative; \
     base64_data; content:"test"; sid:10001; rev:1;)
```

## 6.25 SIP Keywords

The SIP keywords are implemented as sticky buffers and can be used to match on fields in SIP messages.

| Keyword | Direction |
|---|---|
| sip.method | Request |
| sip.uri | Request |
| sip.request_line | Request |
| sip.stat_code | Response |
| sip.stat_msg | Response |
| sip.response_line | Response |
| sip.protocol | Both |

### 6.25.1 sip.method

This keyword matches on the method found in a SIP request.

#### Syntax

```
sip.method; content:<method>;
```

Examples of methods are:

- INVITE
- BYE
- REGISTER
- CANCEL
- ACK
- OPTIONS

#### Examples

```
sip.method; content:"INVITE";
```

### 6.25.2 sip.uri

This keyword matches on the uri found in a SIP request.

#### Syntax

```
sip.uri; content:<uri>;
```

Where <uri> is an uri that follows the SIP URI scheme.

#### Examples

```
sip.uri; content:"sip:sip.url.org";
```

### 6.25.3 sip.request_line

This keyword forces the whole SIP request line to be inspected.

#### Syntax

```
sip.request_line; content:<request_line>;
```

Where <request_line> is a partial or full line.

#### Examples

```
sip.request_line; content:"REGISTER sip:sip.url.org SIP/2.0"
```

### 6.25.4 sip.stat_code

This keyword matches on the status code found in a SIP response.

#### Syntax

```
sip.stat_code; content:<stat_code>
```

Where <status_code> belongs to one of the following groups of codes:

- 1xx - Provisional Responses
- 2xx - Successful Responses
- 3xx - Redirection Responses
- 4xx - Client Failure Responses
- 5xx - Server Failure Responses
- 6xx - Global Failure Responses

#### Examples

```
sip.stat_code; content:"100";
```

### 6.25.5 sip.stat_msg

This keyword matches on the status message found in a SIP response.

#### Syntax

```
sip.stat_msg; content:<stat_msg>
```

Where <stat_msg> is a reason phrase associated to a status code.

#### Examples

```
sip.stat_msg; content:"Trying";
```

### 6.25.6 sip.response_line

This keyword forces the whole SIP response line to be inspected.

#### Syntax

```
sip.response_line; content:<response_line>;
```

Where <response_line> is a partial or full line.

#### Examples

```
sip.response_line; content:"SIP/2.0 100 OK"
```

### 6.25.7 sip.protocol

This keyword matches the protocol field from a SIP request or response line.

If the response line is 'SIP/2.0 100 OK', then this buffer will contain 'SIP/2.0'

#### Syntax

```
sip.protocol; content:<protocol>
```

Where <protocol> is the SIP protocol version.

#### Example

```
sip.protocol; content:"SIP/2.0"
```

## 6.26 Generic App Layer Keywords

### 6.26.1 app-layer-protocol

Match on the detected app-layer protocol.

Syntax:

```
app-layer-protocol:[!]<protocol>;
```

Examples:

```
app-layer-protocol:ssh;
app-layer-protocol:!tls;
app-layer-protocol:failed;
```

A special value 'failed' can be used for matching on flows in which protocol detection failed. This can happen if Suricata doesn't know the protocol or when certain 'bail out' conditions happen.

#### Bail out conditions

Protocol detection gives up in several cases:

- both sides are inspected and no match was found
- side A detection failed, side B has no traffic at all (e.g. FTP data channel)
- side A detection failed, side B has so little data detection is inconclusive

In these last 2 cases the `app-layer-event:applayer_proto_detection_skipped` is set.

### 6.26.2 app-layer-event

Match on events generated by the App Layer Parsers and the protocol detection engine.

Syntax:

```
app-layer-event:<event name>;
```

Examples:

```
app-layer-event:applayer_mismatch_protocol_both_directions;
app-layer-event:http.gzip_decompression_failed;
```

#### Protocol Detection

##### applayer_mismatch_protocol_both_directions

The toserver and toclient directions have different protocols. For example a client talking HTTP to a SSH server.

##### applayer_wrong_direction_first_data

Some protocol implementations in Suricata have a requirement with regards to the first data direction. The HTTP parser is an example of this.

https://redmine.openinfosecfoundation.org/issues/993

##### applayer_detect_protocol_only_one_direction

Protocol detection only succeeded in one direction. For FTP and SMTP this is expected.

##### applayer_proto_detection_skipped

Protocol detection was skipped because of *Bail out conditions*.

## 6.27 Xbits Keyword

Set, unset, toggle and check for bits stored per host or ip_pair.

Syntax:

```
xbits:<set|unset|isset|isnotset|toggle>,<name>,track <ip_src|ip_dst|ip_pair>;
xbits:<set|unset|isset|toggle>,<name>,track <ip_src|ip_dst|ip_pair> \
    [,expire <seconds>];
xbits:<set|unset|isset|toggle>,<name>,track <ip_src|ip_dst|ip_pair> \
    [,expire <seconds>];
```

### 6.27.1 Notes

- No difference between using `hostbits` and `xbits` with `track ip_<src|dst>`
- If you `set` on a client request and use `track ip_dst`, if you want to match on the server response, you check it (`isset`) with `track ip_src`.
- To not alert, use `noalert;`
- the `toggle` option will flip the value of the xbits.
- See also:
  - https://blog.inliniac.net/2014/12/21/crossing-the-streams-in-suricata/
  - http://www.cipherdyne.org/blog/2013/07/crossing-the-streams-in-ids-signature-languages.html

#### YAML settings

Bits that are stored per host are stored in the Host table. This means that host table settings affect hostsbits and xbits per host.

Bits that are stored per IP pair are stored in the IPPair table. This means that ippair table settings, especially memcap, affect xbits per ip_pair.

#### Threading

Due to subtle timing issues between threads the order of sets and checks can be slightly unpredictable.

#### Unix Socket

Hostbits can be added, removed and listed through the unix socket.

Add:

```
suricatasc -c "add-hostbit <ip> <bit name> <expire in seconds>"
suricatasc -c "add-hostbit 1.2.3.4 blacklist 3600"
```

If an hostbit is added for an existing hostbit, it's expiry timer is updated.

Remove:

```
suricatasc -c "remove-hostbit <ip> <bit name>"
suricatasc -c "remove-hostbit 1.2.3.4 blacklist"
```

List:

```
suricatasc -c "list-hostbit <ip>"
suricatasc -c "list-hostbit 1.2.3.4"
```

This results in:

```
{
    "message":
    {
       "count": 1,
       "hostbits":
            [{
                "expire": 89,
                "name": "blacklist"
            }]
    },
    "return": "OK"
}
```

#### Examples

#### Creating a SSH blacklist

Below is an example of rules incoming to a SSH server.

The first 2 rules match on a SSH software version often used in bots. They drop the traffic and create an 'xbit' 'badssh' for the source ip. It expires in an hour:

```
drop ssh any any -> $MYSERVER 22 (msg:"DROP libssh incoming";    \
  flow:to_server,established; ssh.softwareversion:"libssh";      \
  xbits:set, badssh, track ip_src, expire 3600; sid:4000000005;)
drop ssh any any -> $MYSERVER 22 (msg:"DROP PUTTY incoming";     \
  flow:to_server,established; ssh.softwareversion:"PUTTY";       \
  xbits:set, badssh, track ip_src, expire 3600; sid:4000000007;)
```

Then the following rule simply drops any incoming traffic to that server that is on that 'badssh' list:

```
drop ssh any any -> $MYSERVER 22 (msg:"DROP BLACKLISTED";        \
  xbits:isset, badssh, track ip_src; sid:4000000006;)
```

## 6.28 Thresholding Keywords

Thresholding can be configured per rule and also globally, see *Global-Thresholds*.

*Note: mixing rule and global thresholds is not supported in 1.3 and before. See bug #425.* For the state of the support in 1.4 see *Global thresholds vs rule thresholds*

### 6.28.1 threshold

The threshold keyword can be used to control the rule's alert frequency. It has 3 modes: threshold, limit and both.

Syntax:

```
threshold: type <threshold|limit|both>, track <by_src|by_dst>, count <N>, seconds <T>
```

### type "threshold"

This type can be used to set a minimum threshold for a rule before it generates alerts. A threshold setting of N means on the Nth time the rule matches an alert is generated.

Example:

```
alert tcp !$HOME_NET any -> $HOME_NET 25 (msg:"ET POLICY Inbound Frequent Emails -␣
↪Possible Spambot Inbound"; \
flow:established; content:"mail from|3a|"; nocase;                              ␣
↪                    \
threshold: type threshold, track by_src, count 10, seconds 60;                  ␣
↪                    \
reference:url,doc.emergingthreats.net/2002087; classtype:misc-activity; sid:2002087;␣
↪rev:10;)
```

This signature only generates an alert if we get 10 inbound emails or more from the same server in a time period of one minute.

If a signature sets a flowbit, flowint, etc. those actions are still performed for each of the matches.

> *Rule actions drop (IPS mode) and reject are applied to each packet (not only the one that meets the threshold condition).*

### type "limit"

This type can be used to make sure you're not getting flooded with alerts. If set to limit N, it alerts at most N times.

Example:

```
alert http $HOME_NET any -> any $HTTP_PORTS (msg:"ET USER_AGENTS Internet Explorer 6␣
↪in use - Significant Security Risk"; \
flow:to_server,established; content:"|0d 0a|User-Agent|3a| Mozilla/4.0␣
↪(compatible|3b| MSIE 6.0|3b|";                \
threshold: type limit, track by_src, seconds 180, count 1;                      ␣
↪                                  \
reference:url,doc.emergingthreats.net/2010706; classtype:policy-violation;␣
↪sid:2010706; rev:7;)
```

In this example at most 1 alert is generated per host within a period of 3 minutes if MSIE 6.0 is detected.

If a signature sets a flowbit, flowint, etc. those actions are still performed for each of the matches.

> *Rule actions drop (IPS mode) and reject are applied to each packet (not only the one that meets the limit condition).*

### type "both"

This type is a combination of the "threshold" and "limit" types. It applies both thresholding and limiting.

Example:

```
alert tcp $HOME_NET 5060 -> $EXTERNAL_NET any (msg:"ET VOIP Multiple Unauthorized SIP␣
↪Responses TCP"; \
flow:established,from_server; content:"SIP/2.0 401 Unauthorized"; depth:24;     ␣
↪           \
```

(continues on next page)

(continued from previous page)

```
threshold: type both, track by_src, count 5, seconds 360;
↪           \
reference:url,doc.emergingthreats.net/2003194; classtype:attempted-dos; sid:2003194;
↪rev:6;)
```

This alert will only generate an alert if within 6 minutes there have been 5 or more "SIP/2.0 401 Unauthorized" responses, and it will alert only once in that 6 minutes.

If a signature sets a flowbit, flowint, etc. those actions are still performed for each of the matches.

*Rule actions drop (IPS mode) and reject are applied to each packet.*

### 6.28.2 detection_filter

The detection_filter keyword can be used to alert on every match after a threshold has been reached. It differs from the threshold with type threshold in that it generates an alert for each rule match after the initial threshold has been reached, where the latter will reset it's internal counter and alert again when the threshold has been reached again.

Syntax:

```
detection_filter: track <by_src|by_dst>, count <N>, seconds <T>
```

Example:

```
alert http $EXTERNAL_NET any -> $HOME_NET any \
    (msg:"ET WEB_SERVER WebResource.axd access without t (time) parameter - possible
↪ASP padding-oracle exploit"; \
    flow:established,to_server; content:"GET"; http_method; content:"WebResource.axd
↪"; http_uri; nocase;          \
    content:!"&t="; http_uri; nocase; content:!"&amp|3b|t="; http_uri; nocase;
↪                              \
    detection_filter:track by_src,count 15,seconds 2;
↪                              \
    reference:url,netifera.com/research/; reference:url,www.microsoft.com/technet/
↪security/advisory/2416728.mspx; \
    classtype:web-application-attack; sid:2011807; rev:5;)
```

Alerts each time after 15 or more matches have occurred within 2 seconds.

If a signature sets a flowbit, flowint, etc. those actions are still performed for each of the matches.

*Rule actions drop (IPS mode) and reject are applied to each packet that generate an alert*

## 6.29 IP Reputation Keyword

IP Reputation can be used in rules through a new rule keyword "iprep".

For more information about IP Reputation see *IP Reputation Config* and *IP Reputation Format*.

### 6.29.1 iprep

The iprep directive matches on the IP reputation information for a host.

```
iprep:<side to check>,<category>,<operator>,<reputation score>
```

side to check: <any|src|dst|both>

category: the category short name

operator: <, >, =

reputation score: 1-127

Example:

```
alert ip $HOME_NET any -> any any (msg:"IPREP internal host talking to CnC server";␣
↪flow:to_server; iprep:dst,CnC,>,30; sid:1; rev:1;)
```

This rule will alert when a system in $HOME_NET acts as a client while communicating with any IP in the CnC category that has a reputation score set to greater than 30.

#### IP-only

The "iprep" keyword is compatible to "IP-only" rules. This means that a rule like:

```
alert ip any any -> any any (msg:"IPREP High Value CnC"; iprep:src,CnC,>,100; sid:1;␣
↪rev:1;)
```

will only be checked once per flow-direction.

## 6.30 Datasets

**Warning:** This is an experimental feature. Syntax and functionality may change in future releases.

Using the `dataset` and `datarep` keyword it is possible to match on large amounts of data against any sticky buffer.

For example, to match against a dns black list called `dns-bl`:

```
dns.query; dataset:isset,dns-bl;
```

These keywords are aware of transforms. So to look up a DNS query against a MD5 black list:

```
dns.query; to_md5; dataset:isset,dns-bl;
```

### 6.30.1 Global config (optional)

Datasets can optionally be defined in the main config. Sets can also be declared from the rule syntax.

Example of sets for tracking unique values:

```
datasets:
  ua-seen:
    type: string
    state: ua-seen.lst
  dns-sha256-seen:
```

(continues on next page)

(continued from previous page)

```
    type: sha256
    state: dns-sha256-seen.lst
```

Rules to go with the above:

alert dns any any -> any any (msg:"dns list test"; dns.query; to_sha256; dataset:isset,dns-sha256-seen; sid:123; rev:1;)

alert http any any -> any any (msg: "http user-agent test"; http.user_agent; dataset:set,ua-seen; sid:234; rev:1;)

### 6.30.2 Rule keywords

#### dataset

Datasets are binary: something is in the set or it's not.

Syntax:

```
dataset:<cmd>,<name>,<options>;

dataset:<set|isset|isnotset>,<name> \
    [, type <string|md5|sha256>, save <file name>, load <file name>, state <file name>
↪];
```

**type <type>** the data type: string, md5, sha256

**save <file name>** file name for saving the in-memory data when Suricata exits

**load <file name>** file name for load the data when Suricata starts up

**state** sets both 'save' and 'load' to the same value

---

**Note:** 'load' or 'load' plus 'save' and 'state' cannot be mixed.

---

#### datarep

Data Reputation allows matching data against a reputation list.

Syntax:

```
datarep:<name>,<operator>,<value>, \
    [, load <file name>, type <string|md5|sha256>];
```

Example rules could look like:

```
alert dns any any -> any any (dns.query; to_md5; datarep:dns_md5, >, 200, load dns_
↪md5.rep, type md5; sid:1;)
alert dns any any -> any any (dns.query; to_sha256; datarep:dns_sha256, >, 200, load␣
↪dns_sha256.rep, type sha256; sid:2;)
alert dns any any -> any any (dns.query; datarep:dns_string, >, 200, load dns_string.
↪rep, type string; sid:3;)
```

In these examples the DNS query string is checked against three different reputation lists. A MD5 list, a SHA256 list, and a raw string (buffer) list. The rules will only match if the data is in the list and the reputation value is higher than 200.

### 6.30.3 Unix Socket

#### dataset-add

Unix Socket command to add data to a set. On success, the addition becomes active instantly.

Syntax:

```
dataset-add <set name> <set type> <data>
```

**set name** Name of an already defined dataset

**type** Data type: string, md5, sha256

**data** Data to add in serialized form (base64 for string, hex notation for md5/sha256)

Example adding 'google.com' to set 'myset':

```
dataset-add myset string Z29vZ2xlLmNvbQ==
```

### 6.30.4 File formats

Datasets use a simple CSV format where data is per line in the file.

#### data types

**string** in the file as base64 encoded string

**md5** in the file as hex encoded string

**sha256** in the file as hex encoded string

#### dataset

Datasets have a simple structure, where there is one piece of data per line in the file.

Syntax:

```
<data>
```

e.g. for ua-seen with type string:

```
TW96aWxsYS80LjAgKGNvbXBhdGlibGU7ICk=
```

which when piped to `base64 -d` reveals its value:

```
Mozilla/4.0 (compatible; )
```

#### datarep

The datarep format follows the dataset, expect that there are 1 more CSV field:

Syntax:

```
<data>,<value>
```

## 6.31 Lua Scripting

Syntax:

```
lua:[!]<scriptfilename>;
```

The script filename will be appended to your default rules location.

The script has 2 parts, an init function and a match function. First, the init.

### 6.31.1 Init function

```
function init (args)
    local needs = {}
    needs["http.request_line"] = tostring(true)
    return needs
end
```

The init function registers the buffer(s) that need inspection. Currently the following are available:

- packet – entire packet, including headers
- payload – packet payload (not stream)
- buffer – the current sticky buffer
- http.uri
- http.uri.raw
- http.request_line
- http.request_headers
- http.request_headers.raw
- http.request_cookie
- http.request_user_agent
- http.request_body
- http.response_headers
- http.response_headers.raw
- http.response_body
- http.response_cookie

All the HTTP buffers have a limitation: only one can be inspected by a script at a time.

### 6.31.2 Match function

```
function match(args)
    a = tostring(args["http.request_line"])
    if #a > 0 then
        if a:find("^POST%s+/.*%.php%s+HTTP/1.0$") then
            return 1
        end
    end

    return 0
end
```

The script can return 1 or 0. It should return 1 if the condition(s) it checks for match, 0 if not.

Entire script:

```
function init (args)
    local needs = {}
    needs["http.request_line"] = tostring(true)
    return needs
end

function match(args)
    a = tostring(args["http.request_line"])
    if #a > 0 then
        if a:find("^POST%s+/.*%.php%s+HTTP/1.0$") then
            return 1
        end
    end

    return 0
end

return 0
```

## 6.32 Differences From Snort

This document is intended to highlight the major differences between Suricata and Snort that apply to rules and rule writing.

Where not specified, the statements below apply to Suricata. In general, references to Snort refer to the version 2.9 branch.

### 6.32.1 Automatic Protocol Detection

- Suricata does automatic protocol detection of the following application layer protocols:
  - dcerpc
  - dnp3
  - dns
  - http
  - imap (detection only by default; no parsing)
  - ftp

  - modbus (disabled by default; minimalist probe parser; can lead to false positives)
  - smb
  - smb2 (disabled internally inside the engine)
  - smtp
  - ssh
  - tls (SSLv2, SSLv3, TLSv1, TLSv1.1 and TLSv1.2)

- In Suricata, protocol detection is port agnostic (in most cases). In Snort, in order for the `http_inspect` and other preprocessors to be applied to traffic, it has to be over a configured port.
  - Some configurations for app-layer in the Suricata yaml can/do by default specify specific destination ports (e.g. DNS)
  - **You can look on 'any' port without worrying about the performance impact that you would have to be concerned about with Snort.**
- If the traffic is detected as HTTP by Suricata, the `http_*` buffers are populated and can be used, regardless of port(s) specified in the rule.
- You don't have to check for the http protocol (i.e. `alert http ...`) to use the `http_*` buffers although it is recommended.
- If you are trying to detect legitimate (supported) application layer protocol traffic and don't want to look on specific port(s), the rule should be written as `alert <protocol> ...` with `any` in place of the usual protocol port(s). For example, when you want to detect HTTP traffic and don't want to limit detection to a particular port or list of ports, the rules should be written as `alert http ...` with `any` in place of `$HTTP_PORTS`.
  - You can also use `app-layer-protocol:<protocol>;` inside the rule instead.

  So, instead of this Snort rule:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS ...
```

  Do this for Suricata:

```
alert http $HOME_NET -> $EXTERNAL_NET any ...
```

  Or:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (app-layer-protocol:http; ...
```

### 6.32.2 `urilen` Keyword

- Ranges given in the `urilen` keyword are inclusive for Snort but not inclusive for Suricata.

  Example: `urilen:2<>10`
  - Snort interprets this as, "the URI length must be **greater than or equal to** 2, and **less than or equal to** 10".
  - Suricata interprets this as "the URI length must be **greater than** 2 and **less than** 10".
  - There is a request to have Suricata behave like Snort in future versions – https://redmine.openinfosecfoundation.org/issues/1416
    - Currently on hold
- By default, with *Suricata*, `urilen` applies to the **normalized** buffer

  - Use `,raw` for raw buffer
  - e.g. `urilen:>20,raw;`
- By default, with *Snort*, `urilen` applies to the **raw** buffer
  - Use `,norm` for normalized buffer
  - e.g. `urilen:>20,norm;`

### 6.32.3 `http_uri` Buffer

- In Snort, the `http_uri` buffer normalizes '+' characters (0x2B) to spaces (0x20).
  - Suricata can do this as well but you have to explicitly set `query-plusspace-decode: yes` in the `libhtp` section of Suricata's yaml file.
- https://redmine.openinfosecfoundation.org/issues/1035
- https://github.com/inliniac/suricata/pull/620

### 6.32.4 `http_header` Buffer

- In Snort, the `http_header` buffer includes the CRLF CRLF (0x0D 0x0A 0x0D 0x0A) that separates the end of the last HTTP header from the beginning of the HTTP body. Suricata includes a CRLF after the last header in the `http_header` buffer but not an extra one like Snort does. If you want to match the end of the buffer, use either the `http_raw_header` buffer, a relative `isdataat` (e.g. `isdataat:!1,relative`) or a PCRE (although PCRE will be worse on performance).
- Suricata *will* include CRLF CRLF at the end of the `http_raw_header` buffer like Snort does.
- Snort will include a *leading* CRLF in the `http_header` buffer of *server responses* (but not client requests). Suricata does not have the leading CRLF in the `http_header` buffer of the server response or client request.
- In the `http_header` buffer, Suricata will normalize HTTP header lines such that there is a single space (0x20) after the colon (':') that separates the header name from the header value; this single space replaces zero or more whitespace characters (including tabs) that may be present in the raw HTTP header line immediately after the colon. If the extra whitespace (or lack thereof) is important for matching, use the `http_raw_header` buffer instead of the `http_header` buffer.
- Snort will also normalize superfluous whitespace between the header name and header value like Suricata does but only if there is at least one space character (0x20 only so not 0x90) immediately after the colon. This means that, unlike Suricata, if there is no space (or if there is a tab) immediately after the colon before the header value, the content of the header line will remain unchanged in the `http_header` buffer.
- When there are duplicate HTTP headers (referring to the header name only, not the value), the normalized buffer (`http_header`) will concatenate the values in the order seen (from top to bottom), with a comma and space (", ") between each of them. If this hinders detection, use the `http_raw_header` buffer instead.

  Example request:

```
GET /test.html HTTP/1.1
Content-Length: 44
Accept: */*
Content-Length: 55
```

  The Content-Length header line becomes this in the `http_header` buffer:

```
Content-Length: 44, 55
```

- The HTTP 'Cookie' and 'Set-Cookie' headers are **NOT** included in the `http_header` buffer; instead they are extracted and put into their own buffer – `http_cookie`. See the *http_cookie Buffer* section.
- The HTTP 'Cookie' and 'Set-Cookie' headers **ARE** included in the `http_raw_header` buffer so if you are trying to match on something like particular header ordering involving (or not involving) the HTTP Cookie headers, use the `http_raw_header` buffer.
- If 'enable_cookie' is set for Snort, the HTTP Cookie header names and trailing CRLF (i.e. "Cookie: \r\n" and "Set-Cooke \r\n") are kept in the `http_header` buffer. This is not the case for Suricata which removes the entire "Cookie" or "Set-Cookie" line from the `http_header` buffer.
- Other HTTP headers that have their own buffer (`http_user_agent`, `http_host`) are not removed from the `http_header` buffer like the Cookie headers are.
- When inspecting server responses and `file_data` is used, content matches in `http_*` buffers should come before `file_data` unless you use `pkt_data` to reset the cursor before matching in `http_*` buffers. Snort will not complain if you use `http_*` buffers after `file_data` is set.

### 6.32.5 `http_cookie` Buffer

- The `http_cookie` buffer will NOT include the header name, colon, or leading whitespace. i.e. it will not include "Cookie: " or "Set-Cookie: ".
- The `http_cookie` buffer does not include a CRLF (0x0D 0x0A) at the end. If you want to match the end of the buffer, use a relative `isdataat` or a PCRE (although PCRE will be worse on performance).
- There is no `http_raw_cookie` buffer in Suricata. Use `http_raw_header` instead.
- You do not have to configure anything special to use the 'http_cookie' buffer in Suricata. This is different from Snort where you have to set `enable_cookie` in the `http_inspect_server` preprocessor config in order to have the `http_cookie` buffer treated separate from the `http_header` buffer.
- If Snort has 'enable_cookie' set and multiple "Cookie" or "Set-Cookie" headers are seen, it will concatenate them together (with no separator between them) in the order seen from top to bottom.
- If a request contains multiple "Cookie" or "Set-Cookie" headers, the values will be concatenated in the Suricata `http_cookie` buffer, in the order seen from top to bottom, with a comma and space (", ") between each of them.

  Example request:

```
GET /test.html HTTP/1.1
Cookie: monster
Accept: */*
Cookie: elmo
```

  Suricata `http_cookie` buffer contents:

```
monster, elmo
```

  Snort `http_cookie` buffer contents:

```
monsterelmo
```

- Corresponding PCRE modifier: `C` (same as Snort)

### 6.32.6 New HTTP keywords

Suricata supports several HTTP keywords that Snort doesn't have.

Examples are `http_user_agent`, `http_host` and `http_content_type`.

See *HTTP Keywords* for all HTTP keywords.

### 6.32.7 `byte_extract` Keyword

- Suricata supports `byte_extract` from `http_*` buffers, including `http_header` which does not always work as expected in Snort.
- In Suricata, variables extracted using `byte_extract` must be used in the same buffer, otherwise they will have the value "0" (zero). Snort does allow cross-buffer byte extraction and usage.
- Be sure to always positively and negatively test Suricata rules that use `byte_extract` and `byte_test` to verify that they work as expected.

### 6.32.8 `isdataat` Keyword

- The `rawbytes` keyword is supported in the Suricata syntax but doesn't actually do anything.
- Absolute `isdataat` checks will succeed if the offset used is **less than** the size of the inspection buffer. This is true for Suricata and Snort.
- For *relative* `isdataat` checks, there is a **1 byte difference** in the way Snort and Suricata do the comparisons.
  - Suricata will succeed if the relative offset is **less than or equal to** the size of the inspection buffer. This is different from absolute `isdataat` checks.
  - Snort will succeed if the relative offset is **less than** the size of the inspection buffer, just like absolute `isdataat` checks.
  - Example - to check that there is no data in the inspection buffer after the last content match:
    - Snort: `isdataat:!0,relative;`
    - Suricata: `isdataat:!1,relative;`
- With Snort, the "inspection buffer" used when checking an `isdataat` keyword is generally the packet/segment with some exceptions:
  - With PAF enabled the PDU is examined instead of the packet/segment. When `file_data` or `base64_data` has been set, it is those buffers (unless `rawbytes` is set).
  - With some preprocessors - modbus, gtp, sip, dce2, and dnp3 - the buffer can be particular portions of those protocols (unless `rawbytes` is set).
  - With some preprocessors - rpc_decode, ftp_telnet, smtp, and dnp3 - the buffer can be particular *decoded* portions of those protocols (unless `rawbytes` is set).
- With Suricata, the "inspection buffer" used when checking an absolute `isdataat` keyword is the packet/segment if looking at a packet (e.g. `alert tcp-pkt...`) or the reassembled stream segments.
- In Suricata, a *relative* `isdataat` keyword **will apply to the buffer of the previous content match**. So if the previous content match is a `http_*` buffer, the relative `isdataat` applies to that buffer, starting from the end of the previous content match in that buffer. *Snort does not behave like this!*
- For example, this Suricata rule looks for the string ".exe" at the end of the URI; to do the same thing in the normalized URI buffer in Snort you would have to use a PCRE – `pcre:"/\x2Eexe$/U";`

```
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:".EXE File Download Request";␣
↪flow:established,to_server; content:"GET"; http_method; content:".exe"; http_
↪uri; isdataat:!1,relative; priority:3; sid:18332111;)
```

- If you are unclear about behavior in a particular instance, you are encouraged to positively and negatively test your rules that use an `isdataat` keyword.

### 6.32.9 Relative PCRE

- You can do relative PCRE matches in normalized/special buffers with Suricata. Example:

```
content:".php?sign="; http_uri; pcre:"/^[a-zA-Z0-9]{8}$/UR";
```

- With Snort you can't combine the "relative" PCRE option ('R') with other buffer options like normalized URI ('U') – you get a syntax error.

### 6.32.10 `tls*` Keywords

In addition to TLS protocol identification, Suricata supports the storing of certificates to disk, verifying the validity dates on certificates, matching against the calculated SHA1 fingerprint of certificates, and matching on certain TLS/SSL certificate fields including the following:

- Negotiated TLS/SSL version.
- Certificate Subject field.
- Certificate Issuer field.
- Certificate SNI Field

For details see *SSL/TLS Keywords*.

### 6.32.11 `dns_query` Keyword

- Sets the detection pointer to the DNS query.
- Works like `file_data` does ("sticky buffer") but for a DNS request query.
- Use `pkt_data` to reset the detection pointer to the beginning of the packet payload.
- See *DNS Keywords* for details.

### 6.32.12 IP Reputation and `iprep` Keyword

- Snort has the "reputation" preprocessor that can be used to define whitelist and blacklist files of IPs which are used generate GID 136 alerts as well as block/drop/pass traffic from listed IPs depending on how it is configured.
- Suricata also has the concept of files with IPs in them but provides the ability to assign them:
  - Categories
  - Reputation score
- Suricata rules can leverage these IP lists with the `iprep` keyword that can be configured to match on:
  - Direction

  - Category
  - Value (reputation score)

- *Reputation*
- *IP Reputation Config*
- *IP Reputation Keyword*
- *IP Reputation Format*
- https://blog.inliniac.net/2012/11/21/ip-reputation-in-suricata/

### 6.32.13 Flowbits

- Suricata fully supports the setting and checking of flowbits (including the same flowbit) on the same packet/stream. Snort does not always allow for this.
- In Suricata, `flowbits:isset` is checked after the fast pattern match but before other `content` matches. In Snort, `flowbits:isset` is checked in the order it appears in the rule, from left to right.
- If there is a chain of flowbits where multiple rules set flowbits and they are dependent on each other, then the order of the rules or the `sid` values can make a difference in the rules being evaluated in the proper order and generating alerts as expected. See bug 1399 - https://redmine.openinfosecfoundation.org/issues/1399.
- *Flow Keywords*

### 6.32.14 flowbits:noalert;

A common pattern in existing rules is to use `flowbits:noalert;` to make sure a rule doesn't generate an alert if it matches.

Suricata allows using just `noalert;` as well. Both have an identical meaning in Suricata.

### 6.32.15 Negated Content Match Special Case

- For Snort, a *negated* content match where the starting point for searching is at or beyond the end of the inspection buffer will never return true.
  - For negated matches, you want it to return true if the content is not found.
  - This is believed to be a Snort bug rather than an engine difference but it was reported to Sourcefire and acknowledged many years ago indicating that perhaps it is by design.
  - This is not the case for Suricata which behaves as expected.

    Example HTTP request:

    ```
    POST /test.php HTTP/1.1
    Content-Length: 9

    user=suri
    ```

    This rule snippet will never return true in Snort but will in Suricata:

    ```
    content:!"snort"; offset:10; http_client_body;
    ```

### 6.32.16 File Extraction

- Suricata has the ability to match on files from FTP, HTTP and SMTP streams and log them to disk.
- Snort has the "file" preprocessor that can do something similar but it is experimental, development of it has been stagnant for years, and it is not something that should be used in a production environment.
- Files can be matched on using a number of keywords including:
  - `filename`
  - `fileext`
  - `filemagic`
  - `filesize`
  - `filemd5`
  - `filesha1`
  - `filesha256`
  - `filesize`
  - See *File Keywords* for a full list.
- The `filestore` keyword tells Suricata to save the file to disk.
- Extracted files are logged to disk with meta data that includes things like timestamp, src/dst IP, protocol, src/dst port, HTTP URI, HTTP Host, HTTP Referer, filename, file magic, md5sum, size, etc.
- There are a number of configuration options and considerations (such as stream reassembly depth and libhtp body-limit) that should be understood if you want fully utilize file extraction in Suricata.
- *File Keywords*
- *File Extraction*
- https://blog.inliniac.net/2011/11/29/file-extraction-in-suricata/
- https://blog.inliniac.net/2014/11/11/smtp-file-extraction-in-suricata/

### 6.32.17 Lua Scripting

- Suricata has the `lua` (or `luajit`) keyword which allows for a rule to reference a Lua script that can access the packet, payload, HTTP buffers, etc.
- Provides powerful flexibility and capabilities that Snort does not have.
- *Lua Scripting*

### 6.32.18 Fast Pattern

- Snort's fast pattern matcher is always case insensitive; Suricata's is case sensitive unless 'nocase' is set on the content match used by the fast pattern matcher.
- Snort will truncate fast pattern matches based on the `max-pattern-len` config (default no limit) unless `fast_pattern:only` is used in the rule. Suricata does not do any automatic fast pattern truncation cannot be configured to do so.
- Just like in Snort, in Suricata you can specify a substring of the content string to be use as the fast pattern match. e.g. `fast_pattern:5,20;`

- In Snort, leading NULL bytes (0x00) will be removed from content matches when determining/using the longest content match unless `fast_pattern` is explicitly set. Suricata does not truncate anything, including NULL bytes.
- Snort does not allow for all `http_*` buffers to be used for the fast pattern match (e.g. `http_raw_*`, `http_method`, `http_cookie`, etc.). Suricata lets you use any 'http_*' buffer you want for the fast pattern match, including `http_raw_*' and ``http_cookie` buffers.
- Suricata supports the `fast_pattern:only` syntax but technically it is not really implemented; the `only` is silently ignored when encountered in a rule. It is still recommended that you use `fast_pattern:only` where appropriate in case this gets implemented in the future and/or if the rule will be used by Snort as well.
- With Snort, unless `fast_pattern` is explicitly set, content matches in normalized HTTP Inspect buffers (e.g. http content modifiers such `http_uri`, `http_header`, etc.) take precedence over non-HTTP Inspect content matches, even if they are shorter. Suricata does the same thing and gives a higher 'priority' (precedence) to `http_*` buffers (except for `http_method`, `http_stat_code`, and `http_stat_msg`).
- See *Suricata Fast Pattern Determination Explained* for full details on how Suricata automatically determines which content to use as the fast pattern match.
- When in doubt about what is going to be use as the fast pattern match by Suricata, set `fast_pattern` explicitly in the rule and/or run Suricata with the `--engine-analysis` switch and view the generated file (`rules_fast_pattern.txt`).
- Like Snort, the fast pattern match is checked before `flowbits` in Suricata.
- Using Hyperscan as the MPM matcher (`mpm-algo` setting) for Suricata can greatly improve performance, especially when it comes to fast pattern matching. Hyperscan will also take in to account depth and offset when doing fast pattern matching, something the other algorithims and Snort do not do.
- *fast_pattern*

### 6.32.19 Don't Cross The Streams

Suricata will examine network traffic as individual packets and, in the case of TCP, as part of a (reassembled) stream. However, there are certain rule keywords that only apply to packets only (`dsize`, `flags`, `ttl`) and certain ones that only apply to streams only (`http_*`) and you can't mix packet and stream keywords. Rules that use packet keywords will inspect individual packets only and rules that use stream keywords will inspect streams only. Snort is a little more forgiving when you mix these – for example, in Snort you can use `dsize` (a packet keyword) with `http_*` (stream keywords) and Snort will allow it although, because of `dsize`, it will only apply detection to individual packets (unless PAF is enabled then it will apply it to the PDU).

If `dsize` is in a rule that also looks for a stream-based application layer protocol (e.g. `http`), Suricata will not match on the *first application layer packet* since `dsize` make Suricata evaluate the packet and protocol detection doesn't happen until after the protocol is checked for that packet; *subsequent* packets in that flow should have the application protocol set appropriately and will match rules using `dsize` and a stream-based application layer protocol.

If you need to check sizes on a stream in a rule that uses a stream keyword, or in a rule looking for a stream-based application layer protocol, consider using the `stream_size` keyword and/or `isdataat`.

Suricata also supports these protocol values being used in rules and Snort does not:

- `tcp-pkt` – example:
  - `alert tcp-pkt ...`
  - This tells Suricata to only apply the rule to TCP packets and not the (reassembled) stream.
- `tcp-stream` – example:
  - `alert tcp-stream ...`

– This tells Suricata to inspect the (reassembled) TCP stream only.

### 6.32.20 Alerts

- In Snort, the number of alerts generated for a packet/stream can be limited by the `event_queue` configuration.
- Suricata has an internal hard-coded limit of 15 alerts per packet/stream (and this cannot be configured); all rules that match on the traffic being analyzed will fire up to that limit.
- Sometimes Suricata will generate what appears to be two alerts for the same TCP packet. This happens when Suricata evaluates the packet by itself and as part of a (reassembled) stream.

### 6.32.21 Buffer Reference Chart

| Buffer | Snort 2.9.x Support? | Suricata Support? | PCRE flag | Can be used as Fast Pattern? | Suricata Fast Pattern Priority (lower number is higher priority) |
|---|---|---|---|---|---|
| content (no modifier) | YES | YES | <none> | YES | 3 |
| http_method | YES | YES | M | Suricata only | 3 |
| http_stat_code | YES | YES | S | Suricata only | 3 |
| http_stat_msg | YES | YES | Y | Suricata only | 3 |
| uricontent | YES but deprecated, use http_uri instead | YES but deprecated, use http_uri instead | U | YES | 2 |
| http_uri | YES | YES | U | YES | 2 |
| http_raw_uri | YES | YES | I | Suricata only | 2 |
| http_header | YES | YES | H | YES | 2 |
| http_raw_header | YES | YES | D | Suricata only | 2 |
| http_cookie | YES | YES | C | Suricata only | 2 |
| http_raw_cookie | YES | NO (use http_raw_header instead) | K | NO | n/a |
| http_host | NO | YES | W | Suricata only | 2 |
| http_raw_host | NO | YES | Z | Suricata only | 2 |
| http_client_body | YES | YES | P | YES | 2 |
| http_server_body | NO | YES | Q | Suricata only | 2 |
| http_user_agent | NO | YES | V | Suricata only | 2 |
| dns_query | NO | YES | n/a* | Suricata only | 2 |
| tls_sni | NO | YES | n/a* | Suricata only | 2 |
| tls_cert_issuer | NO | YES | n/a* | Suricata only | 2 |
| tls_cert_subject | NO | YES | n/a* | Suricata only | 2 |
| file_data | YES | YES | n/a* | YES | 2 |

* Sticky buffer

CHAPTER 7

# Rule Management

## 7.1 Rule Management with Suricata-Update

While it is possible to download and install rules manually, it is recommended to use a management tool for this. Suricata-Update is the official way to update and manage rules for Suricata.

Suricata-update is bundled with Suricata and is normally installed with it. For instructions on installing manually, see http://suricata-update.readthedocs.io/en/latest/quickstart.html#install-suricata-update

**Note:** `suricata-update` is bundled with Suricata version 4.1 and later. It can be used with older versions as well. It will have to be installed separately in that case.

To download the Emerging Threats Open ruleset, it is enough to simply run:

```
sudo suricata-update
```

This will download the ruleset into /var/lib/suricata/rules/

Suricata's configuration will have to be updated to have a rules config like this:

```
default-rule-path: /var/lib/suricata/rules
rule-files:
  - suricata.rules
```

Now (re)start Suricata.

### 7.1.1 Updating your rules

To update the rules, simply run

```
sudo suricata-update
```

It is recommended to update your rules frequently.

### 7.1.2 Using other rulesets

Suricata-Update is capable of making other rulesets accessible as well.

To see what is available, fetch the master index from the OISF hosts:

```
sudo suricata-update update-sources
```

Then have a look at what is available:

```
sudo suricata-update list-sources
```

This will give a result similar to

```
Name: oisf/trafficid
  Vendor: OISF
  Summary: Suricata Traffic ID ruleset
  License: MIT
Name: ptresearch/attackdetection
  Vendor: Positive Technologies
  Summary: Positive Technologies Attack Detection Team ruleset
  License: Custom
Name: sslbl/ssl-fp-blacklist
  Vendor: Abuse.ch
  Summary: Abuse.ch SSL Blacklist
  License: Non-Commercial
Name: et/open
  Vendor: Proofpoint
  Summary: Emerging Threats Open Ruleset
  License: MIT
Name: scwx/security
  Vendor: Secureworks
  Summary: Secureworks suricata-security ruleset.
  License: Commercial
  Parameters: secret-code
  Subscription: https://www.secureworks.com/contact/ (Please reference CTU Countermeasures)
Name: scwx/malware
  Vendor: Secureworks
  Summary: Secureworks suricata-malware ruleset.
  License: Commercial
  Parameters: secret-code
  Subscription: https://www.secureworks.com/contact/ (Please reference CTU Countermeasures)
Name: et/pro
  Vendor: Proofpoint
  Summary: Emerging Threats Pro Ruleset
  License: Commercial
  Replaces: et/open
  Parameters: secret-code
  Subscription: https://www.proofpoint.com/us/threat-insight/et-pro-ruleset
```

Each of the rulesets has a name that has a 'vendor' prefix, followed by a set name. For example, OISF's traffic id ruleset is called 'oisf/trafficid'.

To enable 'oisf/trafficid', enter:

```
sudo suricata-update enable-source oisf/trafficid
sudo suricata-update
```

Now restart Suricata again and the rules from the OISF TrafficID ruleset are loaded.

To see which rulesets are currently active, use "list-enabled-sources".

### 7.1.3 Controlling which rules are used

By default suricata-update will merge all rules into a single file "/var/lib/suricata/rules/suricata.rules".

To enable rules that are disabled by default, use '/etc/suricata/enable.conf'

```
2019401                    # enable this sid
group:emerging-icmp.rules  # enable this rulefile
re:trojan                  # enable all rules with this string
```

Similarly, to disable rules use /etc/suricata/disable.conf:

```
2019401                    # enable this sid
group:emerging-info.rules  # disable this rulefile
re:heartbleed              # disable all rules with this string
```

After updating these files, rerun `suricata-update` again:

```
sudo suricata-update
```

Finally restart Suricata.

### 7.1.4 Further reading

See https://suricata-update.readthedocs.io/en/latest/

## 7.2 Adding Your Own Rules

If you would like to create a rule yourself and use it with Suricata, this guide might be helpful.

Start creating a file for your rule. Type for example the following in your console:

```
sudo nano local.rules
```

Write your rule, see *Rules Format* and save it.

Open yaml

```
sudo nano /etc/suricata/suricata.yaml
```

and make sure your local.rules file is added to the list of rules.

Now, run Suricata and see if your rule is being loaded.

```
suricata -c /etc/suricata/suricata.yaml -i wlan0
```

If your rule failed to load, check if you have made a mistake anywhere in the rule. Mind the details; look for mistakes in special characters, spaces, capital characters etc.

Next, check if your log-files are enabled in suricata.yaml.

If you had to correct your rule and/or modify yaml, you have to restart Suricata.

If you see your rule is successfully loaded, you can double check your rule by doing something that should trigger it.

Enter:

```
tail -f /var/log/suricata/fast.log
```

If you would make a rule like this:

```
alert http any any -> any any (msg:"Do not read gossip during work";
content:"Scarlett"; nocase; classtype:policy-violation; sid:1; rev:1;)
```

Your alert should look like this:

```
09/15/2011-16:50:27.725288  [**] [1:1:1] Do not read gossip during work [**]
[Classification: Potential Corporate Privacy Violation] [Priority: 1] {TCP} 192.168.0.
↪32:55604 -> 68.67.185.210:80
```

## 7.3 Rule Reloads

Suricata can be told to reloads it's rules without restarting.

This works by sending Suricata a signal or by using the unix socket. When Suricata is told to reload the rules these are the basic steps it takes:

- Load new config
- Load new rules
- Construct new detection engine
- Swap old and new detection engines
- Make sure all threads are updated
- Free old detection engine

Suricata will continue to process packets normally during this process. Keep in mind though, that the system should have enough memory for both detection engines.

Signal:

```
kill -USR2 $(pidof suricata)
```

Unix socket has two method for rules reload.

Blocking reload

```
suricatasc -c reload-rules
```

Non blocking reload

```
suricatasc -c ruleset-reload-nonblocking
```

It is also possible to get information about the last reload via dedicated commands. See *Commands in standard running mode* for more information.

# CHAPTER 8

## Making sense out of Alerts

When alert happens it's important to figure out what it means. Is it serious? Relevant? A false positive?

To find out more about the rule that fired, it's always a good idea to look at the actual rule.

The first thing to look at in a rule is the description that follows the "msg" keyword. Lets consider an example:

```
msg:"ET SCAN sipscan probe";
```

The "ET" indicates the rule came from the Emerging Threats project. "SCAN" indicates the purpose of the rule is to match on some form of scanning. Following that a more or less detailed description is given.

Most rules contain some pointers to more information in the form of the "reference" keyword.

Consider the following example rule:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS \
  (msg:"ET CURRENT_EVENTS Adobe 0day Shovelware"; \
  flow:established,to_server; content:"GET "; nocase; depth:4; \
  content:!"|0d 0a|Referer\:"; nocase; \
  uricontent:"/ppp/listdir.php?dir="; \
  pcre:"/\/[a-z]{2}\/[a-z]{4}01\/ppp\/listdir\.php\?dir=/U"; \
  classtype:trojan-activity; \
  reference:url,isc.sans.org/diary.html?storyid=7747; \
  reference:url,doc.emergingthreats.net/2010496; \
  reference:url,www.emergingthreats.net/cgi-bin/cvsweb.cgi/sigs/CURRENT_EVENTS/
↪CURRENT_Adobe; \
  sid:2010496; rev:2;)
```

In this rule the reference keyword indicates 3 url's to visit for more information:

```
isc.sans.org/diary.html?storyid=7747
doc.emergingthreats.net/2010496
www.emergingthreats.net/cgi-bin/cvsweb.cgi/sigs/CURRENT_EVENTS/CURRENT_Adobe
```

Some rules contain a reference like: "reference:cve,2009-3958;" should allow you to find info about the specific CVE using your favorite search engine.

It's not always straight forward and sometimes not all of that information is available publicly. Usually asking about it on the signature support channel helps a lot then.

In *Rule Management with Suricata-Update* more information on the rule sources and their documentation and support methods can be found.

In many cases, looking at just the alert and the packet that triggered it won't be enough to be conclusive. When using the default Eve settings a lot of metadata will be added to the alert.

For example, if a rule fired that indicates your web application is attacked, looking at the metadata might reveal that the web application replied with 404 not found. This will usually mean the attack failed. Usually, not always.

Not every protocol leads to metadata generation, so when running an IDS engine like Suricata, it's often recommended to combine it with full packet capture. Using tools like Evebox, Sguil or Snorby, the full TCP session or UDP flow can be inspected.

Obviously there is a lot more to Incidence Response, but this should get you started.

CHAPTER 9

# Performance

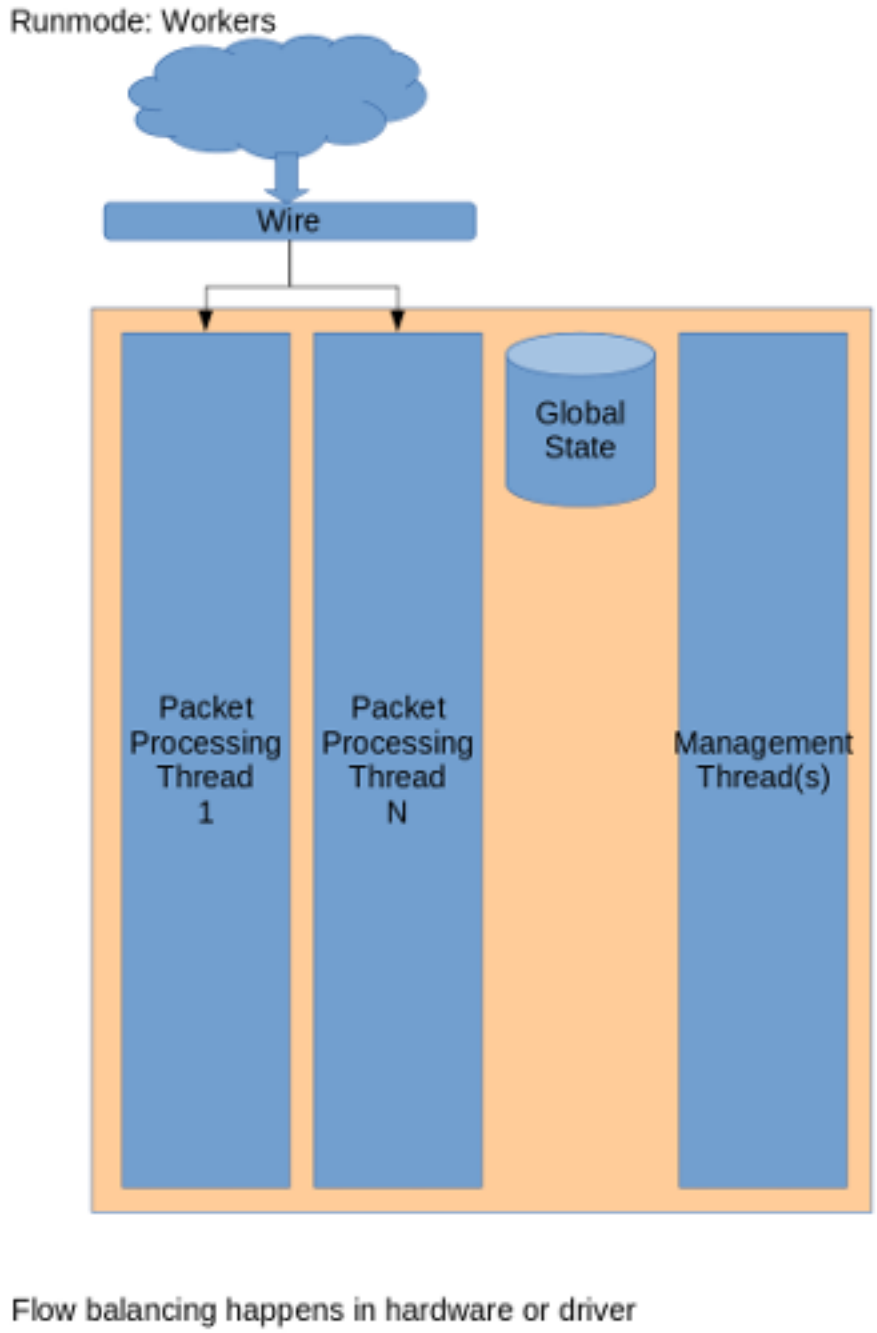
## 9.1 Runmodes

Suricata consists of several 'building blocks' called threads, thread-modules and queues. A thread is like a process that runs on a computer. Suricata is multi-threaded, so multiple threads are active at once. A thread-module is a part of a functionality. One module is for example for decoding a packet, another is the detect-module and another one the output-module. A packet can be processed by more than one thread. The packet will be passed on to the next thread through a queue. Packets will be processed by one thread at a time, but there can be multiple packets being processed at a time by the engine. (see *Max-pending-packets*) A thread can have one or more thread-modules. If they have more modules, they can only be active on a a time. The way threads, modules and queues are arranged together is called the Runmode.

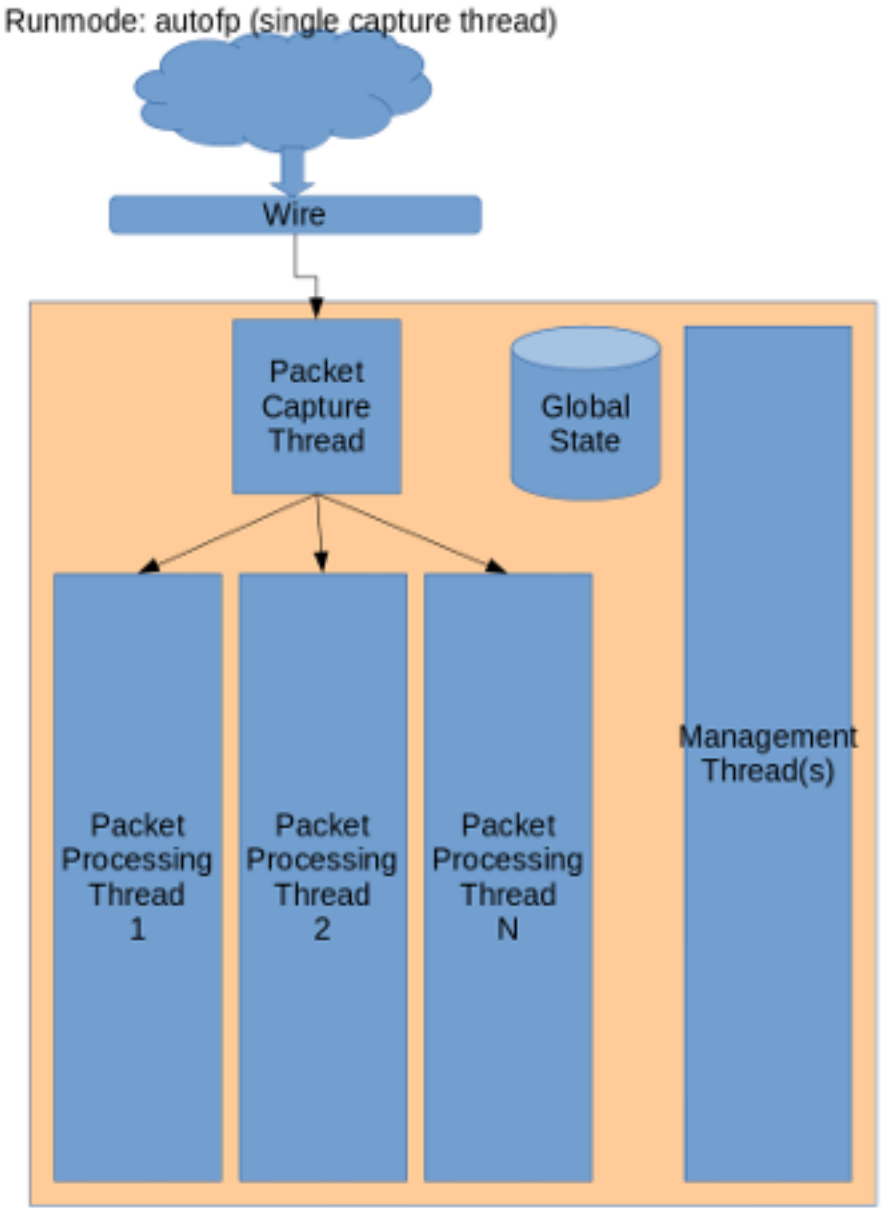
### 9.1.1 Different runmodes

You can choose a runmode out of several predefined runmodes. The command line option –list-runmodes shows all available runmodes. All runmodes have a name: single, workers, autofp.

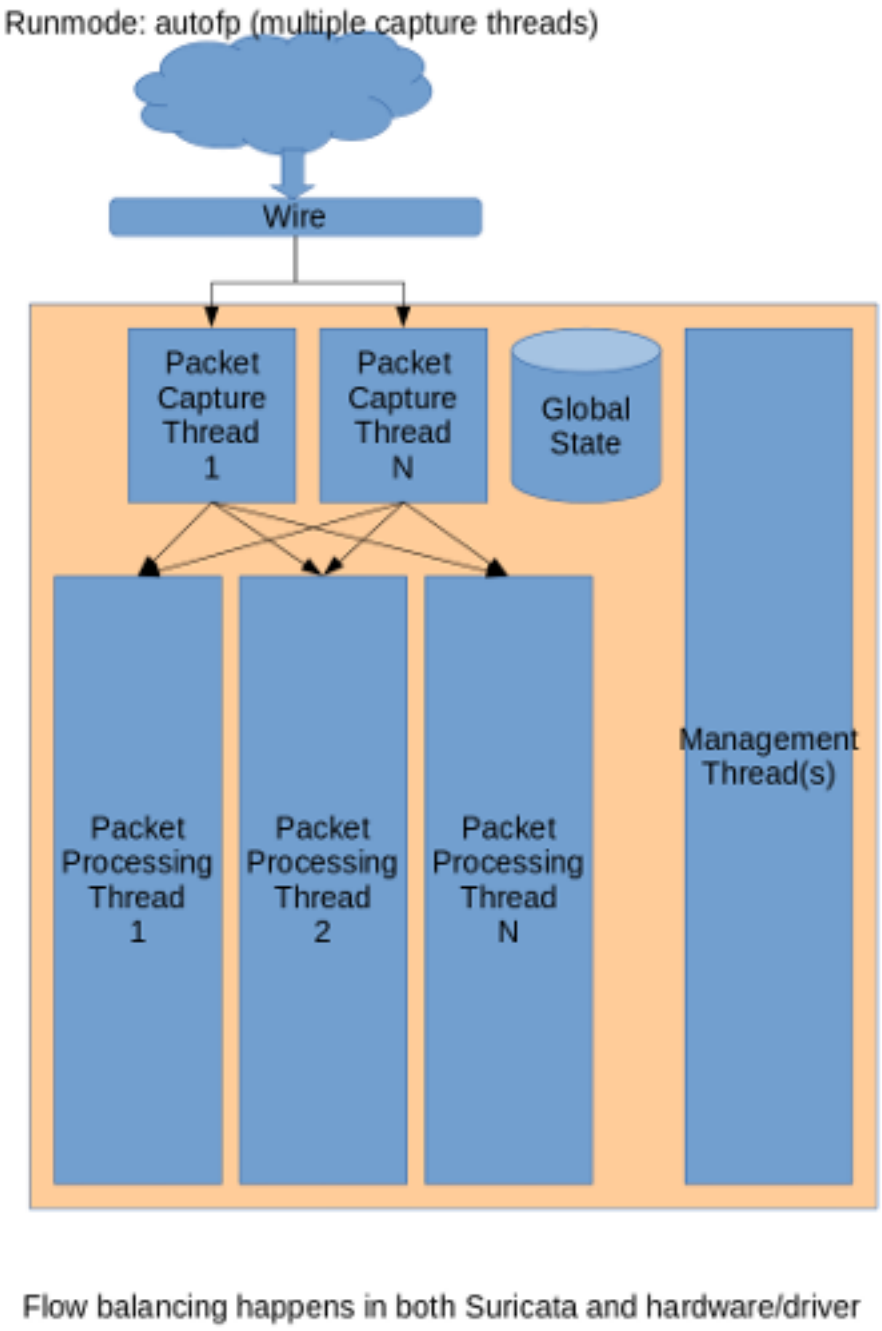
Generally, the `workers` runmode performs the best. In this mode the NIC/driver makes sure packets are properly balanced over Suricata's processing threads. Each packet processing thread then contains the full packet pipeline.




For processing PCAP files, or in case of certain IPS setups (like NFQ), `autofp` is used. Here there are one or more capture threads, that capture the packet and do the packet decoding, after which it is passed on to the `flow worker` threads.

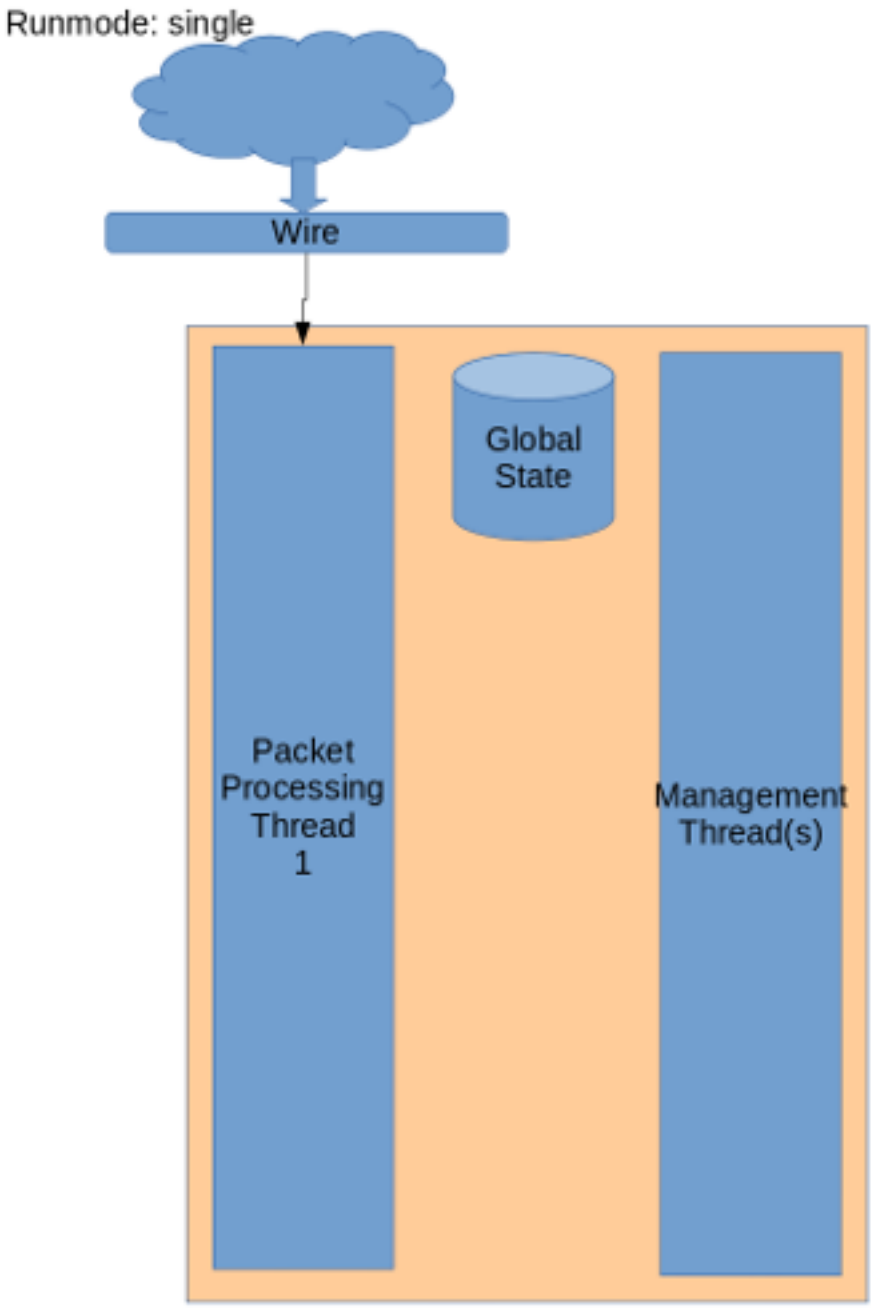



Flow balancing happens inside Suricata




Flow balancing happens in both Suricata and hardware/driver

Finally, the `single` runmode is the same as the `workers` mode, however there is only a single packet processing thread. This useful during development.




For more information about the command line options concerning the runmode, see *Command Line Options*.

## 9.2 Packet Capture

### 9.2.1 Load balancing

To get the best performance, Suricata will need to run in 'workers' mode. This effectively means that there are multiple threads, each running a full packet pipeline and each receiving packets from the capture method. This means that we rely on the capture method to distribute the packets over the various threads. One critical aspect of this is that Suricata

needs to get both sides of a flow in the same thread, in the correct order.

The AF_PACKET and PF_RING capture methods both have options to select the 'cluster-type'. These default to 'cluster_flow' which instructs the capture method to hash by flow (5 tuple). This hash is symmetric. Netmap does not have a cluster_flow mode built-in. It can be added separately by using the "'lb' tool":https://github.com/luigirizzo/netmap/tree/master/apps/lb

> **WARNING** Recent AF_PACKET changes have "broken":https://redmine.openinfosecfoundation.org/issues/1777 this symmetry. Work is under way to "address this":https://redmine.openinfosecfoundation.org/issues/1777#note-7, but for now stay on kernel <=4.2 or update to 4.4.16+, 4.6.5+ or 4.7+.

On multi-queue NICs, which is almost any modern NIC, RSS settings need to be considered.

### 9.2.2 RSS

Receive Side Scaling is a technique used by network cards to distribute incoming traffic over various queues on the NIC. This is meant to improve performance but it is important to realize that it was designed for normal traffic, not for the IDS packet capture scenario. RSS using a hash algorithm to distribute the incoming traffic over the various queues. This hash is normally *not* symmetrical. This means that when receiving both sides of a flow, each side may end up in a different queue. Sadly, when deploying Suricata, this is the common scenario when using span ports or taps.

The problem here is that by having both sides of the traffic in different queues, the order of processing of packets becomes unpredictable. Timing differences on the NIC, the driver, the kernel and in Suricata will lead to a high chance of packets coming in at a different order than on the wire. This is specifically about a mismatch between the two traffic directions. For example, Suricata tracks the TCP 3-way handshake. Due to this timing issue, the SYN/ACK may only be received by Suricata long after the client to server side has already started sending data. Suricata would see this traffic as invalid.

None of the supported capture methods like AF_PACKET, PF_RING or NETMAP can fix this problem for us. It would require buffering and packet reordering which is expensive.

To see how many queues are configured:

```
$ ethtool -l ens2f1
Channel parameters for ens2f1:
Pre-set maximums:
RX:             0
TX:             0
Other:          1
Combined:       64
Current hardware settings:
RX:             0
TX:             0
Other:          1
Combined:       8
```

Some NIC's allow you to set it into a symmetric mode. The Intel X(L)710 card can do this in theory, but the drivers aren't capable of enabling this yet (work is underway to try to address this). Another way to address is by setting a special "Random Secret Key" that will make the RSS symmetrical. See http://www.ndsl.kaist.edu/~kyoungsoo/papers/TR-symRSS.pdf (PDF).

In most scenario's however, the optimal solution is to reduce the number of RSS queues to 1:

Example:

```
# Intel X710 with i40e driver:
ethtool -L $DEV combined 1
```

Some drivers do not support setting the number of queues through ethtool. In some cases there is a module load time option. Read the driver docs for the specifics.

### 9.2.3 Offloading

Network cards, drivers and the kernel itself have various techniques to speed up packet handling. Generally these will all have to be disabled.

LRO/GRO lead to merging various smaller packets into big 'super packets'. These will need to be disabled as they break the dsize keyword as well as TCP state tracking.

Checksum offloading can be left enabled on AF_PACKET and PF_RING, but needs to be disabled on PCAP, NETMAP and others.

### 9.2.4 Recommendations

Read your drivers documentation! E.g. for i40e the ethtool change of RSS queues may lead to kernel panics if done wrong.

Generic: set RSS queues to 1 or make sure RSS hashing is symmetric. Disable NIC offloading.

AF_PACKET: 1 RSS queue and stay on kernel <=4.2 or make sure you have >=4.4.16, >=4.6.5 or >=4.7. Exception: if RSS is symmetric cluster-type 'cluster_qm' can be used to bind Suricata to the RSS queues. Disable NIC offloading except the rx/tx csum.

PF_RING: 1 RSS queue and use cluster-type 'cluster_flow'. Disable NIC offloading except the rx/tx csum.

NETMAP: 1 RSS queue. There is no flow based load balancing built-in, but the 'lb' tool can be helpful. Another option is to use the 'autofp' runmode. Exception: if RSS is symmetric, load balancing is based on the RSS hash and multiple RSS queues can be used. Disable all NIC offloading.

## 9.3 Tuning Considerations

Settings to check for optimal performance.

### 9.3.1 max-pending-packets: <number>

This setting controls the number simultaneous packets that the engine can handle. Setting this higher generally keeps the threads more busy, but setting it too high will lead to degradation.

Suggested setting: 10000 or higher. Max is ~65000. This setting is per thread. The memory is set up at start and the usage is as follows:

```
number_of.threads X max-pending-packets X (default-packet-size + ~750 bytes)
```

### 9.3.2 mpm-algo: <ac|hs|ac-bs|ac-ks>

Controls the pattern matcher algorithm. AC (Aho-Corasick ``) is the default. On supported platforms, :doc:`hyperscan` is the best option. On commodity hardware if Hyperscan is not available the suggested setting is ``mpm-algo: ac-ks (Aho-Corasick `` Ken Steele variant) as it performs better than ``mpm-algo: ac

### 9.3.3 detect.profile: <low|medium|high|custom>

The detection engine tries to split out separate signatures into groups so that a packet is only inspected against signatures that can actually match. As in large rule set this would result in way too many groups and memory usage similar groups are merged together. The profile setting controls how aggressive this merging is done. The default setting of `high` usually is good enough.

The "custom" setting allows modification of the group sizes:

```
custom-values:
  toclient-groups: 100
  toserver-groups: 100
```

In general, increasing will improve performance. It will lead to minimal increase in memory usage. The default value for `toclient-groups` and `toserver-groups` with `detect.profile: high` is 75.

### 9.3.4 detect.sgh-mpm-context: <auto|single|full>

The multi pattern matcher can have it's context per signature group (full) or globally (single). Auto selects between single and full based on the **mpm-algo** selected. ac, ac-bs, ac-ks, hs default to "single". Setting this to "full" with `mpm-algo: ac` or `mpm-algo: ac-ks` offers better performance. Setting this to "full" with `mpm-algo: hs` is not recommended as it leads to much higher startup time. Instead with Hyperscan either `detect.profile: high` or bigger custom group size settings can be used as explained above which offers better performance than `ac` and `ac-ks` even with `detect.sgh-mpm-context: full`.

### 9.3.5 af-packet

If using `af-packet` (default on Linux) it is recommended that af-packet v3 is used for IDS/NSM deployments. For IPS it is recommended af-packet v2. To make sure af-packet v3 is used it can specifically be enforced it in the `af-packet` config section of suricata.yaml like so:

```
af-packet:
 - interface: eth0
   ....
   ....
   ....
   use-mmap: yes
   tpacket-v3: yes
```

### 9.3.6 ring-size

Ring-size is another `af-packet` variable that can be considered for tuning and performance benefits. It basically means the buffer size for packets per thread. So if the setting is `ring-size: 100000` like below:

```
af-packet:
 - interface: eth0
   threads: 5
   ring-size: 100000
```

it means there will be 100,000 packets allowed in each buffer of the 5 threads. If any of the buffers gets filled (for example packet processing can not keep up) that will result in packet `drop` counters increasing in the stats logs.

The memory used for those is set up and dedicated at start and is calculated as follows:

:: af-packet.threads X af-packet.ring-size X (default-packet-size + ~750 bytes)

where `af-packet.threads`, `af-packet.ring-size`, `default-packet-size` are the values set in suricata.yaml. Config values for example for af-packet could be quickly displayed with on the command line as well with `suricata --dump-config |grep af-packet`.

### 9.3.7 stream.bypass

Another option that can be used to improve performance is `stream.bypass`. In the example below:

```
stream:
 memcap: 64mb
 checksum-validation: yes      # reject wrong csums
 inline: auto                  # auto will use inline mode in IPS mode, yes or no set␣
↪it statically
 bypass: yes
 reassembly:
   memcap: 256mb
   depth: 1mb                  # reassemble 1mb into a stream
   toserver-chunk-size: 2560
   toclient-chunk-size: 2560
   randomize-chunk-size: yes
```

Inspection will be skipped when `stream.reassembly.depth` of 1mb is reached for a particular flow.

## 9.4 Hyperscan

### 9.4.1 Introduction

"Hyperscan is a high-performance multiple regex matching library." https://www.hyperscan.io

In Suricata it can be used to perform multi pattern matching (mpm). Support was implemented by Justin Viiret and Jim Xu from Intel: https://github.com/inliniac/suricata/pull/1965, https://redmine.openinfosecfoundation.org/issues/1704

### 9.4.2 Compilation

It's possible to pass –with-libhs-includes=/usr/local/include/hs/ –with-libhs-libraries=/usr/local/lib/, although by default this shouldn't be necessary. Suricata should pick up Hyperscan's pkg-config file automagically.

When Suricata's compilation succeeded, you should have:

```
suricata --build-info|grep Hyperscan
  Hyperscan support:                       yes
```

### 9.4.3 Using Hyperscan

To use the hyperscan support edit your suricata.yaml. Change the mpm-algo and spm-algo values to 'hs'.

Alternatively, use this commandline option: –set mpm-algo=hs –set spm-algo=hs

### 9.4.4 Ubuntu Hyperscan Installation

To use Suricata with Hyperscan support, install dependencies:

```
apt-get install cmake ragel
```

#### libboost headers

Hyperscan needs the libboost headers from 1.58+.

On Ubuntu 15.10 or 16.04+, simply do:

```
apt-get install libboost-dev
```

#### Trusty

Trusty has 1.57, so it's too old. We can grab a newer libboost version, but we *don't* install it system wide. It's only the headers we care about during compilation of Hyperscan.

```
sudo apt-get python-dev libbz2-dev
wget https://dl.bintray.com/boostorg/release/1.66.0/source/boost_1_66_0.tar.gz
tar xvzf boost_1_66_0.tar.gz
cd boost_1_66_0
./bootstrap.sh --prefix=~/tmp/boost-1.66
./b2 install
```

#### Hyperscan

We'll install version 5.0.0.

```
git clone https://github.com/intel/hyperscan
cd hyperscan
mkdir build
cd build
cmake -DBUILD_STATIC_AND_SHARED=1 ../
```

If you have your own libboost headers, use this cmake line instead:

```
cmake -DBUILD_STATIC_AND_SHARED=1 -DBOOST_ROOT=~/tmp/boost-1.66 ../
```

Finally, make and make install:

```
make
sudo make install
```

Compilation can take a long time, but it should in the end look something like this:

```
Install the project...
-- Install configuration: "RELWITHDEBINFO"
-- Installing: /usr/local/lib/pkgconfig/libhs.pc
-- Up-to-date: /usr/local/include/hs/hs.h
-- Up-to-date: /usr/local/include/hs/hs_common.h
-- Up-to-date: /usr/local/include/hs/hs_compile.h
```

(continues on next page)

(continued from previous page)

```
-- Up-to-date: /usr/local/include/hs/hs_runtime.h
-- Installing: /usr/local/lib/libhs_runtime.a
-- Installing: /usr/local/lib/libhs_runtime.so.4.2.0
-- Installing: /usr/local/lib/libhs_runtime.so.4.2
-- Installing: /usr/local/lib/libhs_runtime.so
-- Installing: /usr/local/lib/libhs.a
-- Installing: /usr/local/lib/libhs.so.4.2.0
-- Installing: /usr/local/lib/libhs.so.4.2
-- Installing: /usr/local/lib/libhs.so
```

Note that you may have to add /usr/local/lib to your ld search path

```
echo "/usr/local/lib" | sudo tee --append /etc/ld.so.conf.d/usrlocal.conf
sudo ldconfig
```

## 9.5 High Performance Configuration

### 9.5.1 NIC

One of the major dependencies for Suricata's performance is the Network Interface Card. There are many vendors and possibilities. Some NICs have and require their own specific instructions and tools of how to set up the NIC. This ensures the greatest benefit when running Suricata. Vendors like Napatech, Netronome, Accolade, Myricom include those tools and documentation as part of their sources.

For Intel, Mellanox and commodity NICs the following suggestions below could be utilized.

It is recommended that the latest available stable NIC drivers are used. In general when changing the NIC settings it is advisable to use the latest `ethtool` version. Some NICs ship with their own `ethtool` that is recommended to be used. Here is an example of how to set up the ethtool if needed:

```
wget https://mirrors.edge.kernel.org/pub/software/network/ethtool/ethtool-5.2.tar.xz
tar -xf ethtool-5.2.tar.xz
cd ethtool-5.2
./configure && make clean && make && make install
/usr/local/sbin/ethtool --version
```

When doing high performance optimisation make sure `irqbalance` is off and not running:

```
service irqbalance stop
```

Depending on the NIC's available queues (for example Intel's x710/i40 has 64 available per port/interface) the worker threads can be set up accordingly. Usually the available queues can be seen by running:

```
/usr/local/sbin/ethtool -l eth1
```

Some NICs - generally lower end 1Gbps - do not support symmetric hashing see *Packet Capture*. On those systems due to considerations for out of order packets the following setup with af-packet is suggested (the example below uses `eth1`):

```
/usr/local/sbin/ethtool -L eth1 combined 1
```

then set up af-packet with number of desired workers threads `threads: auto` (auto by default will use number of CPUs available) and `cluster-type: cluster_flow` (also the default setting)

For higher end systems/NICs a better and more performant solution could be utilizing the NIC itself a bit more. x710/i40 and similar Intel NICs or Mellanox MT27800 Family [ConnectX-5] for example can easily be set up to do a bigger chunk of the work using more RSS queues and symmetric hashing in order to allow for increased performance on the Suricata side by using af-packet with `cluster-type: cluster_qm` mode. In that mode with af-packet all packets linked by network card to a RSS queue are sent to the same socket. Below is an example of a suggested config set up based on a 16 core one CPU/NUMA node socket system using x710:

```
rmmod i40e && modprobe i40e
ifconfig eth1 down
/usr/local/sbin/ethtool -L eth1 combined 16
/usr/local/sbin/ethtool -K eth1 rxhash on
/usr/local/sbin/ethtool -K eth1 ntuple on
ifconfig eth1 up
/usr/local/sbin/ethtool -X eth1 hkey␣
↪6D:5A:6D:5A:6D:5A:
```

below uses a 72 core machine and the sniffing NIC that Suricata uses located on NUMA node 1. In such 2 socket configurations it is recommended to have Suricata and the sniffing NIC to be running and residing on the second NUMA node as by default CPU 0 is widely used by many services in Linux. In a case where this is not possible it is recommended that (via the cpu affinity config section in suricata.yaml and the irq affinity script for the NIC) CPU 0 is never used.

In the case below 36 worker threads are used out of NUMA node 1's CPU, af-packet runmode with `cluster-type: cluster_qm`.

If the CPU's NUMA set up is as follows:

```
lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                72
On-line CPU(s) list:   0-71
Thread(s) per core:    2
Core(s) per socket:    18
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 79
Model name:            Intel(R) Xeon(R) CPU E5-2697 v4 @ 2.30GHz
Stepping:              1
CPU MHz:               1199.724
CPU max MHz:           3600.0000
CPU min MHz:           1200.0000
BogoMIPS:              4589.92
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              46080K
NUMA node0 CPU(s):     0-17,36-53
NUMA node1 CPU(s):     18-35,54-71
```

It is recommended that 36 worker threads are used and the NIC set up could be as follows:

```
rmmod i40e && modprobe i40e
ifconfig eth1 down
/usr/local/sbin/ethtool -L eth1 combined 36
/usr/local/sbin/ethtool -K eth1 rxhash on
/usr/local/sbin/ethtool -K eth1 ntuple on
ifconfig eth1 up
./set_irq_affinity local eth1
/usr/local/sbin/ethtool -X eth1 hkey␣
↪6D:5A:6D:5A:6D:5A:
```

section of suricata.yaml config:

```
# Suricata is multi-threaded. Here the threading can be influenced.
threading:
 cpu-affinity:
   - management-cpu-set:
       cpu: [ "1-10" ]  # include only these CPUs in affinity settings
   - receive-cpu-set:
       cpu: [ "0-10" ]  # include only these CPUs in affinity settings
   - worker-cpu-set:
       cpu: [ "18-35", "54-71" ]
       mode: "exclusive"
       prio:
         low: [ 0 ]
         medium: [ "1" ]
         high: [ "18-35","54-71" ]
         default: "high"
```

In the af-packet section of suricata.yaml config :

```
- interface: eth1
  # Number of receive threads. "auto" uses the number of cores
  threads: 18
  cluster-id: 99
  cluster-type: cluster_qm
  defrag: no
  use-mmap: yes
  mmap-locked: yes
  tpacket-v3: yes
  ring-size: 100000
  block-size: 1048576
- interface: eth1
  # Number of receive threads. "auto" uses the number of cores
  threads: 18
  cluster-id: 99
  cluster-type: cluster_qm
  defrag: no
  use-mmap: yes
  mmap-locked: yes
  tpacket-v3: yes
  ring-size: 100000
  block-size: 1048576
```

That way 36 worker threads can be mapped (18 per each af-packet interface slot) in total per CPUs NUMA 1 range - 18-35,54-71. That part is done via the `worker-cpu-set` affinity settings. `ring-size` and `block-size` in the config section above are decent default values to start with. Those can be better adjusted if needed as explained in *Tuning Considerations*.

### AMD based systems

Another example can be using an AMD based system where the architecture and design of the system itself plus the NUMA node's interaction is different as it is based on the HyperTransport (HT) technology. In that case per NUMA thread/lock would not be needed. The example below shows a suggestion for such a configuration utilising af-packet, `cluster-type: cluster_flow`. The Mellanox NIC is located on NUMA 0.

The CPU set up is as follows:

```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                128
On-line CPU(s) list:   0-127
Thread(s) per core:    2
Core(s) per socket:    32
Socket(s):             2
NUMA node(s):          8
Vendor ID:             AuthenticAMD
CPU family:            23
Model:                 1
Model name:            AMD EPYC 7601 32-Core Processor
Stepping:              2
CPU MHz:               1200.000
CPU max MHz:           2200.0000
CPU min MHz:           1200.0000
BogoMIPS:              4391.55
Virtualization:        AMD-V
L1d cache:             32K
L1i cache:             64K
L2 cache:              512K
L3 cache:              8192K
NUMA node0 CPU(s):     0-7,64-71
NUMA node1 CPU(s):     8-15,72-79
NUMA node2 CPU(s):     16-23,80-87
NUMA node3 CPU(s):     24-31,88-95
NUMA node4 CPU(s):     32-39,96-103
NUMA node5 CPU(s):     40-47,104-111
NUMA node6 CPU(s):     48-55,112-119
NUMA node7 CPU(s):     56-63,120-127
```

The `ethtool`, `show_irq_affinity.sh` and `set_irq_affinity_cpulist.sh` tools are provided from the official driver sources. Set up the NIC, including offloading and load balancing:

```
ifconfig eno6 down
/opt/mellanox/ethtool/sbin/ethtool -L eno6 combined 15
/opt/mellanox/ethtool/sbin/ethtool -K eno6 rxhash on
/opt/mellanox/ethtool/sbin/ethtool -K eno6 ntuple on
ifconfig eno6 up
/sbin/set_irq_affinity_cpulist.sh 1-7,64-71 eno6
/opt/mellanox/ethtool/sbin/ethtool -X eno6 hfunc toeplitz
/opt/mellanox/ethtool/sbin/ethtool -X eno6 hkey␣
↪6D:5A:6D:5A:6D:5A:
```

(continued from previous page)

```
cpu-affinity:
  - management-cpu-set:
      cpu: [ "120-127" ]  # include only these cpus in affinity settings
  - receive-cpu-set:
      cpu: [ 0 ]  # include only these cpus in affinity settings
  - worker-cpu-set:
      cpu: [ "8-55" ]
      mode: "exclusive"
      prio:
        high: [ "8-55" ]
        default: "high"
```

In the af-packet section of suricata.yaml config:

```
- interface: eth1
  # Number of receive threads. "auto" uses the number of cores
  threads: 48 # 48 worker threads on cpus "8-55" above
  cluster-id: 99
  cluster-type: cluster_flow
  defrag: no
  use-mmap: yes
  mmap-locked: yes
  tpacket-v3: yes
  ring-size: 100000
  block-size: 1048576
```

In the example above there are 15 RSS queues pinned to cores 1-7,64-71 on NUMA node 0 and 40 worker threads using other CPUs on different NUMA nodes. The reason why CPU 0 is skipped in this set up is as in Linux systems it is very common for CPU 0 to be used by default by many tools/services. The NIC itself in this config is positioned on NUMA 0 so starting with 15 RSS queues on that NUMA node and keeping those off for other tools in the system could offer the best advantage.

---

**Note:** Performance and optimization of the whole system can be affected upon regular NIC driver and pkg/kernel upgrades so it should be monitored regularly and tested out in QA/test environments first. As a general suggestion it is always recommended to run the latest stable firmware and drivers as instructed and provided by the particular NIC vendor.

---

### Other considerations

Another advanced option to consider is the `isolcpus` kernel boot parameter is a way of allowing CPU cores to be isolated for use of general system processes. That way ensures total dedication of those CPUs/ranges for the Suricata process only.

`stream.wrong_thread` / `tcp.pkt_on_wrong_thread` are counters available in `stats.log` or `eve.json` as `event_type: stats` that indicate issues with the load balancing. There could be traffic/NICs settings related as well. In very high/heavily increasing counter values it is recommended to experiment with a different load balancing method either via the NIC or for example using XDP/eBPF. There is an issue open https://redmine.openinfosecfoundation.org/issues/2725 that is a placeholder for feedback and findings.

## 9.6 Statistics

The stats.log produces statistics records on a fixed interval, by default every 8 seconds.

### 9.6.1 stats.log file

```
-------------------------------------------------------------------
Counter                     | TM Name                   | Value
-------------------------------------------------------------------
flow_mgr.closed_pruned      | FlowManagerThread         | 154033
flow_mgr.new_pruned         | FlowManagerThread         | 67800
flow_mgr.est_pruned         | FlowManagerThread         | 100921
flow.memuse                 | FlowManagerThread         | 6557568
flow.spare                  | FlowManagerThread         | 10002
flow.emerg_mode_entered     | FlowManagerThread         | 0
flow.emerg_mode_over        | FlowManagerThread         | 0
decoder.pkts                | RxPcapem21                | 450001754
decoder.bytes               | RxPcapem21                | 409520714250
decoder.ipv4                | RxPcapem21                | 449584047
decoder.ipv6                | RxPcapem21                | 9212
decoder.ethernet            | RxPcapem21                | 450001754
decoder.raw                 | RxPcapem21                | 0
decoder.sll                 | RxPcapem21                | 0
decoder.tcp                 | RxPcapem21                | 448124337
decoder.udp                 | RxPcapem21                | 542040
decoder.sctp                | RxPcapem21                | 0
decoder.icmpv4              | RxPcapem21                | 82292
decoder.icmpv6              | RxPcapem21                | 9164
decoder.ppp                 | RxPcapem21                | 0
decoder.pppoe               | RxPcapem21                | 0
decoder.gre                 | RxPcapem21                | 0
decoder.vlan                | RxPcapem21                | 0
decoder.avg_pkt_size        | RxPcapem21                | 910
decoder.max_pkt_size        | RxPcapem21                | 1514
defrag.ipv4.fragments       | RxPcapem21                | 4
defrag.ipv4.reassembled     | RxPcapem21                | 1
defrag.ipv4.timeouts        | RxPcapem21                | 0
defrag.ipv6.fragments       | RxPcapem21                | 0
defrag.ipv6.reassembled     | RxPcapem21                | 0
defrag.ipv6.timeouts        | RxPcapem21                | 0
tcp.sessions                | Detect                    | 41184
tcp.ssn_memcap_drop         | Detect                    | 0
tcp.pseudo                  | Detect                    | 2087
tcp.invalid_checksum        | Detect                    | 8358
tcp.no_flow                 | Detect                    | 0
tcp.reused_ssn              | Detect                    | 11
tcp.memuse                  | Detect                    | 36175872
tcp.syn                     | Detect                    | 85902
tcp.synack                  | Detect                    | 83385
tcp.rst                     | Detect                    | 84326
tcp.segment_memcap_drop     | Detect                    | 0
tcp.stream_depth_reached    | Detect                    | 109
tcp.reassembly_memuse       | Detect                    | 67755264
tcp.reassembly_gap          | Detect                    | 789
detect.alert                | Detect                    | 14721
```

### Detecting packet loss

At shut down, Suricata reports the packet loss statistics it gets from pcap, pfring or afpacket

```
[18088] 30/5/2012 -- 07:39:18 - (RxPcapem21) Packets 451595939, bytes 410869083410
[18088] 30/5/2012 -- 07:39:18 - (RxPcapem21) Pcap Total:451674222 Recv:451596129␣
↪Drop:78093 (0.0%).
```

Usually, this is not the complete story though. These are kernel drop stats, but the NIC may also have dropped packets. Use ethtool to get to those:

```
# ethtool -S em2
NIC statistics:
     rx_packets: 35430208463
     tx_packets: 216072
     rx_bytes: 32454370137414
     tx_bytes: 53624450
     rx_broadcast: 17424355
     tx_broadcast: 133508
     rx_multicast: 5332175
     tx_multicast: 82564
     rx_errors: 47
     tx_errors: 0
     tx_dropped: 0
     multicast: 5332175
     collisions: 0
     rx_length_errors: 0
     rx_over_errors: 0
     rx_crc_errors: 51
     rx_frame_errors: 0
     rx_no_buffer_count: 0
     rx_missed_errors: 0
     tx_aborted_errors: 0
     tx_carrier_errors: 0
     tx_fifo_errors: 0
     tx_heartbeat_errors: 0
     tx_window_errors: 0
     tx_abort_late_coll: 0
     tx_deferred_ok: 0
     tx_single_coll_ok: 0
     tx_multi_coll_ok: 0
     tx_timeout_count: 0
     tx_restart_queue: 0
     rx_long_length_errors: 0
     rx_short_length_errors: 0
     rx_align_errors: 0
     tx_tcp_seg_good: 0
     tx_tcp_seg_failed: 0
     rx_flow_control_xon: 0
     rx_flow_control_xoff: 0
     tx_flow_control_xon: 0
     tx_flow_control_xoff: 0
     rx_long_byte_count: 32454370137414
     rx_csum_offload_good: 35270755306
     rx_csum_offload_errors: 65076
     alloc_rx_buff_failed: 0
     tx_smbus: 0
     rx_smbus: 0
```

(continues on next page)

(continued from previous page)

```
    dropped_smbus: 0
```

### 9.6.2 Kernel drops

stats.log contains interesting information in the capture.kernel_packets and capture.kernel_drops. The meaning of them is different following the capture mode.

In AF_PACKET mode:

- kernel_packets is the number of packets correctly sent to userspace
- kernel_drops is the number of packets that have been discarded instead of being sent to userspace

In PF_RING mode:

- kernel_packets is the total number of packets seen by pf_ring
- kernel_drops is the number of packets that have been discarded instead of being sent to userspace

In the Suricata stats.log the TCP data gap counter is also an indicator, as it accounts missing data packets in TCP streams:

```
tcp.reassembly_gap         | Detect                       | 789
```

Ideally, this number is 0. Not only pkt loss affects it though, also bad checksums and stream engine running out of memory.

### 9.6.3 Tools to plot graphs

Some people made nice tools to plot graphs of the statistics file.

- ipython and matplotlib script
- Monitoring with Zabbix or other and Code on Github

## 9.7 Ignoring Traffic

In some cases there are reasons to ignore certain traffic. Certain hosts may be trusted, or perhaps a backup stream should be ignored.

### 9.7.1 capture filters (BPF)

Through BPFs the capture methods pcap, af-packet and pf_ring can be told what to send to Suricata, and what not. For example a simple filter 'tcp' will only send tcp packets.

If some hosts and or nets need to be ignored, use something like "not (host IP1 or IP2 or IP3 or net NET/24)".

Example:

```
not host 1.2.3.4
```

Capture filters are specified on the commandline after all other options:

```
suricata -i eth0 -v not host 1.2.3.4
suricata -i eno1 -c suricata.yaml tcp or udp
```

Capture filters can be set per interface in the pcap, af-packet, netmap and pf_ring sections. It can also be put in a file:

```
echo "not host 1.2.3.4" > capture-filter.bpf
suricata -i ens5f0 -F capture-filter.bpf
```

Using a capture filter limits what traffic Suricata processes. So the traffic not seen by Suricata will not be inspected, logged or otherwise recorded.

### 9.7.2 pass rules

Pass rules are Suricata rules that if matching, pass the packet and in case of TCP the rest of the flow. They look like normal rules, except that instead of 'alert' or 'drop' they start with 'pass'.

Example:

```
pass ip 1.2.3.4 any <> any any (msg:"pass all traffic from/to 1.2.3.4"; sid:1;)
```

A big difference with capture filters is that logs such as Eve or http.log are still generated for this traffic.

### 9.7.3 suppress

Suppress rules can be used to make sure no alerts are generated for a host. This is not efficient however, as the suppression is only considered post-matching. In other words, Suricata first inspects a rule, and only then will it consider per-host suppressions.

Example:

```
suppress gen_id 0, sig_id 0, track by_src, ip 1.2.3.4
```

### 9.7.4 encrypted traffic

The TLS app layer parser has the ability to stop processing encrypted traffic after the initial handshake. By setting the *app-layer.protocols.tls.encryption-handling* option to *bypass* the rest of this flow is ignored. If flow bypass is enabled, the bypass is done in the kernel or in hardware.

### 9.7.5 bypassing traffic

Aside from using the `bypass` keyword in rules, there are three other ways to bypass traffic.

- Within suricata (local bypass). Suricata reads a packet, decodes it, checks it in the flow table. If the corresponding flow is local bypassed then it simply skips all streaming, detection and output and the packet goes directly out in IDS mode and to verdict in IPS mode.
- Within the kernel (capture bypass). When Suricata decides to bypass it calls a function provided by the capture method to declare the bypass in the capture. For NFQ this is a simple mark that will be used by the ruleset. For AF_PACKET this will be a call to add an element in an eBPF hash table stored in kernel.
- Within the nic driver. This method relies upon XDP, XDP can process the traffic prior to reaching the kernel.

Additional bypass documentation:

https://suricon.net/wp-content/uploads/2017/12/SuriCon17-Manev_Purzynski.pdf https://www.stamus-networks.com/2016/09/28/suricata-bypass-feature/

## 9.8 Packet Profiling

In this guide will be explained how to enable packet profiling and use it with the most recent code of Suricata on Ubuntu. It is based on the assumption that you have already installed Suricata once from the GIT repository.

Packet profiling is convenient in case you would like to know how long packets take to be processed. It is a way to figure out why certain packets are being processed quicker than others, and this way a good tool for developing Suricata.

Update Suricata by following the steps from Installation from Git. Start at the end at

```
cd suricata/oisf
git pull
```

And follow the described next steps. To enable packet profiling, make sure you enter the following during the configuring stage:

```
./configure --enable-profiling
```

Find a folder in which you have pcaps. If you do not have pcaps yet, you can get these with Wireshark. See Sniffing Packets with Wireshark.

Go to the directory of your pcaps. For example:

```
cd  ~/Desktop
```

With the ls command you can see the content of the folder. Choose a folder and a pcap file

for example:

```
cd ~/Desktop/2011-05-05
```

Run Suricata with that pcap:

```
suricata -c /etc/suricata/suricata.yaml -r log.pcap.(followed by the number/name of␣
↪your pcap)
```

for example:

```
suricata -c /etc/suricata/suricata.yaml -r log.pcap.1304589204
```

## 9.9 Rule Profiling

```
--------------------------------------------------------------------------------
Date: 9/5/2013 -- 14:59:58
--------------------------------------------------------------------------------
 Num      Rule         Gid      Rev      Ticks        %      Checks   Matches  Max␣
↪Ticks   Avg Ticks   Avg Match   Avg No Match
-------- ------------ -------- -------- ------------ ------ -------- -------- --------
↪--- ----------- ----------- --------------
```

(continues on next page)

(continued from previous page)

```
1        2210021      1        3        12037        4.96   1        1        12037
↪    12037.00    12037.00    0.00
2        2210054      1        1        107479       44.26  12       0        35805
↪    8956.58     0.00        8956.58
3        2210053      1        1        4513         1.86   1        0        4513
↪    4513.00     0.00        4513.00
4        2210023      1        1        3077         1.27   1        0        3077
↪    3077.00     0.00        3077.00
5        2210008      1        1        3028         1.25   1        0        3028
↪    3028.00     0.00        3028.00
6        2210009      1        1        2945         1.21   1        0        2945
↪    2945.00     0.00        2945.00
7        2210055      1        1        2945         1.21   1        0        2945
↪    2945.00     0.00        2945.00
8        2210007      1        1        2871         1.18   1        0        2871
↪    2871.00     0.00        2871.00
9        2210005      1        1        2871         1.18   1        0        2871
↪    2871.00     0.00        2871.00
10       2210024      1        1        2846         1.17   1        0        2846
↪    2846.00     0.00        2846.00
```

The meaning of the individual fields:

- Ticks – total ticks spent on this rule, so a sum of all inspections
- % – share of this single sig in the total cost of inspection
- Checks – number of times a signature was inspected
- Matches – number of times it matched. This may not have resulted in an alert due to suppression and thresholding.
- Max ticks – single most expensive inspection
- Avg ticks – per inspection average, so "ticks" / "checks".
- Avg match – avg ticks spent resulting in match
- Avg No Match – avg ticks spent resulting in no match.

The "ticks" are CPU clock ticks: http://en.wikipedia.org/wiki/CPU_time

## 9.10 Tcmalloc

'tcmalloc' is a library Google created as part of the google-perftools suite for improving memory handling in a threaded program. It's very simple to use and does work fine with Suricata. It leads to minor speed ups and also reduces memory usage quite a bit.

### 9.10.1 Installation

On Ubuntu, install the libtcmalloc-minimal0 package:

```
apt-get install libtcmalloc-minimal0
```

On Fedora, install the gperftools-libs package:

```
yum install gperftools-libs
```

### 9.10.2 Usage

Use the tcmalloc by preloading it:

Ubuntu:

```
LD_PRELOAD="/usr/lib/libtcmalloc_minimal.so.0" suricata -c suricata.yaml -i eth0
```

Fedora:

```
LD_PRELOAD="/usr/lib64/libtcmalloc_minimal.so.4" suricata -c suricata.yaml -i eth0
```

CHAPTER 10

# Configuration

## 10.1 Suricata.yaml

Suricata uses the Yaml format for configuration. The Suricata.yaml file included in the source code, is the example configuration of Suricata. This document will explain each option.

At the top of the YAML-file you will find % YAML 1.1. Suricata reads the file and identifies the file as YAML.

### 10.1.1 Max-pending-packets

With the max-pending-packets setting you can set the number of packets you allow Suricata to process simultaneously. This can range from one packet to tens of thousands/hundreds of thousands of packets. It is a trade of higher performance and the use of more memory (RAM), or lower performance and less use of memory. A high number of packets being processed results in a higher performance and the use of more memory. A low number of packets, results in lower performance and less use of memory. Choosing a low number of packets being processed while having many CPU's/CPU cores, can result in not making use of the whole computer-capacity. (For instance: using one core while having three waiting for processing packets.)

```
max-pending-packets: 1024
```

### 10.1.2 Runmodes

By default the runmode option is disabled With the runmodes setting you can set the runmode you would like to use. For all runmodes available, enter **–list-runmodes** in your command line. For more information, see *Runmodes*.

```
runmode: autofp
```

### 10.1.3 Default-packet-size

For the max-pending-packets option, Suricata has to keep packets in memory. With the default-packet-size option, you can set the size of the packets on your network. It is possible that bigger packets have to be processed sometimes. The engine can still process these bigger packets, but processing it will lower the performance.

```
default-packet-size: 1514
```

### 10.1.4 User and group

It is possible to set the user and group to run Suricata as:

```
run-as:
  user: suri
  group: suri
```

### 10.1.5 PID File

This option sets the name of the PID file when Suricata is run in daemon mode. This file records the Suricata process ID.

```
pid-file: /var/run/suricata.pid
```

**Note:** This configuration file option only sets the PID file when running in daemon mode. To force creation of a PID file when not running in daemon mode, use the *--pidfile* command line option.

Also, if running more than one Suricata process, each process will need to specify a different pid-file location.

### 10.1.6 Action-order

All signatures have different properties. One of those is the Action property. This one determines what will happen when a signature matches. There are four types of Action. A summary of what will happen when a signature matches and contains one of those Actions:

1) Pass

If a signature matches and contains pass, Suricata stops scanning the packet and skips to the end of all rules (only for the current packet).

2) Drop

This only concerns the IPS/inline mode. If the program finds a signature that matches, containing drop, it stops immediately. The packet will not be sent any further. Drawback: The receiver does not receive a message of what is going on, resulting in a time-out (certainly with TCP). Suricata generates an alert for this packet.

3) Reject

This is an active rejection of the packet. Both receiver and sender receive a reject packet. There are two types of reject packets that will be automatically selected. If the offending packet concerns TCP, it will be a Reset-packet. For all other protocols it will be an ICMP-error packet. Suricata also generates an alert. When in Inline/IPS mode, the offending packet will also be dropped like with the 'drop' action.

4) Alert

If a signature matches and contains alert, the packet will be treated like any other non-threatening packet, except for this one an alert will be generated by Suricata. Only the system administrator can notice this alert.

Inline/IPS can block network traffic in two ways. One way is by drop and the other by reject.

Rules will be loaded in the order of which they appear in files. But they will be processed in a different order. Signatures have different priorities. The most important signatures will be scanned first. There is a possibility to change the order of priority. The default order is: pass, drop, reject, alert.

```
action-order:
 - pass
 - drop
 - reject
 - alert
```

This means a pass rule is considered before a drop rule, a drop rule before a reject rule and so on.

### 10.1.7 Splitting configuration in multiple files

Some users might have a need or a wish to split their suricata.yaml file in to separate files, this is available vis the 'include' and '!include' keyword. The first example is of taking the contents of the outputs section and storing them in outputs.yaml

```
# outputs.yaml
- fast
    enabled: yes
    filename: fast.log
    append: yes

- unified2-alert:
    enabled: yes

...
```

```
# suricata.yaml
...

outputs: !include outputs.yaml

...
```

The second scenario is where multiple sections are migrated to a different YAML file.

```
# host_1.yaml

max-pending-packets: 2048

outputs:
    - fast
        enabled: yes
        filename: fast.log
        append: yes

    - unified2-alert:
        enabled: yes
```

```
# suricata.yaml

include: host_1.yaml

...
```

If the same section, say outputs is later redefined after the include statement it will overwrite the included file. Therefor any include statement at the end of the document will overwrite the already configured sections.

### 10.1.8 Event output

#### Default logging directory

In the /var/log/suricata directory, all of Suricata's output (alerts and events) will be stored.

```
default-log-dir: /var/log/suricata
```

This directory can be overridden by entering the -l command line parameter or by changing the directory directly in Yaml. To change it with the -l command line parameter, enter the following:

```
suricata -c suricata.yaml -i eth0 -l /var/log/suricata-logs/
```

#### Stats

Engine statistics such as packet counters, memory use counters and others can be logged in several ways. A separate text log 'stats.log' and an EVE record type 'stats' are enabled by default.

The stats have a global configuration and a per logger configuration. Here the global config is documented.

```
# global stats configuration
stats:
  enabled: yes
  # The interval field (in seconds) controls at what interval
  # the loggers are invoked.
  interval: 8
  # Add decode events as stats.
  #decoder-events: true
  # Decoder event prefix in stats. Has been 'decoder' before, but that leads
  # to missing events in the eve.stats records. See issue #2225.
  #decoder-events-prefix: "decoder.event"
  # Add stream events as stats.
  #stream-events: false
```

Statistics can be *enabled* or disabled here.

Statistics are dumped on an *interval*. Setting this below 3 or 4 seconds is not useful due to how threads are synchronized internally.

The decoder events that the decoding layer generates, can create a counter per event type. This behaviour is enabled by default. The *decoder-events* option can be set to *false* to disable.

In 4.1.x there was a naming clash between the regular decoder counters and the decoder-event counters. This lead to a fair amount of decoder-event counters not being shown in the EVE.stats records. To address this without breaking existing setups, a config option *decoder-events-prefix* was added to change the naming of the decoder-events from decoder.<proto>.<event> to decoder.event.<proto>.<event>. In 5.0 this became the default. See issue 2225.

Similar to the *decoder-events* option, the *stream-events* option controls whether the stream-events are added as counters as well. This is disabled by default.

### Outputs

There are several types of output. The general structure is:

```
outputs:
  - fast:
    enabled: yes
    filename: fast.log
    append: yes/no
```

Enabling all of the logs, will result in a much lower performance and the use of more disc space, so enable only the outputs you need.

### Line based alerts log (fast.log)

This log contains alerts consisting of a single line. Example of the appearance of a single fast.log-file line:

```
10/05/10-10:08:59.667372  [**] [1:2009187:4] ET WEB_CLIENT ACTIVEX iDefense
  COMRaider ActiveX Control Arbitrary File Deletion [**] [Classification: Web
  Application Attack] [Priority: 3] {TCP} xx.xx.232.144:80 -> 192.168.1.4:56068
```

```
-fast:                    #The log-name.
   enabled:yes            #This log is enabled. Set to 'no' to disable.
   filename: fast.log     #The name of the file in the default logging directory.
   append: yes/no         #If this option is set to yes, the last filled fast.log-
↪file will not be
                          #overwritten while restarting Suricata.
```

### Eve (Extensible Event Format)

This is an JSON output for alerts and events. It allows for easy integration with 3rd party tools like logstash.

```
outputs:
  # Extensible Event Format (nicknamed EVE) event log in JSON format
  - eve-log:
      enabled: yes
      filetype: regular #regular|syslog|unix_dgram|unix_stream|redis
      filename: eve.json
      #prefix: "@cee: " # prefix to prepend to each log entry
      # the following are valid when type: syslog above
      #identity: "suricata"
      #facility: local5
      #level: Info ## possible levels: Emergency, Alert, Critical,
                   ## Error, Warning, Notice, Info, Debug
      #redis:
      #  server: 127.0.0.1
      #  port: 6379
      #  async: true ## if redis replies are read asynchronously
      #  mode: list ## possible values: list|lpush (default), rpush, channel|publish
      #             ## lpush and rpush are using a Redis list. "list" is an alias for␣
↪lpush
```

(continues on next page)

(continued from previous page)

```
      #             ## publish is using a Redis channel. "channel" is an alias for␣
↪publish
      #  key: suricata ## key or channel to use (default to suricata)
      # Redis pipelining set up. This will enable to only do a query every
      # 'batch-size' events. This should lower the latency induced by network
      # connection at the cost of some memory. There is no flushing implemented
      # so this setting as to be reserved to high traffic suricata.
      #  pipelining:
      #    enabled: yes ## set enable to yes to enable query pipelining
      #    batch-size: 10 ## number of entry to keep in buffer

      # Include top level metadata. Default yes.
      #metadata: no

      types:
        - alert:
            # payload: yes             # enable dumping payload in Base64
            # payload-buffer-size: 4kb # max size of payload buffer to output in eve-
↪log
            # payload-printable: yes   # enable dumping payload in printable (lossy)␣
↪format
            # packet: yes              # enable dumping of packet (without stream␣
↪segments)
            # http-body: yes           # Requires metadata; enable dumping of http␣
↪body in Base64
            # http-body-printable: yes # Requires metadata; enable dumping of http␣
↪body in printable format

            # Enable the logging of tagged packets for rules using the
            # "tag" keyword.
            tagged-packets: yes

            # Configure the metadata to be logged along with an
            # alert. The following shows the default configuration
            # which is used if this field is not provided or simply
            # set to a truthful value. Setting of this section is only
            # required if you wish to enable/disable specific fields.
            #metadata:

              # Include the decoded application layer (ie. http, dns)
              app-layer: true

              # Log the the current state of the flow record.
              flow: true

              rule:
                # Log the metadata field from the rule in a structured
                # format.
                metadata: true

                # Log the raw rule text.
                raw: false

            # HTTP X-Forwarded-For support by adding an extra field or overwriting
            # the source or destination IP address (depending on flow direction)
            # with the one reported in the X-Forwarded-For HTTP header. This is
            # helpful when reviewing alerts for traffic that is being reverse
```

(continues on next page)

(continued from previous page)

```
            # or forward proxied.
            xff:
              enabled: no
              # Two operation modes are available, "extra-data" and "overwrite".
              mode: extra-data
              # Two proxy deployments are supported, "reverse" and "forward". In
              # a "reverse" deployment the IP address used is the last one, in a
              # "forward" deployment the first IP address is used.
              deployment: reverse
              # Header name where the actual IP address will be reported, if more
              # than one IP address is present, the last IP address will be the
              # one taken into consideration.
              header: X-Forwarded-For
        - http:
            extended: yes     # enable this for extended logging information
            # custom allows additional http fields to be included in eve-log
            # the example below adds three additional fields when uncommented
            #custom: [Accept-Encoding, Accept-Language, Authorization]
        - dns:
            # Use version 2 logging with the new format:
            # dns answers will be logged in one single event
            # rather than an event for each of the answers.
            # Without setting a version the version
            # will fallback to 1 for backwards compatibility.
            version: 2

            # Enable/disable this logger. Default: enabled.
            #enabled: no

            # Control logging of requests and responses:
            # - requests: enable logging of DNS queries
            # - responses: enable logging of DNS answers
            # By default both requests and responses are logged.
            #requests: no
            #responses: no

            # Format of answer logging:
            # - detailed: array item per answer
            # - grouped: answers aggregated by type
            # Default: all
            #answer-format: [detailed, grouped]

            # Answer types to log.
            # Default: all
            #answer-types: [a, aaaa, cname, mx, ns, ptr, txt]
        - dns:
            # Version 1 (deprecated) DNS logger.
            version: 1

            enabled: no
            # control logging of queries and answers
            # default yes, no to disable
            query: yes     # enable logging of DNS queries
            answer: yes    # enable logging of DNS answers
            # control which RR types are logged
            # all enabled if custom not specified
            #custom: [a, aaaa, cname, mx, ns, ptr, txt]
```

(continues on next page)

(continued from previous page)

```
        - tls:
            extended: yes     # enable this for extended logging information
            # output TLS transaction where the session is resumed using a
            # session id
            #session-resumption: no
            # custom allows to control which tls fields that are included
            # in eve-log
            #custom: [subject, issuer, session_resumed, serial, fingerprint, sni,␣
↪version, not_before, not_after, certificate, chain]
        - files:
            force-magic: no   # force logging magic on all logged files
            # force logging of checksums, available hash functions are md5,
            # sha1 and sha256
            #force-hash: [md5]
        #- drop:
        #    alerts: yes      # log alerts that caused drops
        #    flows: all       # start or all: 'start' logs only a single drop
        #                     # per flow direction. All logs each dropped pkt.
        - smtp:
            #extended: yes # enable this for extended logging information
            # this includes: bcc, message-id, subject, x_mailer, user-agent
            # custom fields logging from the list:
            #  reply-to, bcc, message-id, subject, x-mailer, user-agent, received,
            #  x-originating-ip, in-reply-to, references, importance, priority,
            #  sensitivity, organization, content-md5, date
            #custom: [received, x-mailer, x-originating-ip, relays, reply-to, bcc]
            # output md5 of fields: body, subject
            # for the body you need to set app-layer.protocols.smtp.mime.body-md5
            # to yes
            #md5: [body, subject]

        # NFS logging. Requires Rust.
        - nfs
        # IKEv2 logging. Requires Rust.
        - ikev2
        - ssh
        - stats:
            totals: yes       # stats for all threads merged together
            threads: no       # per thread stats
            deltas: no        # include delta values
        - dhcp:
            # DHCP logging requires Rust.
            enabled: yes
            # When extended mode is on, all DHCP messages are logged
            # with full detail. When extended mode is off (the
            # default), just enough information to map a MAC address
            # to an IP address is logged.
            extended: no
        # bi-directional flows
        - flow
        # uni-directional flows
        #- netflow

        # An event for logging metadata, specifically pktvars when
        # they are set, but will also include the full metadata object.
        #- metadata
```

For more advanced configuration options, see *Eve JSON Output*.

The format is documented in *Eve JSON Format*.

### Alert output for use with Barnyard2 (unified2.alert)

**Note:** Unified2 output has been deprecated and will be removed by December 2019. Please see the deprecation policy for more information.

This log format is a binary format compatible with the unified2 output of another popular IDS format and is designed for use with Barnyard2 or other tools that consume the unified2 log format.

By default a file with the given filename and a timestamp (unix epoch format) will be created until the file hits the configured size limit, then a new file, with a new timestamp will be created. It is the job of other tools, such as Barnyard2 to cleanup old unified2 files.

If the *nostamp* option is set the log file will not have a timestamp appended. The file will be re-opened on SIGHUP like other log files allowing external log rotation tools to work as expected. However, if the limit is reach the file will be deleted and re-opened.

This output supports IPv6 and IPv4 events.

```
- unified2-alert:
    enabled: yes

    # The filename to log to in the default log directory. A
    # timestamp in unix epoch time will be appended to the filename
    # unless nostamp is set to yes.
    filename: unified2.alert

    # File size limit.  Can be specified in kb, mb, gb.  Just a number
    # is parsed as bytes.
    #limit: 32mb

    # By default unified2 log files have the file creation time (in
    # unix epoch format) appended to the filename. Set this to yes to
    # disable this behavior.
    #nostamp: no

    # Sensor ID field of unified2 alerts.
    #sensor-id: 0

    # Include payload of packets related to alerts. Defaults to true, set to
    # false if payload is not required.
    #payload: yes

    # HTTP X-Forwarded-For support by adding the unified2 extra header or
    # overwriting the source or destination IP address (depending on flow
    # direction) with the one reported in the X-Forwarded-For HTTP header.
    # This is helpful when reviewing alerts for traffic that is being reverse
    # or forward proxied.
    xff:
      enabled: no
      # Two operation modes are available, "extra-data" and "overwrite". Note
      # that in the "overwrite" mode, if the reported IP address in the HTTP
      # X-Forwarded-For header is of a different version of the packet
```

(continues on next page)

(continued from previous page)

```
      # received, it will fall-back to "extra-data" mode.
      mode: extra-data
      # Two proxy deployments are supported, "reverse" and "forward". In
      # a "reverse" deployment the IP address used is the last one, in a
      # "forward" deployment the first IP address is used.
      deployment: reverse
      # Header name where the actual IP address will be reported, if more
      # than one IP address is present, the last IP address will be the
      # one taken into consideration.
      header: X-Forwarded-For
```

This alert output needs Barnyard2.

### A line based log of HTTP requests (http.log)

This log keeps track of all HTTP-traffic events. It contains the HTTP request, hostname, URI and the User-Agent. This information will be stored in the http.log (default name, in the suricata log directory). This logging can also be performed through the use of the *Eve-log capability*.

Example of a HTTP-log line with non-extended logging:

```
07/01/2014-04:20:14.338309 vg.no [**] / [**] Mozilla/5.0 (Macintosh; Intel Mac OS X␣
↪10_9_2)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0.1916.114 Safari/537.36 [**]
192.168.1.6:64685 -> 195.88.54.16:80
```

Example of a HTTP-log line with extended logging:

```
07/01/2014-04:21:06.994705 vg.no [**] / [**] Mozilla/5.0 (Macintosh; Intel Mac OS X␣
↪10_9_2)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0.1916.114 Safari/537.36 [**] <no␣
↪referer> [**]
GET [**] HTTP/1.1 [**] 301 => http://www.vg.no/ [**] 239 bytes [**] 192.168.1.6:64726␣
↪-> 195.88.54.16:80
```

```
- http-log:                        #The log-name.
    enabled: yes                   #This log is enabled. Set 'no' to disable.
    filename: http.log             #The name of the file in the default logging␣
↪directory.
    append: yes/no                 #If this option is set to yes, the last filled http.
↪log-file will not be
                                   # overwritten while restarting Suricata.
    extended: yes                  # If set to yes more information is written about the␣
↪event.
```

### Packet log (pcap-log)

With the pcap-log option you can save all packets, that are registered by Suricata, in a log file named _log.pcap_. This way, you can take a look at all packets whenever you want. In the normal mode a pcap file is created in the default-log-dir. It can also be created elsewhere if a absolute path is set in the yaml-file.

The file that is saved in example the default -log-dir /var/log/suricata, can be be opened with every program which supports the pcap file format. This can be Wireshark, TCPdump, Suricata, Snort and many others.

The pcap-log option can be enabled and disabled.

There is a size limit for the pcap-log file that can be set. The default limit is 32 MB. If the log-file reaches this limit, the file will be rotated and a new one will be created. The pcap-log option has an extra functionality for "Sguil":http://sguil.sourceforge.net/ that can be enabled in the 'mode' option. In the sguil mode the "sguil_base_dir" indicates the base directory. In this base dir the pcaps are created in a Sguil-specific directory structure that is based on the day:

```
$sguil_base_dir/YYYY-MM-DD/$filename.<timestamp>
```

If you would like to use Suricata with Sguil, do not forget to enable (and if necessary modify) the base dir in the suricata.yaml file. Remember that in the 'normal' mode, the file will be saved in default-log-dir or in the absolute path (if set).

The pcap files can be compressed before being written to disk by setting the compression option to lz4. This option is incompatible with sguil mode. Note: On Windows, this option increases disk I/O instead of reducing it. When using lz4 compression, you can enable checksums using the lz4-checksum option, and you can set the compression level lz4-level to a value between 0 and 16, where higher levels result in higher compression.

By default all packets are logged except:

- TCP streams beyond stream.reassembly.depth
- encrypted streams after the key exchange

```
- pcap-log:
    enabled:  yes
    filename: log.pcap

    # Limit in MB.
    limit: 32

    mode: sguil # "normal" (default) or sguil.
    sguil_base_dir: /nsm_data/
```

### Verbose Alerts Log (alert-debug.log)

This is a log type that gives supplementary information about an alert. It is particularly convenient for people who investigate false positives and who write signatures. However, it lowers the performance because of the amount of information it has to store.

```
- alert-debug:                  #The log-name.
    enabled: no                 #This log is not enabled. Set 'yes' to enable.
    filename: alert-debug.log   #The name of the file in the default logging␣
↪directory.
    append: yes/no              #If this option is set to yes, the last filled fast.
↪log-file will not be
                                # overwritten while restarting Suricata.
```

### Alert output to prelude (alert-prelude)

To be able to use this type, you have to connect with the prelude manager first.

Prelude alerts contain a lot of information and fields, including the IPfields in of the packet which triggered the alert. This information can be divided in three parts:

- The alert description (sensor name, date, ID (sid) of the rule, etc). This is always included
- The packets headers (almost all IP fields, TCP UDP etc. if relevant)

- A binary form of the entire packet.

Since the last two parts can be very big (especially since they are stored in the Prelude SQL database), they are optional and controlled by the two options 'log_packet_header' and 'log_packet_content'. The default setting is to log the headers, but not the content.

The profile name is the name of the Prelude profile used to connect to the prelude manager. This profile must be registered using an external command (prelude-admin), and must match the uid/gid of the user that will run Suricata. The complete procedure is detailed in the Prelude Handbook.

```
- alert-prelude:                  #The log-name.
    enabled: no                   #This log is not enabled. Set 'yes' to enable.
    profile: suricata             #The profile-name used to connect to the prelude␣
↪manager.
    log_packet_content: no        #The log_packet_content is disabled by default.
    log_packet_header: yes        #The log _packet_header is enabled by default.
```

### Stats

In stats you can set the options for stats.log. When enabling stats.log you can set the amount of time in seconds after which you want the output-data to be written to the log file.

```
- stats:
    enabled: yes                  #By default, the stats-option is enabled
    filename: stats.log           #The log-name. Combined with the default logging␣
↪directory
                                  #(default-log-dir) it will result in /var/log/
↪suricata/stats.log.
                                  #This directory can be overruled with a absolute path.
↪ (A
                                  #directory starting with / ).
    append: yes/no                #If this option is set to yes, the last filled fast.
↪log-file will not be
                                  #overwritten while restarting Suricata.
```

The interval and several other options depend on the global stats section as described above.

### Syslog

With this option it is possible to send all alert and event output to syslog.

```
- syslog:                         #This is a output-module to direct log-output to␣
↪several directions.
    enabled: no                   #The use of this output-module is not enabled.
    facility: local5              #In this option you can set a syslog facility.
    level: Info                   #In this option you can set the level of output. The␣
↪possible levels are:
                                  #Emergency, Alert, Critical, Error, Warning, Notice,␣
↪Info and Debug.
```

### Drop.log, a line based information for dropped packets

**Note:** This output has been deprecated and will be removed by June 2020.

If Suricata works in IPS mode, it can drop packets based on rules. Packets that are being dropped are saved in the drop.log file, a Netfilter log format.

```
- drop:
    enabled: yes                #The option is enabled.
    filename: drop.log          #The log-name of the file for dropped packets.
    append: yes                 #If this option is set to yes, the last filled
                                #drop.log-file will not be overwritten while
                                #restarting Suricata. If set to 'no' the last
                                #filled drop.log file will be overwritten.
    filetype: regular           #regular, unis_stream, unix_dgram
```

#### File-store (File Extraction)

The *file-store* output enables storing of extracted files to disk and configures where they are stored.

The following shows the configuration options for version 2 of the *file-store* output.

```
- file-store:
    # This configures version 2 of the file-store.
    version: 2

    enabled: no

    # Set the directory for the filestore. If the path is not
    # absolute will be be relative to the default-log-dir.
    #dir: filestore

    # Write out a fileinfo record for each occurrence of a
    # file. Disabled by default as each occurrence is already logged
    # as a fileinfo record to the main eve-log.
    #write-fileinfo: yes

    # Force storing of all files. Default: no.
    #force-filestore: yes

    # Override the global stream-depth for sessions in which we want
    # to perform file extraction. Set to 0 for unlimited.
    #stream-depth: 0

    # Uncomment the following variable to define how many files can
    # remain open for filestore by Suricata. Default value is 0 which
    # means files get closed after each write
    #max-open-files: 1000

    # Force logging of checksums, available hash functions are md5,
    # sha1 and sha256. Note that SHA256 is automatically forced by
    # the use of this output module as it uses the SHA256 as the
    # file naming scheme.
    #force-hash: [sha1, md5]
```

### 10.1.9 Detection engine

### Inspection configuration

The detection-engine builds internal groups of signatures. Suricata loads signatures, with which the network traffic will be compared. The fact is, that many rules certainly will not be necessary. (For instance: if there appears a packet with the UDP-protocol, all signatures for the TCP-protocol won't be needed.) For that reason, all signatures will be divided in groups. However, a distribution containing many groups will make use of a lot of memory. Not every type of signature gets its own group. There is a possibility that different signatures with several properties in common, will be placed together in a group. The quantity of groups will determine the balance between memory and performance. A small amount of groups will lower the performance yet uses little memory. The opposite counts for a higher amount of groups. The engine allows you to manage the balance between memory and performance. To manage this, (by determining the amount of groups) there are several general options: high for good performance and more use of memory, low for low performance and little use of memory. The option medium is the balance between performance and memory usage. This is the default setting. The option custom is for advanced users. This option has values which can be managed by the user.

```
detect:
  profile: medium
  custom-values:
    toclient-groups: 2
    toserver-groups: 25
  sgh-mpm-context: auto
  inspection-recursion-limit: 3000
```

At all of these options, you can add (or change) a value. Most signatures have the adjustment to focus on one direction, meaning focusing exclusively on the server, or exclusively on the client.

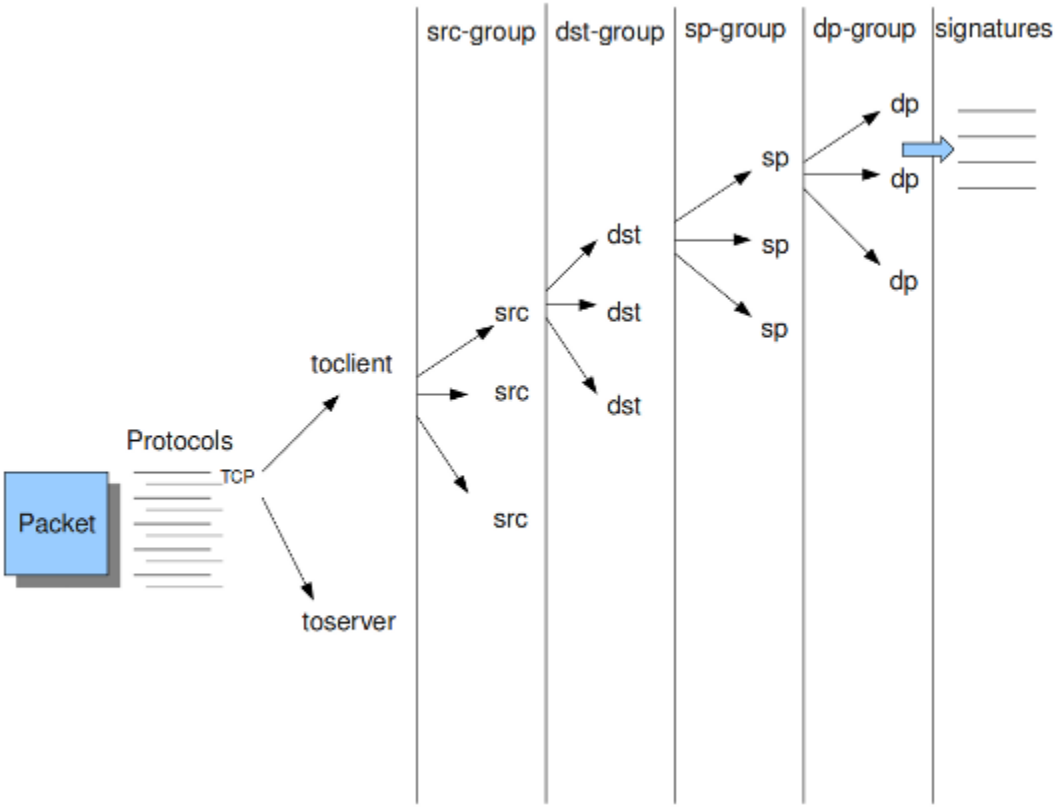
If you take a look at example 4, *the Detection-engine grouping tree*, you see it has many branches. At the end of each branch, there is actually a 'sig group head'. Within that sig group head there is a container which contains a list with signatures that are significant for that specific group/that specific end of the branch. Also within the sig group head the settings for Multi-Pattern-Matcher (MPM) can be found: the MPM-context.

As will be described again at the part 'Pattern matching settings', there are several MPM-algorithms of which can be chosen from. Because every sig group head has its own MPM-context, some algorithms use a lot of memory. For that reason there is the option sgh-mpm-context to set whether the groups share one MPM-context, or to set that every group has its own MPM-context.

For setting the option sgh-mpm-context, you can choose from auto, full or single. The default setting is 'auto', meaning Suricata selects full or single based on the algorithm you use. 'Full' means that every group has its own MPM-context, and 'single' that all groups share one MPM-context. The two algorithms ac and ac-gfbs are new in 1.03. These algorithms use a single MPM-context if the Sgh-MPM-context setting is 'auto'. The rest of the algorithms use full in that case.

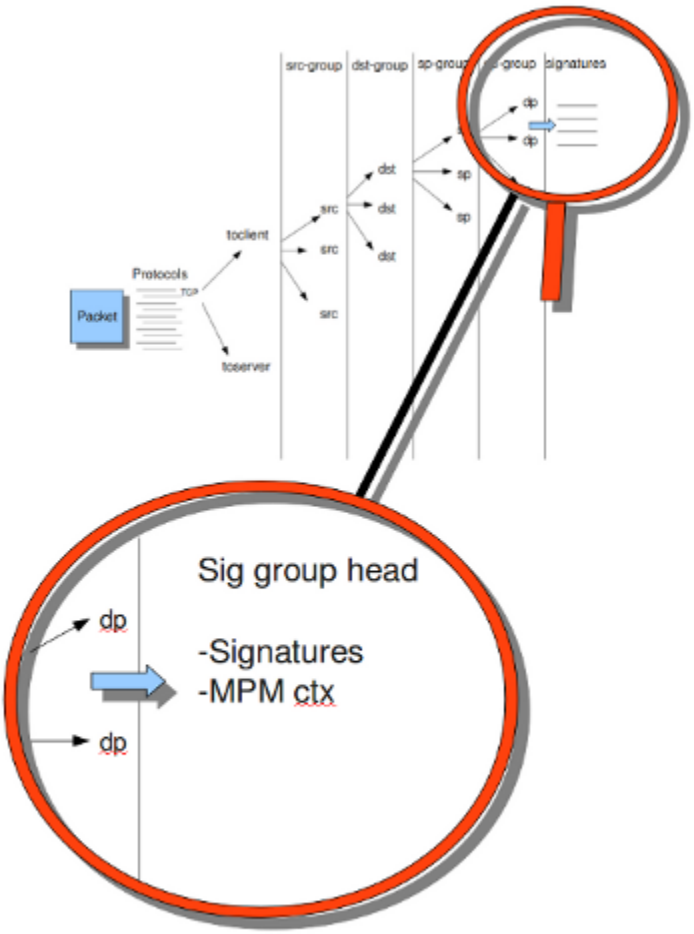
The inspection-recursion-limit option has to mitigate that possible bugs in Suricata cause big problems. Often Suricata has to deal with complicated issues. It could end up in an 'endless loop' due to a bug, meaning it will repeat its actions over and over again. With the option inspection-recursion-limit you can limit this action.

*Example 4 Detection-engine grouping tree*




```
src             Stands for source IP-address.
dst             Stands for destination IP-address.
sp              Stands for source port.
dp              Stands for destination port.
```

*Example 5 Detail grouping tree*




### Prefilter Engines

The concept of prefiltering is that there are far too many rules to inspect individually. The approach prefilter takes is that from each rule one condition is added to prefilter, which is then checked in one step. The most common example is MPM (also known as fast_pattern). This takes a single pattern per rule and adds it to the MPM. Only for those rules that have at least one pattern match in the MPM stage, individual inspection is performed.

Next to MPM, other types of keywords support prefiltering. ICMP itype, icode, icmp_seq and icmp_id for example. TCP window, IP TTL are other examples.

For a full list of keywords that support prefilter, see:

```
suricata --list-keywords=all
```

Suricata can automatically select prefilter options, or it can be set manually.

```
detect:
  prefilter:
    default: mpm
```

By default, only MPM/fast_pattern is used.

The prefilter engines for other non-MPM keywords can then be enabled in specific rules by using the 'prefilter' keyword.

E.g.

```
alert ip any any -> any any (ttl:123; prefilter; sid:1;)
```

To let Suricata make these decisions set default to 'auto':

```
detect:
  prefilter:
    default: auto
```

### Pattern matcher settings

The multi-pattern-matcher (MPM) is a part of the detection engine within Suricata that searches for multiple patterns at once. Generally, signatures have one ore more patterns. Of each signature, one pattern is used by the multi-pattern-matcher. That way Suricata can exclude many signatures from being examined, because a signature can only match when all its patterns match.

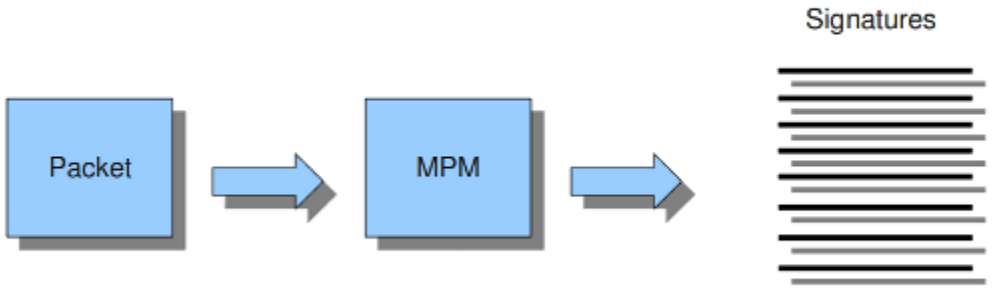
These are the proceedings:

1)A packet comes in.

**2)The packed will be analyzed by the Multi-pattern-matcher in search** of patterns that match.

3)All patterns that match, will be further processed by Suricata (signatures).

*Example 8 Multi-pattern-matcher*




Suricata offers various implementations of different multi-pattern-matcher algorithm's. These can be found below.

To set the multi-pattern-matcher algorithm:

```
mpm-algo: b2gc
```

After 'mpm-algo', you can enter one of the following algorithms: b2g, b2gc, b2gm, b3g, wumanber, ac and ac-gfbs (These last two are new in 1.0.3). For more information about these last two, please read again the the end of the part 'Detection engine'. These algorithms have no options, so the fact that below there is no option being mentioned is no omission.

Subsequently, you can set the options for the mpm-algorithm's.

The hash_size option determines the size of the hash-table that is internal used by the pattern matcher. A low hash-size (small table) causes lower memory usage, but decreases the performance. The opposite counts for a high hash-size: higher memory usage, but (generally) higher performance. The memory settings for hash size of the algorithms can vary from lowest (2048) - low (4096) - medium (8192) - high (16384) - higher (32768) – max (65536). (Higher is 'highest' in YAML 1.0 -1.0.2)

The bf_size option determines the size of the bloom filter, that is used with the final step of the pattern matcher, namely the validation of the pattern. For this option the same counts as for the hash-size option: setting it to low will cause lower memory usage, but lowers the performance. The opposite counts for a high setting of the bf_size: higher memory usage, but (generally) higher performance. The bloom-filter sizes can vary from low (512) - medium (1024) - high (2048).

```
pattern-matcher:
  - b2gc:
      search_algo: B2gSearchBNDMq
      hash_size: low                  #Determines the size of the hash-table.
      bf_size: medium                 #Determines the size of the bloom- filter.
  - b3g:
      search_algo: B3gSearchBNDMq
      hash_size: low                  #See hash-size -b2gc.
      bf_size: medium                 #See bf-size -b2gc.
  - wumanber:
      hash_size: low                  #See hash-size -b2gc.
      bf_size: medium                 #See bf-size -b2gc.
```

### 10.1.10 Threading

Suricata is multi-threaded. Suricata uses multiple CPU' s/CPU cores so it can process a lot of network packets simultaneously. (In a single-core engine, the packets will be processed one at a time.)

There are four thread-modules: Packet acquisition, decode and stream application layer, detection, and outputs.

# The packet acquisition module reads packets from the network.

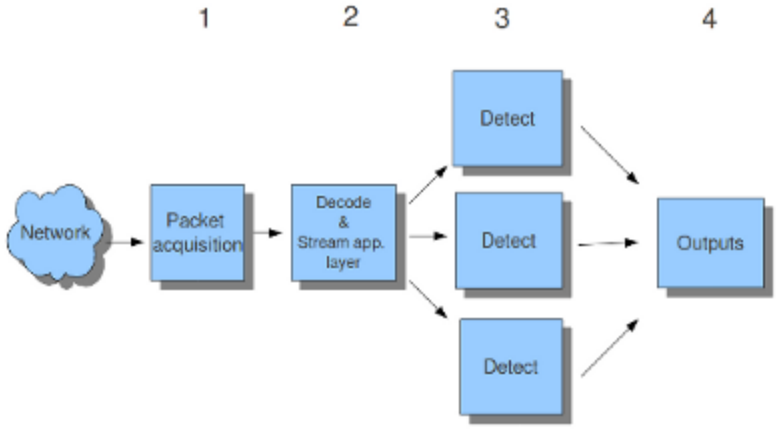
# The decode module decodes the packets and the stream application application layer has three tasks:

```
First: it performs stream-tracking, meaning it is making sure all steps will be taken␣
↪to make a correct network-connection.
Second: TCP-network traffic comes in as packets. The Stream-Assembly engine␣
↪reconstructs the original stream.
Finally: the application layer will be inspected. HTTP and DCERPC will be analyzed.
```

# The detection threads will compare signatures. There can be several detection threads so they can operate simultaneously.

# In Outputs all alerts and events will be processed.

*Example 6 Threading*




```
Packet acquisition:              Reads packets from the network
Decode:                          Decodes packets.
Stream app. Layer:               Performs stream-tracking and reassembly.
Detect:                          Compares signatures.
Outputs:                         Processes all events and alerts.
```

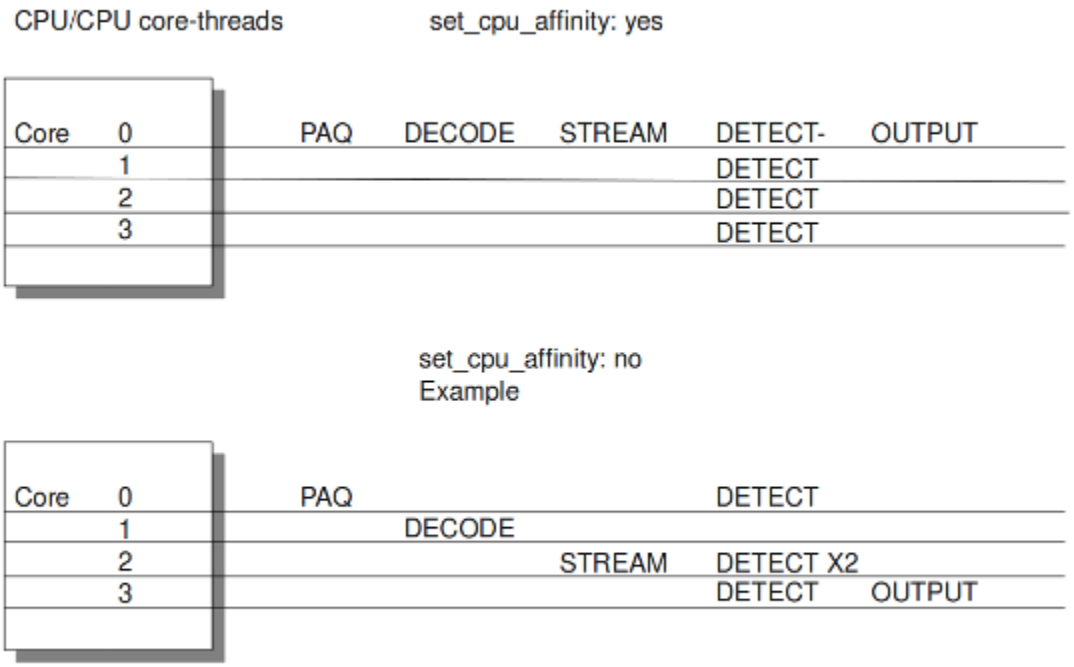
Most computers have multiple CPU's/ CPU cores. By default the operating system determines which core works on which thread. When a core is already occupied, another one will be designated to work on the thread. So, which core works on which thread, can differ from time to time.

There is an option within threading:

```
set-cpu-affinity: no
```

With this option you can cause Suricata setting fixed cores for every thread. In that case 1, 2 and 4 are at core 0 (zero). Each core has its own detect thread. The detect thread running on core 0 has a lower priority than the other threads running on core 0. If these other cores are to occupied, the detect thread on core 0 has not much packets to process. The detect threads running on other cores will process more packets. This is only the case after setting the option to 'yes'.

*Example 7 Balancing workload*




You can set the detect-thread-ratio:

```
detect-thread-ratio: 1.5
```

The detect thread-ratio will determine the amount of detect threads. By default it will be 1.5 x the amount of CPU's/CPU cores present at your computer. This will result in having more detection threads then CPU's/ CPU cores. Meaning you are oversubscribing the amount of cores. This may be convenient at times when there have to be waited for a detection thread. The remaining detection thread can become active.

In the option 'cpu affinity' you can set which CPU's/cores work on which thread. In this option there are several sets of threads. The management-, receive-, worker- and verdict-set. These are fixed names and can not be changed. For each set there are several options: cpu, mode, and prio. In the option 'cpu' you can set the numbers of the CPU's/cores which will run the threads from that set. You can set this option to 'all', use a range (0-3) or a comma separated list (0,1). The option 'mode' can be set to 'balanced' or 'exclusive'. When set to 'balanced', the individual threads can be processed by all cores set in the option 'cpu'. If the option 'mode' is set to 'exclusive', there will be fixed cores for each thread. As mentioned before, threads can have different priority's. In the option 'prio' you can set a priority for each thread. This priority can be low, medium, high or you can set the priority to 'default'. If you do not set a priority for a CPU, than the settings in 'default' will count. By default Suricata creates one 'detect' (worker) thread per available CPU/CPU core.

```
cpu-affinity:
  - management-cpu-set:
      cpu: [ 0 ]  # include only these cpus in affinity settings
  - receive-cpu-set:
      cpu: [ 0 ]  # include only these cpus in affinity settings
  - worker-cpu-set:
      cpu: [ "all" ]
      mode: "exclusive"
      # Use explicitely 3 threads and don't compute number by using
```

(continues on next page)

(continued from previous page)

```
      # detect-thread-ratio variable:
      # threads: 3
      prio:
        low: [ 0 ]
        medium: [ "1-2" ]
        high: [ 3 ]
        default: "medium"
  - verdict-cpu-set:
      cpu: [ 0 ]
      prio:
        default: "high"
```

#### Relevant cpu-affinity settings for IDS/IPS modes

##### IDS mode

Runmode AutoFp:

```
management-cpu-set - used for management (example - flow.managers, flow.recyclers)
receive-cpu-set - used for receive and decode
worker-cpu-set - used for streamtcp,detect,output(logging),reject
```

Rumode Workers:

```
management-cpu-set - used for management (example - flow.managers, flow.recyclers)
worker-cpu-set - used for receive,streamtcp,decode,detect,output(logging),respond/
↪reject
```

##### IPS mode

Runmode AutoFp:

```
management-cpu-set - used for management (example - flow.managers, flow.recyclers)
receive-cpu-set - used for receive and decode
worker-cpu-set - used for streamtcp,detect,output(logging)
verdict-cpu-set - used for verdict and respond/reject
```

Runmode Workers:

```
management-cpu-set - used for management (example - flow.managers, flow.recyclers)
worker-cpu-set - used for receive,streamtcp,decode,detect,output(logging),respond/
↪reject, verdict
```

### 10.1.11 IP Defrag

Occasionally network packets appear fragmented. On some networks it occurs more often than on others. Fragmented packets exist of many parts. Before Suricata is able to inspect these kind of packets accurately, the packets have to be reconstructed. This will be done by a component of Suricata; the defragment-engine. After a fragmented packet is reconstructed by the defragment-engine, the engine sends on the reassembled packet to rest of Suricata.

There are three options within defrag: max-frags, prealloc and timeout. At the moment Suricata receives a fragment of a packet, it keeps in memory that other fragments of that packet will appear soon to complete the packet. However,

there is a possibility that one of the fragments does not appear. To prevent Suricata for keeping waiting for that packet (thereby using memory) there is a timespan after which Suricata discards the fragments. This occurs by default after 60 seconds.

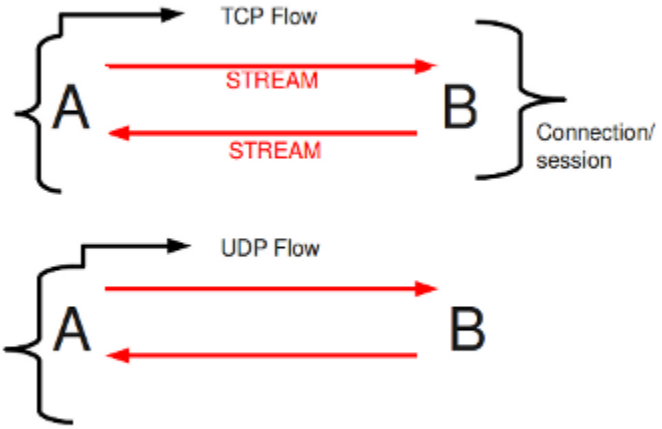
```
defrag:
  max-frags: 65535
  prealloc: yes
  timeout: 60
```

### 10.1.12 Flow and Stream handling

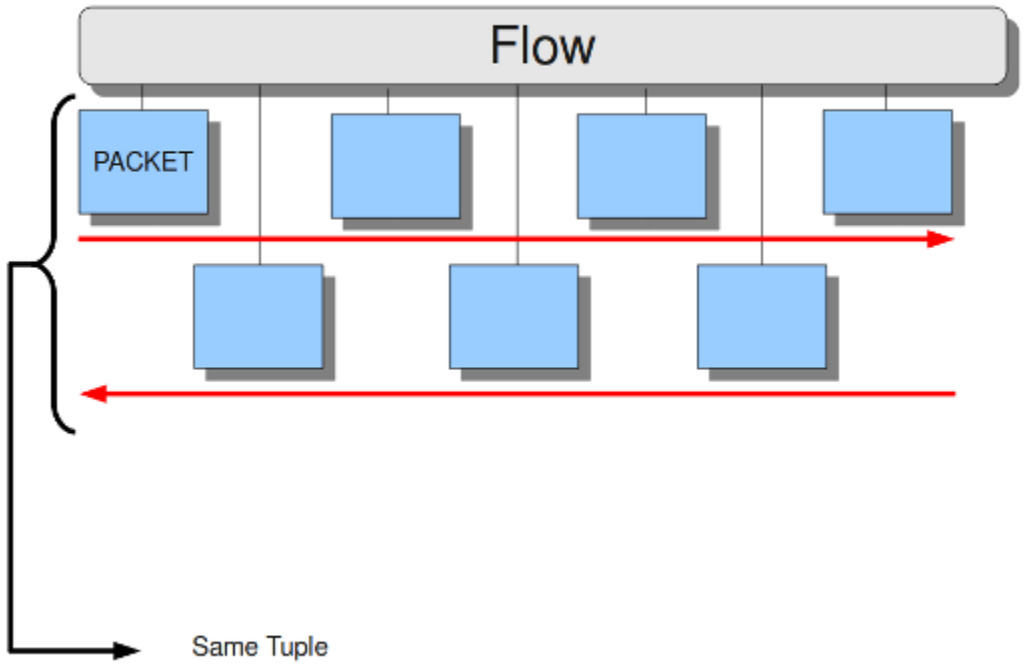
#### Flow Settings

Within Suricata, Flows are very important. They play a big part in the way Suricata organizes data internally. A flow is a bit similar to a connection, except a flow is more general. All packets having the same Tuple (protocol, source IP, destination IP, source-port, destination-port), belong to the same flow. Packets belonging to a flow are connected to it internally.

*Example 9 Flow*




*Example 10 Tuple*




Keeping track of all these flows, uses memory. The more flows, the more memory it will cost.

To keep control over memory usage, there are several options:

The option memcap for setting the maximum amount of bytes the flow-engine will use, hash-size for setting the size of the hash-table and prealloc for the following:

> For packets not yet belonging to a flow, Suricata creates a new flow. This is a relative expensive action. The risk coming with it, is that attackers /hackers can a attack the engine system at this part. When they make sure a computer gets a lot of packets with different tuples, the engine has to make a lot of new flows. This way, an attacker could flood the system. To mitigate the engine from being overloaded, this option instructs Suricata to keep a number of flows ready in memory. This way Suricata is less vulnerable to these kind of attacks.

The flow-engine has a management thread that operates independent from the packet processing. This thread is called the flow-manager. This thread ensures that wherever possible and within the memcap. there will be 10000 flows prepared.

```
flow:
  memcap: 33554432              #The maximum amount of bytes the flow-engine will␣
↪make use of.
  hash_size: 65536              #Flows will be organized in a hash-table. With this␣
↪option you can set the
                                #size of the hash-table.
  Prealloc: 10000               #The amount of flows Suricata has to keep ready in␣
↪memory.
```

At the point the memcap will still be reached, despite prealloc, the flow-engine goes into the emergency-mode. In this mode, the engine will make use of shorter time-outs. It lets flows expire in a more aggressive manner so there will be

more space for new Flows.

There are two options: emergency_recovery and prune_flows. The emergency recovery is set on 30. This is the percentage of prealloc'd flows after which the flow-engine will be back to normal (when 30 percent of the 10000 flows is completed).

> If during the emergency-mode, the aggressive time-outs do not have the desired result, this option is the final resort. It ends some flows even if they have not reached their time-outs yet. The prune-flows option shows how many flows there will be terminated at each time a new flow is set up.

```
emergency_recovery: 30                  #Percentage of 1000 prealloc'd flows.
prune_flows: 5                          #Amount of flows being terminated during the␣
↪emergency mode.
```

### Flow Time-Outs

The amount of time Suricata keeps a flow in memory is determined by the Flow time-out.

There are different states in which a flow can be. Suricata distinguishes three flow-states for TCP and two for UDP. For TCP, these are: New, Established and Closed,for UDP only new and established. For each of these states Suricata can employ different timeouts.

The state new in a TCP-flow, means the period during the three way handshake. The state established is the state when the three way handshake is completed. The state closed in the TCP-flow: there a several ways to end a flow. This is by means of Reset or the Four-way FIN handshake.

New in a UDP-flow: the state in which packets are send from only one direction.

Established in a UDP-flow: packets are send from both directions.

In the example configuration the are settings for each protocol. TCP, UDP, ICMP and default (all other protocols).

```
flow-timeouts:

  default:
    new: 30                     #Time-out in seconds after the last activity in this␣
↪flow in a New state.
    established: 300            #Time-out in seconds after the last activity in this␣
↪flow in a Established
                                #state.
    emergency_new: 10           #Time-out in seconds after the last activity in this␣
↪flow in a New state
                                #during the emergency mode.
    emergency_established: 100  #Time-out in seconds after the last activity in this␣
↪flow in a Established
                                #state in the emergency mode.
  tcp:
    new: 60
    established: 3600
    closed: 120
    emergency_new: 10
    emergency_established: 300
    emergency_closed: 20
  udp:
    new: 30
    established: 300
    emergency_new: 10
    emergency_established: 100
```

(continues on next page)

(continued from previous page)

```
  icmp:
    new: 30
    established: 300
    emergency_new: 10
    emergency_established: 100
```

### Stream-engine

The Stream-engine keeps track of the TCP-connections. The engine exists of two parts: The stream tracking- and the reassembly-engine.

The stream-tracking engine monitors the state of a connection. The reassembly-engine reconstructs the flow as it used to be, so it will be recognized by Suricata.

The stream-engine has two memcaps that can be set. One for the stream-tracking-engine and one for the reassembly-engine.

The stream-tracking-engine keeps information of the flow in memory. Information about the state, TCP-sequence-numbers and the TCP window. For keeping this information, it can make use of the capacity the memcap allows.

TCP packets have a so-called checksum. This is an internal code which makes it possible to see if a packet has arrived in a good state. The stream-engine will not process packets with a wrong checksum. This option can be set off by entering 'no' instead of 'yes'.

```
stream:
  memcap: 64mb                  # Max memory usage (in bytes) for TCP session tracking
  checksum_validation: yes      # Validate packet checksum, reject packets with invalid␣
↪checksums.
```

To mitigate Suricata from being overloaded by fast session creation, the option prealloc_sessions instructs Suricata to keep a number of sessions ready in memory.

A TCP-session starts with the three-way-handshake. After that, data can be send en received. A session can last a long time. It can happen that Suricata will be started after a few TCP sessions have already been started. This way, Suricata misses the original setup of those sessions. This setup always includes a lot of information. If you want Suricata to check the stream from that time on, you can do so by setting the option 'midstream' to 'true'. The default setting is 'false'. Normally Suricata is able to see all packets of a connection. Some networks make it more complicated though. Some of the network-traffic follows a different route than the other part, in other words: the traffic goes asynchronous. To make sure Suricata will check the one part it does see, instead of getting confused, the option 'async-oneside' is brought to life. By default the option is set to 'false'.

Suricata inspects content in the normal/IDS mode in chunks. In the inline/IPS mode it does that on the sliding window way (see example ..) In the case Suricata is set in inline mode, it has to inspect packets immediately before sending it to the receiver. This way Suricata is able to drop a packet directly if needed.(see example . . . ) It is important for Suricata to note which operating system it is dealing with, because operating systems differ in the way they process anomalies in streams. See *Host-os-policy*.

```
prealloc_sessions: 32768      # 32k sessions prealloc'd
midstream: false              # do not allow midstream session pickups
async_oneside: false          # do not enable async stream handling
inline: no                    # stream inline mode
drop-invalid: yes             # drop invalid packets
bypass: no
```

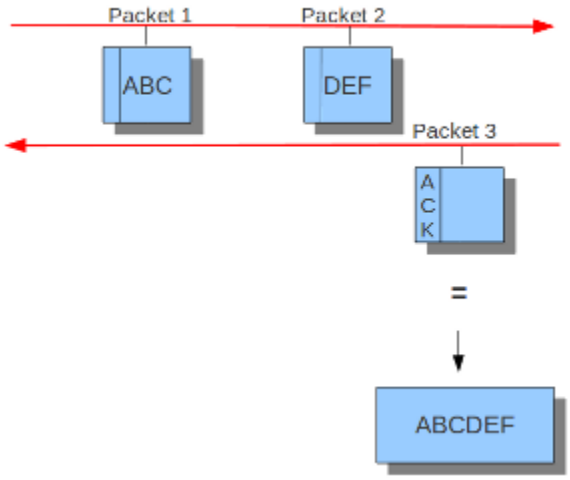
The `drop-invalid` option can be set to no to avoid blocking packets that are seen invalid by the streaming engine. This can be useful to cover some weird cases seen in some layer 2 IPS setup.

The `bypass` option activates 'bypass' for a flow/session when either side of the session reaches its `depth`.

> **Warning:** `bypass` can lead to missing important traffic. Use with care.

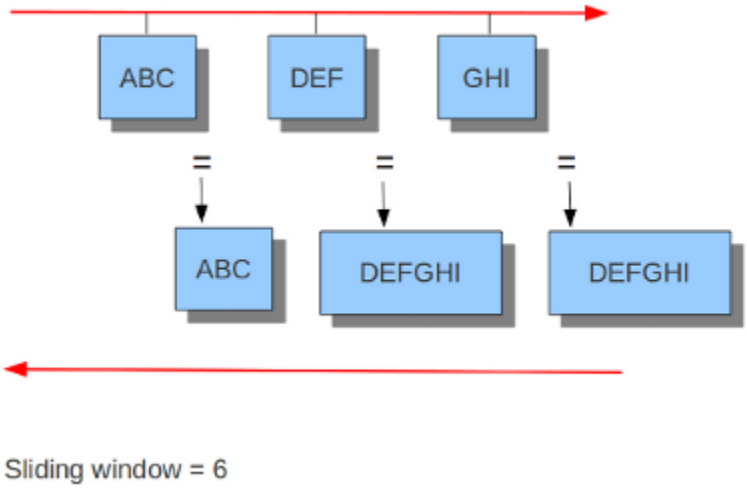
**Example 11 Normal/IDS mode**

Suricata inspects traffic in chunks.

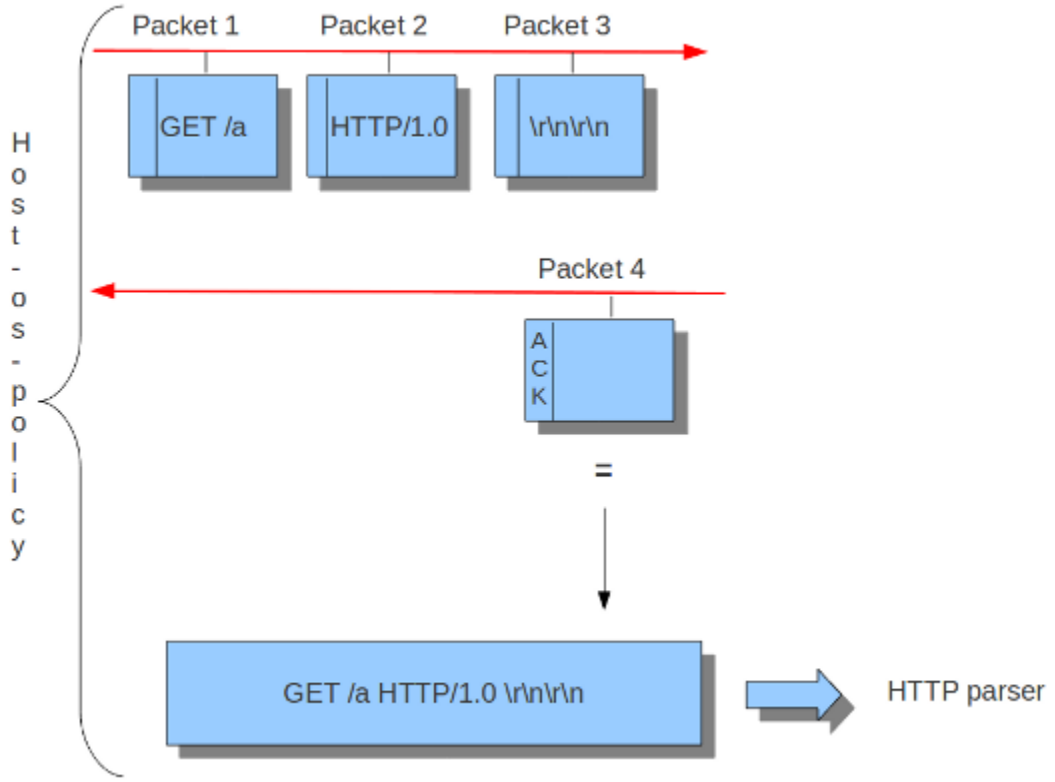



**Example 12 Inline/IPS Sliding Window**

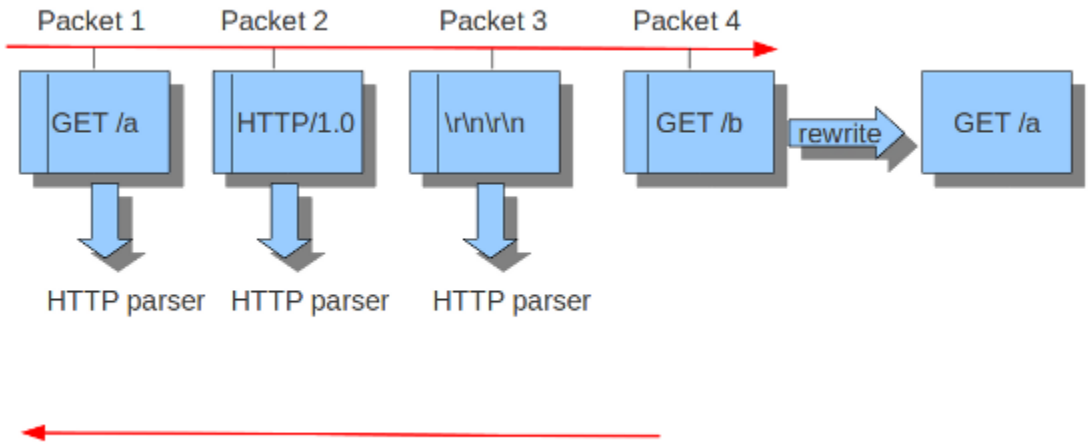
Suricata inspects traffic in a sliding window manner.




**Example 13 Normal/IDS (reasembly on ACK'D data)**




**Example 14 Inline/IPS (reassembly on UNACK'D data)**




The reassembly-engine has to keep data segments in memory in order to be able to reconstruct a stream. To avoid resource starvation a memcap is used to limit the memory used.

Reassembling a stream is an expensive operation. With the option depth you can control how far into a stream reassembly is done. By default this is 1MB. This setting can be overridden per stream by the protocol parsers that do file extraction.

Inspection of reassembled data is done in chunks. The size of these chunks is set with `toserver_chunk_size` and

`toclient_chunk_size`. To avoid making the borders predictable, the sizes van be varied by adding in a random factor.

```
reassembly:
  memcap: 256mb             # Memory reserved for stream data reconstruction (in␣
↪bytes)
  depth: 1mb                # The depth of the reassembling.
  toserver_chunk_size: 2560 # inspect raw stream in chunks of at least this size
  toclient_chunk_size: 2560 # inspect raw stream in chunks of at least
  randomize-chunk-size: yes
  #randomize-chunk-range: 10
```

'Raw' reassembly is done for inspection by simple `content`, `pcre` keywords use and other payload inspection not done on specific protocol buffers like `http_uri`. This type of reassembly can be turned off:

```
reassembly:
  raw: no
```

Incoming segments are stored in a list in the stream. To avoid constant memory allocations a per-thread pool is used.

```
reassembly:
  segment-prealloc: 2048    # pre-alloc 2k segments per thread
```

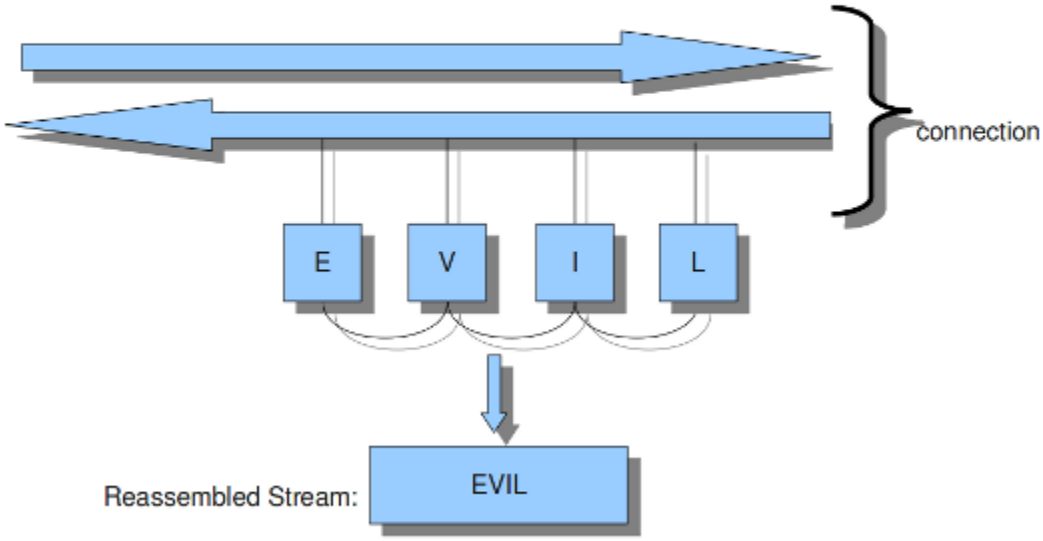
Resending different data on the same sequence number is a way to confuse network inspection.

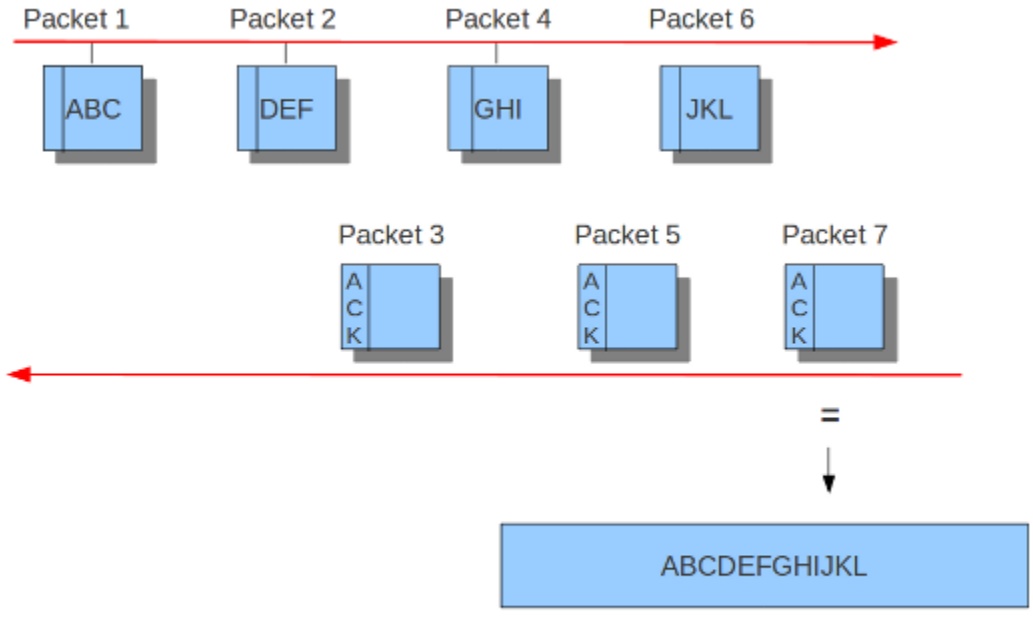
```
reassembly:
  check-overlap-different-data: true
```

*Example 15 Stream reassembly*

Stream Reassembly

Signature: EVIL




toserver_chunk_size: 10

### 10.1.13 Application Layer Parsers

#### Asn1_max_frames (new in 1.0.3 and 1.1)

Asn1 (Abstract Syntax One) is a standard notation to structure and describe data.

Within Asn1_max_frames there are several frames. To protect itself, Suricata will inspect a maximum of 256. You can set this amount differently if wanted.

Application layer protocols such as X.400 electronic mail, X.500 and LDAP directory services, H.323 (VoIP), BACnet and SNMP, use ASN.1 to describe the protocol data units (PDUs) they exchange. It is also extensively used in the Access and Non-Access Strata of UMTS.

Limit for the maximum number of asn1 frames to decode (default 256):

```
asn1_max_frames: 256
```

#### Configure HTTP (libhtp)

The library Libhtp is being used by Suricata to parse HTTP-sessions.

While processing HTTP-traffic, Suricata has to deal with different kind of servers which each process anomalies in HTTP-traffic differently. The most common web-server is Apache. This is an open source web-server program.

Besides Apache, IIS (Internet Information Services/Server) a web-server program of Microsoft is also well-known.

Like with host-os-policy, it is important for Suricata to know which IP-address/network-address is used by which server. In Libhtp this assigning of web-servers to IP-and network addresses is called personality.

Currently Available Personalities:

- Minimal
- Generic
- IDS (default)
- IIS_4_0
- IIS_5_0
- IIS_5_1
- IIS_6_0
- IIS_7_0
- IIS_7_5
- Apache
- Apache_2_2

You can assign names to each block of settings. Which in this case is -apache and -iis7. Under these names you can set IP-addresses, network-addresses the personality and a set of features.

The version-specific personalities know exactly how web servers behave, and emulate that. The IDS personality would try to implement a best-effort approach that would work reasonably well in the cases where you do not know the specifics.

The default configuration also applies to every IP-address for which no specific setting is available.

HTTP request bodies are often big, so they take a lot of time to process which has a significant impact on the performance. With the option 'request-body-limit' you can set the limit (in bytes) of the client-body that will be inspected. Setting it to 0 will inspect all of the body.

The same goes for HTTP response bodies.

```
libhtp:

  default-config:
    personality: IDS
    request-body-limit: 3072
    response-body-limit: 3072

  server-config:
     - apache:
         address: [192.168.1.0/24, 127.0.0.0/8, "::1"]
         personality: Apache_2_2
         request-body-limit: 0
         response-body-limit: 0

     - iis7:
         address:
           - 192.168.0.0/24
           - 192.168.10.0/24
         personality: IIS_7_0
         request-body-limit: 4096
         response-body-limit: 8192
```

Suricata makes available the whole set of libhtp customisations for its users.

You can now use these parameters in the conf to customise suricata's use of libhtp.

```
# Configures whether backslash characters are treated as path segment
# separators. They are not on Unix systems, but are on Windows systems.
# If this setting is enabled, a path such as "/one\two/three" will be
# converted to "/one/two/three".  Accepted values - yes, no.
#path-convert-backslash-separators: yes

# Configures whether input data will be converted to lowercase.
#path-convert-lowercase: yes

# Configures how the server reacts to encoded NUL bytes.
#path-nul-encoded-terminates: no

# Configures how the server reacts to raw NUL bytes.
#path-nul-raw-terminates: no

# Configures whether consecutive path segment separators will be
# compressed. When enabled, a path such as "/one//two" will be normalized
# to "/one/two". The backslash_separators and decode_separators
# parameters are used before compression takes place. For example, if
# backslash_separators and decode_separators are both enabled, the path
# "/one\\/two\/%5cthree/%2f//four" will be converted to
# "/one/two/three/four".  Accepted values - yes, no.
#path-separators-compress: yes

# Configures whether encoded path segment separators will be decoded.
# Apache does not do this, but IIS does. If enabled, a path such as
# "/one%2ftwo" will be normalized to "/one/two". If the
```

(continues on next page)

(continued from previous page)

```
# backslash_separators option is also enabled, encoded backslash
# characters will be converted too (and subsequently normalized to
# forward slashes).  Accepted values - yes, no.
#path-separators-decode: yes

# Configures whether %u-encoded sequences in path will be decoded. Such
# sequences will be treated as invalid URL encoding if decoding is not
# desireable.  Accepted values - yes, no.
#path-u-encoding-decode: yes

# Configures how server reacts to invalid encoding in path.  Accepted
# values - preserve_percent, remove_percent, decode_invalid, status_400
#path-url-encoding-invalid-handling: preserve_percent

# Controls whether the data should be treated as UTF-8 and converted
# to a single-byte stream using best-fit mapping
#path-utf8-convert-bestfit:yes

# Sets the replacement character that will be used to in the lossy
# best-fit mapping from Unicode characters into single-byte streams.
# The question mark is the default replacement character.
#path-bestfit-replacement-char: ?

# Configures whether plus characters are converted to spaces
# when decoding URL-encoded strings.
#query-plusspace-decode: yes

#   response-body-decompress-layer-limit:
#                           Limit to how many layers of compression will be
#                           decompressed. Defaults to 2.

#   uri-include-all:        Include all parts of the URI. By default the
#                           'scheme', username/password, hostname and port
#                           are excluded.

#   meta-field-limit:       Hard size limit for request and response size
#                           limits.

# inspection limits
    request-body-minimal-inspect-size: 32kb
    request-body-inspect-window: 4kb
    response-body-minimal-inspect-size: 40kb
    response-body-inspect-window: 16kb

# auto will use http-body-inline mode in IPS mode, yes or no set it statically
    http-body-inline: auto

# Decompress SWF files.
# 2 types: 'deflate', 'lzma', 'both' will decompress deflate and lzma
# compress-depth:
# Specifies the maximum amount of data to decompress,
# set 0 for unlimited.
# decompress-depth:
# Specifies the maximum amount of decompressed data to obtain,
# set 0 for unlimited.
    swf-decompression:
      enabled: yes
```

(continues on next page)

(continued from previous page)

```
      type: both
      compress-depth: 0
      decompress-depth: 0

# Take a random value for inspection sizes around the specified value.
# This lower the risk of some evasion technics but could lead
# detection change between runs. It is set to 'yes' by default.
#randomize-inspection-sizes: yes
# If randomize-inspection-sizes is active, the value of various
# inspection size will be choosen in the [1 - range%, 1 + range%]
# range
# Default value of randomize-inspection-range is 10.
#randomize-inspection-range: 10

# Can disable LZMA decompression
#lzma-enabled: yes
# Memory limit usage for LZMA decompression dictionary
# Data is decompressed until dictionary reaches this size
#lzma-memlimit: 1 Mb
# Maximum decompressed size with a compression ratio
# above 2048 (only reachable by LZMA)
#compression-bomb-limit: 1 Mb
```

Other parameters are customizable from Suricata.

```
#   double-decode-path:     Double decode path section of the URI
#   double-decode-query:    Double decode query section of the URI
```

#### Configure SMB (Rust)

**Note:** for full SMB support compile Suricata with Rust support

The SMB parser will parse version 1, 2 and 3 of the SMB protocol over TCP.

To enable the parser add the following to the `app-layer` section of the YAML.

```
smb:
  enabled: yes
  detection-ports:
    dp: 139, 445
```

The parser uses pattern based protocol detection and will fallback to `probing parsers` if the pattern based detection fails. As usual, the pattern based detection is port independent. The `probing parsers` will only run on the `detection-ports`.

SMB is commonly used to transfer the DCERPC protocol. This traffic is also handled by this parser.

### 10.1.14 Engine Logging

The engine logging system logs information about the application such as errors and other diagnostic information during startup, runtime and shutdown of the Suricata engine. This does not include Suricata generated alerts and events.

The engine logging system has the following log levels:

- error
- warning
- notice
- info
- perf
- config
- debug

Note that debug level logging will only be emitted if Suricata was compiled with the `--enable-debug` configure option.

The first option within the logging configuration is the default-log-level. This option determines the severity/importance level of information that will be displayed. Messages of lower levels than the one set here, will not be shown. The default setting is Info. This means that error, warning and info will be shown and the other levels won't be.

### Default Configuration Example

```
# Logging configuration.  This is not about logging IDS alerts/events, but
# output about what Suricata is doing, like startup messages, errors, etc.
logging:
  # The default log level, can be overridden in an output section.
  # Note that debug level logging will only be emitted if Suricata was
  # compiled with the --enable-debug configure option.
  #
  # This value is overridden by the SC_LOG_LEVEL env var.
  default-log-level: notice

  # The default output format.  Optional parameter, should default to
  # something reasonable if not provided.  Can be overridden in an
  # output section.  You can leave this out to get the default.
  #
  # This value is overridden by the SC_LOG_FORMAT env var.
  #default-log-format: "[%i] %t - (%f:%l) <%d> (%n) -- "

  # A regex to filter output.  Can be overridden in an output section.
  # Defaults to empty (no filter).
  #
  # This value is overridden by the SC_LOG_OP_FILTER env var.
  default-output-filter:

  # Define your logging outputs.  If none are defined, or they are all
  # disabled you will get the default - console output.
  outputs:
  - console:
      enabled: yes
      # type: json
  - file:
      enabled: yes
      level: info
      filename: suricata.log
      # type: json
  - syslog:
```

(continues on next page)

(continued from previous page)

```
      enabled: no
      facility: local5
      format: "[%i] <%d> -- "
      # type: json
```

### Default Log Level

Example:

```
logging:
  default-log-level: info
```

This option sets the default log level. The default log level is *notice*. This value will be used in the individual logging configuration (console, file, syslog) if not otherwise set.

---

**Note:** The `-v` command line option can be used to quickly increase the log level at runtime. See *the -v command line option*.

---

The `default-log-level` set in the configuration value can be overriden by the `SC_LOG_LEVEL` environment variable.

### Default Log Format

A logging line exists of two parts. First it displays meta information (thread id, date etc.), and finally the actual log message. Example:

```
[27708] 15/10/2010 -- 11:40:07 - (suricata.c:425) <Info> (main) - This is Suricata␣
↪version 1.0.2
```

(Here the part until the – is the meta info, "This is Suricata 1.0.2" is the actual message.)

It is possible to determine which information will be displayed in this line and (the manner how it will be displayed) in which format it will be displayed. This option is the so called format string:

```
default-log-format: "[%i] %t - (%f:%l) <%d> (%n) -- "
```

The % followed by a character, has a special meaning. There are eight specified signs:

```
t:      Time, timestamp, time and date
                  example: 15/10/2010 - -11:40:07
p:      Process ID. Suricata's whole processing consists of multiple threads.
i:      Thread ID. ID of individual threads.
m:      Thread module name. (Outputs, Detect etc.)
d:      Log-level of specific log-event. (Error, info, debug etc.)
f:      Filename. Name of C-file (source code) where log-event is generated.
l:      Line-number within the filename, where the log-event is generated in the␣
↪source-code.
n:      Function-name in the C-code (source code).
```

The last three, f, l and n are mainly convenient for developers.

The log-format can be overridden in the command line by the environment variable: SC_LOG_FORMAT

#### Output Filter

Within logging you can set an output-filter. With this output-filter you can set which part of the event-logs should be displayed. You can supply a regular expression (Regex). A line will be shown if the regex matches.

```
default-output-filter:                #In this option the regular expression can be␣
↪entered.
```

This value is overridden by the environment var: SC_LOG_OP_FILTER

#### Logging Outputs

There are different ways of displaying output. The output can appear directly on your screen, it can be placed in a file or via syslog. The last mentioned is an advanced tool for log-management. The tool can be used to direct log-output to different locations (files, other computers etc.)

```
outputs:
  - console:                                    #Output on your screen.
      enabled: yes                              #This option is enabled.
      #level: notice                            #Use a different level than the␣
↪default.
  - file:                                       #Output stored in a file.
      enabled: no                               #This option is not enabled.
      filename: /var/log/suricata.log           #Filename and location on disc.
      level: info                               #Use a different level than the␣
↪default.
  - syslog:                                     #This is a program to direct log-
↪output to several directions.
      enabled: no                               #The use of this program is not␣
↪enabled.
      facility: local5                          #In this option you can set a syslog␣
↪facility.
      format: "[%i] <%d> -- "                   #The option to set your own format.
      #level: notice                            #Use a different level than the␣
↪default.
```

### 10.1.15 Packet Acquisition

#### Pf-ring

The Pf_ring is a library that aims to improve packet capture performance over libcap. It performs packet acquisition. There are three options within Pf_ring: interface, cluster-id and cluster-type.

```
pfring:
  interface: eth0     # In this option you can set the network-interface
                      # on which you want the packets of the network to be read.
```

Pf_ring will load balance packets based on flow. All packet acquisition threads that will participate in the load balancing need to have the same cluster-id. It is important to make sure this ID is unique for this cluster of threads, so that no other engine / program is making use of clusters with the same id.

```
cluster-id: 99
```

Pf_ring can load balance traffic using pf_ring-clusters. All traffic for pf_ring can be load balanced in one of two ways, in a round robin manner or a per flow manner that are part of the same cluster. All traffic for pf_ring will be load balanced across acquisition threads of the same cluster id.

The cluster_round_robin manner is a way of distributing packets one at a time to each thread (like distributing playing cards to fellow players). The cluster_flow manner is a way of distributing all packets of the same flow to the same thread. The flows itself will be distributed to the threads in a round-robin manner.

```
cluster-type: cluster_round_robin
```

### NFQ

Using NFQUEUE in iptables rules, will send packets to Suricata. If the mode is set to 'accept', the packet that has been send to Suricata by a rule using NFQ, will by default not be inspected by the rest of the iptables rules after being processed by Suricata. There are a few more options to NFQ to change this if desired.

If the mode is set to 'repeat', the packets will be marked by Suricata and be re-injected at the first rule of iptables. To mitigate the packet from being going round in circles, the rule using NFQ will be skipped because of the mark.

If the mode is set to 'route', you can make sure the packet will be send to another tool after being processed by Suricata. It is possible to assign this tool at the mandatory option 'route_queue'. Every engine/tool is linked to a queue-number. This number you can add to the NFQ rule and to the route_queue option.

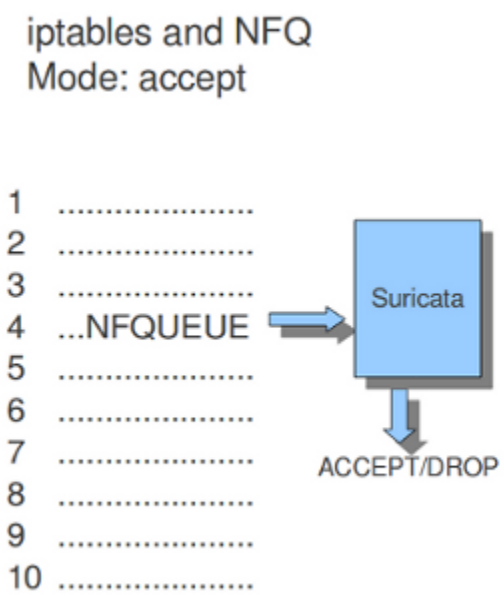
Add the numbers of the options repeat_mark and route_queue to the NFQ-rule:

```
iptables -I FORWARD -m mark ! --mark $MARK/$MASK -j NFQUEUE
```

```
nfq:
   mode: accept                 #By default the packet will be accepted or dropped by␣
↪Suricata
   repeat_mark: 1               #If the mode is set to 'repeat', the packets will be␣
↪marked after being
                                #processed by Suricata.
   repeat_mask: 1
   route_queue: 2               #Here you can assign the queue-number of the tool␣
↪that Suricata has to
                                #send the packets to after processing them.
```

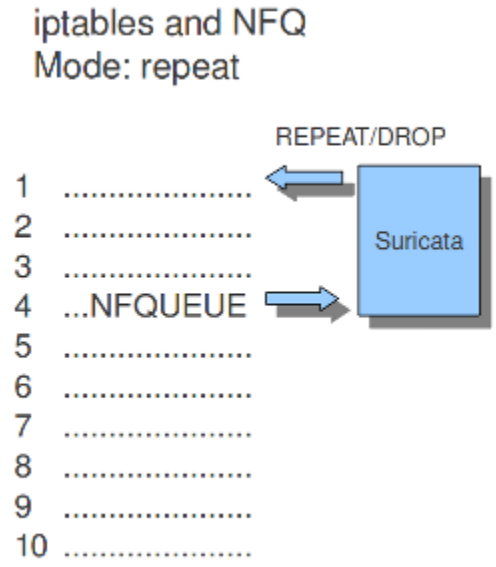
*Example 1 NFQ1*

mode: accept



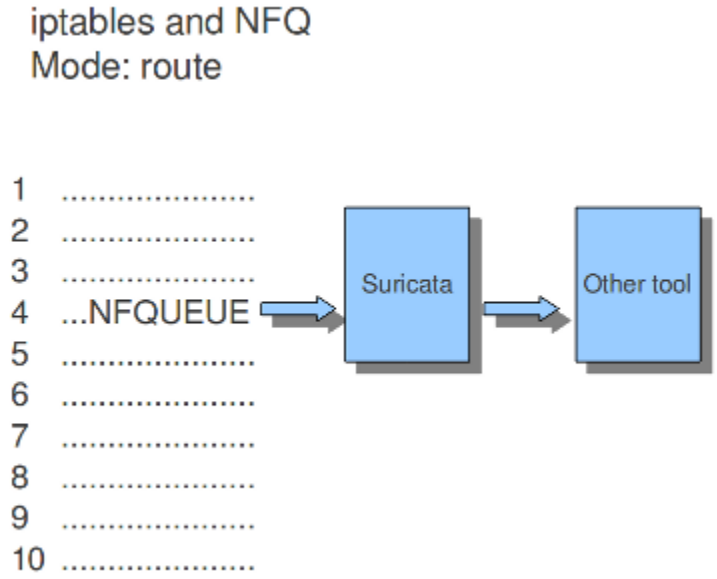

*Example 2 NFQ*

mode: repeat




*Example 3 NFQ*

mode: route




### Ipfw

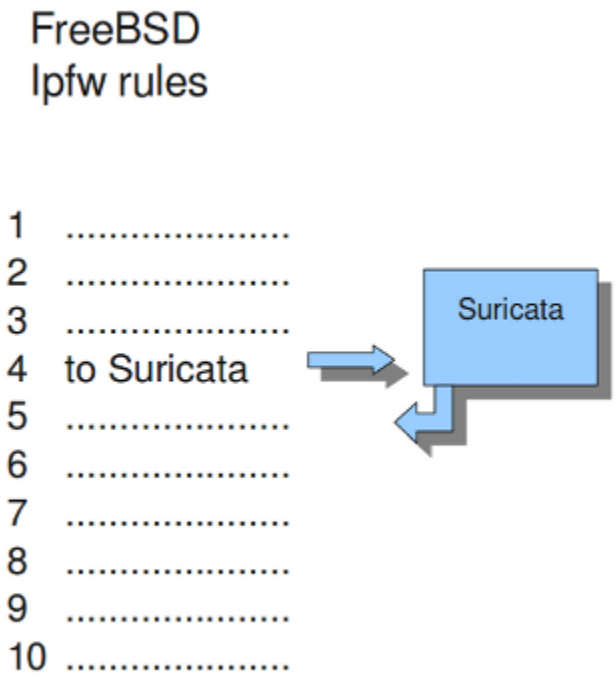
Suricata does not only support Linux, it supports the FreeBSD operating system (this is an open source Unix operating system) and Mac OS X as well. The in-line mode on FreeBSD uses ipfw (IP-firewall).

Certain rules in ipfw send network-traffic to Suricata. Rules have numbers. In this option you can set the rule to which the network-traffic will be placed back. Make sure this rule comes after the one that sends the traffic to Suricata, otherwise it will go around in circles.

The following tells the engine to re-inject packets back into the ipfw firewall at rule number 5500:

```
ipfw:
  ipfw-reinjection-rule-number: 5500
```

*Example 16 Ipfw-reinjection.*




### 10.1.16 Rules

#### Rule-files

For different categories of risk there are different rule-files available containing one or more rules. There is a possibility to instruct Suricata where to find these rules and which rules you want to be load for use. You can set the directory where the files can be found.

```
default-rule-path: /etc/suricata/rules/
rule-files:
  - backdoor.rules
  - bad-traffic.rules
  - chat.rules
  - ddos.rules
  - ....
```

The above mentioned is an example of rule-files of which can be chosen from. There are much more rule-files available.

If wanted, you can set a full path for a specific rule or rule-file. In that case, the above directory (/etc/suricata/rules/) will be ignored for that specific file. This is convenient in case you write your own rules and want to store them separate from other rules like that of VRT, ET or ET pro.

If you set a file-name that appears to be not existing, Suricata will ignore that entry and display a error-message during the engine startup. It will continue with the startup as usual.

### Threshold-file

Within this option, you can state the directory in which the threshold-file will be stored. The default directory is: /etc/suricata/threshold.config

### Classifications

The Classification-file is a file which makes the purpose of rules clear.

Some rules are just for providing information. Some of them are to warn you for serious risks like when you are being hacked etc.

In this classification-file, there is a part submitted to the rule to make it possible for the system-administrator to distinguish events.

A rule in this file exists of three parts: the short name, a description and the priority of the rule (in which 1 has the highest priority and 4 the lowest).

You can notice these descriptions returning in the rule and events / alerts.

```
Example:

configuration classification: misc-activity,Misc activity,3

Rule:

alert tcp $HOME_NET 21 -> $EXTERNAL_NET any (msg:"ET POLICY FTP Login Successful (non-
↪anonymous)";
flow:from_server,established;flowbits:isset,ET.ftp.user.login; flowbits:isnotset,ftp.
↪user.logged_in;
flowbits:set,ftp.user.logged_in; content:"230 ";pcre:!"/^230(\s+USER)?
↪\s+(anonymous|ftp)/smi";
classtype:misc-activity; reference:urldoc.emergingthreats.net/2003410,;
reference:url,www.emergingthreats.net/cgi-bin/cvsweb.cgi/sigs/POLICY/POLICY_FTP_Login;
↪ sid:2003410; rev:7;)

Event/Alert:

10/26/10-10:13:42.904785  [**] [1:2003410:7] ET POLICY FTP Login Successful (non-
↪anonymous) [**]
 [Classification: Misc activity[Priority: 3] {TCP} 192.168.0.109:21 -> x.x.x.x:34117
```

You can set the direction of the classification configuration.

```
classification-file: /etc/suricata/classification.config
```

### Rule-vars

There are variables which can be used in rules.

Within rules, there is a possibility to set for which IP-address the rule should be checked and for which IP-address it should not.

This way, only relevant rules will be used. To prevent you from having to set this rule by rule, there is an option in which you can set the relevant IP-address for several rules. This option contains the address group vars that will be passed in a rule. So, after HOME_NET you can enter your home IP-address.

```
vars:
  address-groups:
    HOME_NET: "[192.168.0.0/16,10.0.0.0/8,172.16.0.0/12]"        #By using [], it is␣
↪possible to set
                                                                 #complicated␣
↪variables.
    EXTERNAL_NET: any
    HTTP_SERVERS: "$HOME_NET"                                    #The $-sign tells␣
↪that what follows is
                                                                 #a variable.
    SMTP_SERVERS: "$HOME_NET"
    SQL_SERVERS: "$HOME_NET"
    DNS_SERVERS: "$HOME_NET"
    TELNET_SERVERS: "$HOME_NET"
    AIM_SERVERS: any
```

It is a convention to use upper-case characters.

There are two kinds of variables: Address groups and Port-groups. They both have the same function: change the rule so it will be relevant to your needs.

In a rule there is a part assigned to the address and one to the port. Both have their variable.

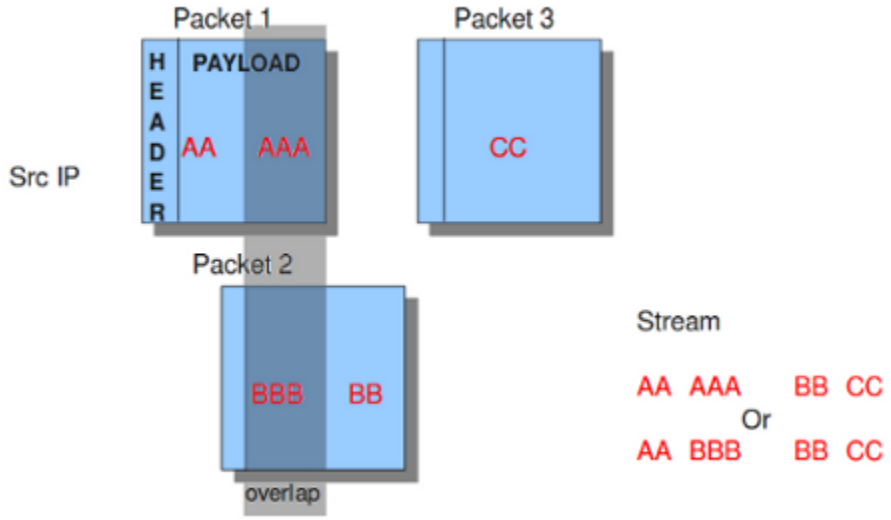
All options have to be set. If it is not necessary to set a specific address, you should enter 'any'.

```
port-groups:
  HTTP_PORTS: "80"
  SHELLCODE_PORTS: "!80"
  ORACLE_PORTS: 1521
  SSH_PORTS: 22
```

### Host-os-policy

Operating systems differ in the way they process fragmented packets and streams. Suricata performs differently with anomalies for different operating systems. It is important to set of which operating system your IP-address makes use of, so Suricata knows how to process fragmented packets and streams. For example in stream-reassembly there can be packets with overlapping payloads.

*Example 17 Overlapping payloads*




In the configuration-file, the operating-systems are listed. You can add your IP-address behind the name of the operating system you make use of.

```
host-os-policy:
  windows: [0.0.0.0/0]
  bsd: []
  bsd_right: []
  old_linux: []
  linux: [10.0.0.0/8, 192.168.1.100, "8762:2352:6241:7245:E000:0000:0000:0000"]
  old_solaris: []
  solaris: ["::1"]
  hpux10: []
  hpux11: []
  irix: []
  macos: []
  vista: []
  windows2k3: []
```

### 10.1.17 Engine analysis and profiling

Suricata offers several ways of analyzing performance of rules and the engine itself.

#### Engine-analysis

The option engine-analysis provides information for signature writers about how Suricata organizes signatures internally.

Like mentioned before, signatures have zero or more patterns on which they can match. Only one of these patterns will be used by the multi pattern matcher (MPM). Suricata determines which patterns will be used unless the fast-pattern rule option is used.

The option engine-analysis creates a new log file in the default log dir. In this file all information about signatures and patterns can be found so signature writers are able to see which pattern is used and change it if desired.

To create this log file, you have to run Suricata with ./src/suricata -c suricata.yaml –engine-analysis.

```
engine-analysis:
   rules-fast-pattern: yes
```

Example:

```
[10703] 26/11/2010 -- 11:41:15 - (detect.c:560) <Info> (SigLoadSignatures)
-- Engine-Analysis for fast_pattern printed to file - /var/log/suricata/rules_fast_
↪pattern.txt

== Sid: 1292 ==
Fast pattern matcher: content
Fast pattern set: no
Fast pattern only set: no
Fast pattern chop set: no
Content negated: no
Original content: Volume Serial Number
Final content: Volume Serial Number

---

alert tcp any any -> any any (content:"abc"; content:"defghi"; sid:1;)

== Sid: 1 ==
Fast pattern matcher: content
Fast pattern set: no
Fast pattern only set: no
Fast pattern chop set: no
Content negated: no
Original content: defghi
Final content: defghi

---

alert tcp any any -> any any (content:"abc"; fast_pattern:only; content:"defghi";␣
↪sid:1;)

== Sid: 1 ==
Fast pattern matcher: content
Fast pattern set: yes
Fast pattern only set: yes
Fast pattern chop set: no
Content negated: no
Original content: abc
Final content: abc

---

alert tcp any any -> any any (content:"abc"; fast_pattern; content:"defghi"; sid:1;)

== Sid: 1 ==
Fast pattern matcher: content
Fast pattern set: yes
Fast pattern only set: no
Fast pattern chop set: no
Content negated: no
Original content: abc
```

(continues on next page)

(continued from previous page)

```
Final content: abc

---

alert tcp any any -> any any (content:"abc"; fast_pattern:1,2; content:"defghi";␣
↪sid:1;)

== Sid: 1 ==
Fast pattern matcher: content
Fast pattern set: yes
Fast pattern only set: no
Fast pattern chop set: yes
Fast pattern offset, length: 1, 2
Content negated: no
Original content: abc
Final content: bc
```

### Rule and Packet Profiling settings

Rule profiling is a part of Suricata to determine how expensive rules are. Some rules are very expensive while inspecting traffic. Rule profiling is convenient for people trying to track performance problems and resolving them. Also for people writing signatures.

Compiling Suricata with rule-profiling will have an impact on performance, even if the option is disabled in the configuration file.

To observe the rule-performance, there are several options.

```
profiling:
  rules:
    enabled: yes
```

This engine is not used by default. It can only be used if Suricata is compiled with:

```
-- enable-profiling
```

At the end of each session, Suricata will display the profiling statistics. The list will be displayed sorted.

This order can be changed as pleased. The choice is between ticks, avgticks, checks, maxticks and matches. The setting of your choice will be displayed from high to low.

The amount of time it takes to check the signatures, will be administrated by Suricata. This will be counted in ticks. One tick is one CPU computation. 3 GHz will be 3 billion ticks.

Beside the amount of checks, ticks and matches it will also display the average and the maximum of a rule per session at the end of the line.

The option Limit determines the amount of signatures of which the statistics will be shown, based on the sorting.

```
sort: avgticks
limit: 100
```

Example of how the rule statistics can look like;

```
Rule            Ticks            %      Checks         Matches            Max Tick       ␣
↪    Avg
Ticks
```

(continues on next page)

(continued from previous page)

```
7560            107766621       0.02    138             37              105155334       ␣
↪  780917.54
11963           1605394413      0.29    2623             1              144418923       ␣
↪  612045.14
7040            1431034011      0.26    2500             0              106018209       ␣
↪  572413.60
5726            1437574662      0.26    2623             1              115632900       ␣
↪  548065.06
7037            1355312799      0.24    2562             0              116048286       ␣
↪  529005.78
11964           1276449255      0.23    2623             1              96412347        ␣
↪  486637.15
7042            1272562974      0.23    2623             1              96405993        ␣
↪  485155.54
5719            1233969192      0.22    2562             0              106439661       ␣
↪  481642.93
5720            1204053246      0.21    2562             0              125155431       ␣
↪  469966.14
```

### Packet Profiling

```
packets:

  # Profiling can be disabled here, but it will still have a
  # performance impact if compiled in.


  enabled: yes                                  #this option is enabled by default
  filename: packet_stats.log                    #name of the file in which packet␣
↪profiling information will be
                                                #stored.
  append: yes                                   #If set to yes, new packet profiling␣
↪information will be added to the
                                                #information that was saved last in␣
↪the file.

  # per packet csv output
  csv:

    # Output can be disabled here, but it will still have a
    # performance impact if compiled in.

    enabled: no                                 #the sending of packet output to a csv-
↪file is by default disabled.
    filename: packet_stats.csv                  #name of the file in which csv packet␣
↪profiling information will be
                                                #stored
```

Packet profiling is enabled by default in suricata.yaml but it will only do its job if you compiled Suricata with –enable profiling.

The filename in which packet profiling information will be stored, is packet-stats.log. Information in this file can be added to the last information that was saved there, or if the append option is set to no, the existing file will be overwritten.

Per packet, you can send the output to a csv-file. This file contains one line for each packet with all profiling information of that packet. This option can be used only if Suricata is build with –enable-profiling and if the packet profiling option is enabled in yaml.

It is best to use runmode 'single' if you would like to profile the speed of the code. When using a single thread, there is no situation in which two threads have to wait for each other. When using two threads, the time threads might have to wait for each other will be taken in account when/during profiling packets. For more information see *Packet Profiling*.

### 10.1.18 Application layers

#### SSL/TLS

SSL/TLS parsers track encrypted SSLv2, SSLv3, TLSv1, TLSv1.1 and TLSv1.2 sessions.

Protocol detection is done using patterns and a probing parser running on only TCP/443 by default. The pattern based protocol detection is port independent.

```
tls:
  enabled: yes
  detection-ports:
    dp: 443

  # What to do when the encrypted communications start:
  # - default: keep tracking TLS session, check for protocol anomalies,
  #            inspect tls_* keywords. Disables inspection of unmodified
  #            'content' signatures.
  # - bypass:  stop processing this flow as much as possible. No further
  #            TLS parsing and inspection. Offload flow bypass to kernel
  #            or hardware if possible.
  # - full:    keep tracking and inspection as normal. Unmodified content
  #            keyword signatures are inspected as well.
  #
  # For best performance, select 'bypass'.
  #
  #encrypt-handling: default
```

#### Encrypted traffic

There is no decryption of encrypted traffic, so once the handshake is complete continued tracking of the session is of limited use. The `encrypt-handling` option controls the behavior after the handshake.

If `encrypt-handling` is set to `default` (or if the option is not set), Suricata will continue to track the SSL/TLS session. Inspection will be limited, as raw `content` inspection will still be disabled. There is no point in doing pattern matching on traffic known to be encrypted. Inspection for (encrypted) Heartbleed and other protocol anomalies still happens.

When `encrypt-handling` is set to `bypass`, all processing of this session is stopped. No further parsing and inspection happens. If `stream.bypass` is enabled this will lead to the flow being bypassed, either inside Suricata or by the capture method if it supports it and is configured for it.

Finally, if `encrypt-handling` is set to `full`, Suricata will process the flow as normal, without inspection limitations or bypass.

The option has replaced the `no-reassemble` option. If `no-reassemble` is present, and `encrypt-handling` is not, `false` is intepreted as `encrypt-handling: default` and `true` is interpreted as `encrypt-handling: bypass`.

#### Modbus

According to MODBUS Messaging on TCP/IP Implementation Guide V1.0b, it is recommended to keep the TCP connection opened with a remote device and not to open and close it for each MODBUS/TCP transaction. In that case, it is important to set the stream-depth of the modbus as unlimited.

```
modbus:
  # Stream reassembly size for modbus, default is 0
  stream-depth: 0
```

#### SMTP

SMTP parsers can extract files from attachments. It is also possible to extract raw conversations as files with the key `raw-extraction`. Note that in this case the whole conversation will be stored as a file, including SMTP headers and body content. The filename will be set to "rawmsg". Usual file-related signatures will match on the raw content of the email. This configuration parameter has a `false` default value. It is incompatible with `decode-mime`. If both are enabled, `raw-extraction` will be automatically disabled.

```
smtp:
  # extract messages in raw format from SMTP
  raw-extraction: true
```

### 10.1.19 Decoder

#### Teredo

The Teredo decoder can be disabled. It is enabled by default.

```
decoder:
  # Teredo decoder is known to not be completely accurate
  # it will sometimes detect non-teredo as teredo.
  teredo:
    enabled: true
```

### 10.1.20 Advanced Options

#### luajit

#### states

Luajit has a strange memory requirement, it's 'states' need to be in the first 2G of the process' memory. For this reason when luajit is used the states are allocated at the process startup. This option controls how many states are preallocated.

If the pool is depleted a warning is generated. Suricata will still try to continue, but may fail if other parts of the engine take too much memory. If the pool was depleted a hint will be printed at the engines exit.

States are allocated as follows: for each detect script a state is used per detect thread. For each output script, a single state is used. Keep in mind that a rule reload temporary doubles the states requirement.

## 10.2 Global-Thresholds

Thresholds can be configured in the rules themselves, see *Thresholding Keywords*. They are often set by rule writers based on their intelligence for creating a rule combined with a judgement on how often a rule will alert.

### 10.2.1 Threshold Config

Next to rule thresholding more thresholding can be configured on the sensor using the threshold.config.

#### threshold/event_filter

Syntax:

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

#### rate_filter

Rate filters allow changing of a rule action when a rule matches.

Syntax:

```
rate_filter: rate_filter gen_id <gid>, sig_id <sid>, track <tracker>, \
  count <c>, seconds <s>, new_action <action>, timeout <timeout>
```

Example:

```
rate_filter gen_id 1, sig_id 1000, track by_rule, count 100, seconds 60, \
  new_action alert, timeout 30
```

#### gen_id

Generator id. Normally 1, but if a rule uses the `gid` keyword to set another value it has to be matched in the `gen_id`.

#### sig_id

Rule/signature id as set by the rule `sid` keyword.

#### track

Where to track the rule matches. When using by_src/by_dst the tracking is done per IP-address. The Host table is used for storage. When using by_rule it's done globally for the rule. Option by_both used to track per IP pair of source and destination. Packets going to opposite directions between same addresses tracked as the same pair.

#### count

Number of rule hits before the `rate_filter` is activated.

### seconds

Time period within which the `count` needs to be reached to activate the `rate_filter`

### new_action

New action that is applied to matching traffic when the `rate_filter` is in place.

Values:

```
<alert|drop|pass|reject>
```

Note: 'sdrop' and 'log' are supported by the parser but not implemented otherwise.

### timeout

Time in seconds during which the `rate_filter` will remain active.

### Example

Lets say we want to limit incoming connections to our SSH server. The rule `888` below simply alerts on SYN packets to the SSH port of our SSH server. If an IP-address triggers this more than 10 or more with a minute, the drop `rate_filter` is set with a timeout of 5 minutes.

Rule:

```
alert tcp any any -> $MY_SSH_SERVER 22 (msg:"Connection to SSH server"; \
  flow:to_server; flags:S,12; sid:888;)
```

Rate filter:

```
rate_filter gen_id 1, sig_id 888, track by_src, count 10, seconds 60, \
  new_action drop, timeout 300
```

### suppress

Suppressions can be used to suppress alerts for a rule or a host/network. Actions performed when a rule matches, such as setting a flowbit, are still performed.

Syntax:

```
suppress gen_id <gid>, sig_id <sid>
suppress gen_id <gid>, sig_id <sid>, track <by_src|by_dst|by_either>, ip
↪<ip|subnet|addressvar>
```

Examples:

```
suppress gen_id 1, sig_id 2002087, track by_src, ip 209.132.180.67
```

This will make sure the signature 2002087 will never match for src host 209.132.180.67.

Other possibilities/examples:

```
suppress gen_id 1, sig_id 2003614, track by_src, ip 217.110.97.128/25
suppress gen_id 1, sig_id 2003614, track by_src, ip [192.168.0.0/16,10.0.0.0/8,172.16.
↪0.0/12]
suppress gen_id 1, sig_id 2003614, track by_src, ip $HOME_NET
suppress gen_id 1, sig_id 2003614, track by_either, ip 217.110.97.128/25
```

In the last example above, the `by_either` tracking means that if either the `source ip` or `destination ip` matches `217.110.97.128/25` the rule with sid 2003614 is suppressed.

### 10.2.2 Global thresholds vs rule thresholds

**Note: this section applies to 1.4+ In 1.3 and before mixing rule and global thresholds is not supported.**

When a rule has a threshold/detection_filter set a rule can still be affected by the global threshold file.

The rule below will only fire if 10 or more emails are being delivered/sent from a host within 60 seconds.

```
alert tcp any any -> any 25 (msg:"ET POLICY Inbound Frequent Emails - Possible␣
↪Spambot Inbound"; \
    flow:established; content:"mail from|3a|"; nocase;                          ␣
↪           \
    threshold: type threshold, track by_src, count 10, seconds 60;              ␣
↪           \
    reference:url,doc.emergingthreats.net/2002087; classtype:misc-activity;␣
↪sid:2002087; rev:10;)
```

Next, we'll see how global settings affect this rule.

#### Suppress

Suppressions can be combined with rules with thresholds/detection_filters with no exceptions.

```
suppress gen_id 1, sig_id 2002087, track by_src, ip 209.132.180.67
suppress gen_id 0, sig_id 0, track by_src, ip 209.132.180.67
suppress gen_id 1, sig_id 0, track by_src, ip 209.132.180.67
```

Each of the rules above will make sure 2002087 doesn't alert when the source of the emails is 209.132.180.67. It **will** alert for all other hosts.

```
suppress gen_id 1, sig_id 2002087
```

This suppression will simply convert the rule to "noalert", meaning it will never alert in any case. If the rule sets a flowbit, that will still happen.

#### Threshold/event_filter

When applied to a specific signature, thresholds and event_filters (threshold from now on) will override the signature setting. This can be useful for when the default in a signature doesn't suit your environment.

```
threshold gen_id 1, sig_id 2002087, type both, track by_src, count 3, seconds 5
threshold gen_id 1, sig_id 2002087, type threshold, track by_src, count 10, seconds 60
threshold gen_id 1, sig_id 2002087, type limit, track by_src, count 1, seconds 15
```

Each of these will replace the threshold setting for 2002087 by the new threshold setting.

**Note:** overriding all gids or sids (by using gen_id 0 or sig_id 0) is not supported. Bug https://redmine.openinfosecfoundation.org/issues/425.

#### Rate_filter

see https://redmine.openinfosecfoundation.org/issues/425.

## 10.3 Snort.conf to Suricata.yaml

This guide is meant for those who are familiar with Snort and the snort.conf configuration format. This guide will provide a 1:1 mapping between Snort and Suricata configuration wherever possible.

### 10.3.1 Variables

snort.conf

```
ipvar HOME_NET any
ipvar EXTERNAL_NET any
...

portvar HTTP_PORTS [80,81,311,591,593,901,1220,1414,1741,1830,2301,2381,2809,3128,
↪3702,4343,4848,5250,7001,7145,7510,7777,7779,8000,8008,8014,8028,8080,8088,8090,
↪8118,8123,8180,8181,8243,8280,8800,8888,8899,9000,9080,9090,9091,9443,9999,11371,
↪55555]
portvar SHELLCODE_PORTS !80
...
```

suricata.yaml

```
vars:
  address-groups:

    HOME_NET: "[192.168.0.0/16,10.0.0.0/8,172.16.0.0/12]"
    EXTERNAL_NET: "!$HOME_NET"

  port-groups:
    HTTP_PORTS: "80"
    SHELLCODE_PORTS: "!80"
```

Note that Suricata can automatically detect HTTP traffic regardless of the port it uses. So the HTTP_PORTS variable is not nearly as important as it is with Snort, **if** you use a Suricata enabled ruleset.

### 10.3.2 Decoder alerts

snort.conf

```
# Stop generic decode events:
config disable_decode_alerts

# Stop Alerts on experimental TCP options
```

(continues on next page)

(continued from previous page)

```
config disable_tcpopt_experimental_alerts

# Stop Alerts on obsolete TCP options
config disable_tcpopt_obsolete_alerts

# Stop Alerts on T/TCP alerts
config disable_tcpopt_ttcp_alerts

# Stop Alerts on all other TCPOption type events:
config disable_tcpopt_alerts

# Stop Alerts on invalid ip options
config disable_ipopt_alerts
```

suricata.yaml

Suricata has no specific decoder options. All decoder related alerts are controlled by rules. See #Rules below.

### 10.3.3 Checksum handling

snort.conf

```
config checksum_mode: all
```

suricata.yaml

Suricata's checksum handling works *on-demand*. The stream engine checks TCP and IP checksum by default:

```
stream:
  checksum-validation: yes      # reject wrong csums
```

Alerting on bad checksums can be done with normal rules. See #Rules, decoder-events.rules specifically.

### 10.3.4 Various configs

#### Active response

snort.conf

```
# Configure active response for non inline operation. For more information, see␣
↪REAMDE.active
# config response: eth0 attempts 2
```

suricata.yaml

Active responses are handled automatically w/o config if rules with the "reject" action are used.

#### Dropping privileges

snort.conf

```
# Configure specific UID and GID to run snort as after dropping privs. For more␣
↪information see snort -h command line options
#
# config set_gid:
# config set_uid:
```

Suricata

To set the user and group use the –user <username> and –group <groupname> commandline options.

#### Snaplen

snort.conf

```
# Configure default snaplen. Snort defaults to MTU of in use interface. For more␣
↪information see README
#
# config snaplen:
#
```

Suricata always works at full snap length to provide full traffic visibility.

#### Bpf

snort.conf

```
# Configure default bpf_file to use for filtering what traffic reaches snort. For␣
↪more information see snort -h command line options (-F)
#
# config bpf_file:
#
```

suricata.yaml

BPF filters can be set per packet acquisition method, with the “bpf-filter: <file>” yaml option and in a file using the -F command line option.

For example:

```
pcap:
  - interface: eth0
    #buffer-size: 16777216
    #bpf-filter: "tcp and port 25"
    #checksum-checks: auto
    #threads: 16
    #promisc: no
    #snaplen: 1518
```

### 10.3.5 Log directory

snort.conf

```
# Configure default log directory for snort to log to.  For more information see␣
↪snort -h command line options (-l)
#
# config logdir:
```

suricata.yaml

```
default-log-dir: /var/log/suricata/
```

This value is overridden by the -l commandline option.

### 10.3.6 Packet acquisition

snort.conf

```
# Configure DAQ related options for inline operation. For more information, see␣
↪README.daq
#
# config daq: <type>
# config daq_dir: <dir>
# config daq_mode: <mode>
# config daq_var: <var>
#
# <type> ::= pcap | afpacket | dump | nfq | ipq | ipfw
# <mode> ::= read-file | passive | inline
# <var> ::= arbitrary <name>=<value passed to DAQ
# <dir> ::= path as to where to look for DAQ module so's
```

suricata.yaml

Suricata has all packet acquisition support built-in. It's configuration format is very verbose.

```
pcap:
  - interface: eth0
    #buffer-size: 16777216
    #bpf-filter: "tcp and port 25"
    #checksum-checks: auto
    #threads: 16
    #promisc: no
    #snaplen: 1518
pfring:
afpacket:
nfq:
ipfw:
```

Passive vs inline vs reading files is determined by how Suricata is invoked on the command line.

### 10.3.7 Rules

snort.conf:

In snort.conf a RULE_PATH variable is set, as well as variables for shared object (SO) rules and preprocessor rules.

```
var RULE_PATH ../rules
var SO_RULE_PATH ../so_rules
```

(continues on next page)

(continued from previous page)

```
var PREPROC_RULE_PATH ../preproc_rules

include $RULE_PATH/local.rules
include $RULE_PATH/emerging-activex.rules
...
```

suricata.yaml:

In the suricata.yaml the default rule path is set followed by a list of rule files. Suricata does not have a concept of shared object rules or preprocessor rules. Instead of preprocessor rules, Suricata has several rule files for events set by the decoders, stream engine, http parser etc.

```
default-rule-path: /etc/suricata/rules
rule-files:
 - local.rules
 - emerging-activex.rules
```

The equivalent of preprocessor rules are loaded like normal rule files:

```
rule-files:
 - decoder-events.rules
 - stream-events.rules
 - http-events.rules
 - smtp-events.rules
```

## 10.4 Multi Tenancy

### 10.4.1 Introduction

Multi tenancy support allows for different rule sets with different rule vars. These tenants can then be assigned to VLANs or interfaces (devices).

### 10.4.2 YAML

In the main ("master") YAML, the suricata.yaml, a new section called "multi-detect" should be added.

Settings:

- enabled: yes/no -> is multi-tenancy support enable
- default: yes/no -> is the normal detect config a default 'fall back' tenant?
- selector: direct (for unix socket pcap processing, see below), vlan or device
- loaders: number of 'loader' threads, for parallel tenant loading at startup
- tenants: list of tenants
  - id: tenant id
  - yaml: separate yaml file with the tenant specific settings
- mappings:
  - vlan id or device
  - tenant id: tenant to associate with the vlan id / device

```
multi-detect:
  enabled: yes
  #selector: direct # direct or vlan
  selector: vlan
  loaders: 3

  tenants:
  - id: 1
    yaml: tenant-1.yaml
  - id: 2
    yaml: tenant-2.yaml
  - id: 3
    yaml: tenant-3.yaml

  mappings:
  - vlan-id: 1000
    tenant-id: 1
  - vlan-id: 2000
    tenant-id: 2
  - vlan-id: 1112
    tenant-id: 3
```

The tenant-1.yaml, tenant-2.yaml, tenant-3.yaml each contain a partial configuration:

```
# Set the default rule path here to search for the files.
# if not set, it will look at the current working dir
default-rule-path: /etc/suricata/rules
rule-files:
  - rules1

# You can specify a threshold config file by setting "threshold-file"
# to the path of the threshold config file:
# threshold-file: /etc/suricata/threshold.config

classification-file: /etc/suricata/classification.config
reference-config-file: /etc/suricata/reference.config

# Holds variables that would be used by the engine.
vars:

  # Holds the address group vars that would be passed in a Signature.
  # These would be retrieved during the Signature address parsing stage.
  address-groups:

    HOME_NET: "[192.168.0.0/16,10.0.0.0/8,172.16.0.0/12]"

    EXTERNAL_NET: "!$HOME_NET"

    ...

  port-groups:

    HTTP_PORTS: "80"

    SHELLCODE_PORTS: "!80"

    ...
```

#### vlanid

Assign tenants to vlan id's.

Example of vlan mapping:

```
mappings:
- vlan-id: 1000
  tenant-id: 1
- vlan-id: 2000
  tenant-id: 2
- vlan-id: 1112
  tenant-id: 3
```

The mappings can also be modified over the unix socket, see below.

Note: can only be used if 'vlan.use-for-tracking' is enabled.

#### device

Assign tenants to devices. A single tenant can be assigned to a device. Multiple devices can have the same tenant.

Example of device mapping:

```
mappings:
- device: ens5f0
  tenant-id: 1
- device: ens5f1
  tenant-id: 3
```

The mappings are static and cannot be modified over the unix socket.

Note: Not currently supported for IPS.

Note: support depends on a capture method using the 'livedev' API. Currently these are: pcap, AF_PACKET, PF_RING and Netmap.

### 10.4.3 Per tenant settings

The following settings are per tenant:

- default-rule-path
- rule-files
- classification-file
- reference-config-file
- threshold-file
- address-vars
- port-vars

### 10.4.4 Unix Socket

#### Registration

register-tenant <id> <yaml>

Examples:

```
register-tenant 1 tenant-1.yaml
register-tenant 2 tenant-2.yaml
register-tenant 3 tenant-3.yaml
register-tenant 5 tenant-5.yaml
register-tenant 7 tenant-7.yaml
```

unregister-tenant <id>

```
unregister-tenant 2
unregister-tenant 1
```

#### Unix socket runmode (pcap processing)

The Unix Socket "pcap-file" command can be used to select the tenant to inspect the pcap against:

```
pcap-file traffic1.pcap /logs1/ 1
pcap-file traffic2.pcap /logs2/ 2
pcap-file traffic3.pcap /logs3/ 3
pcap-file traffic4.pcap /logs5/ 5
pcap-file traffic5.pcap /logs7/ 7
```

This runs the traffic1.pcap against tenant 1 and it logs into /logs1/, traffic2.pcap against tenant 2 and logs to /logs2/ and so on.

#### Live traffic mode

For live traffic currently only a vlan based multi-tenancy is supported.

The master yaml needs to have the selector set to "vlan".

#### Registration

Tenants can be mapped to vlan id's.

register-tenant-handler <tenant id> vlan <vlan id>

```
register-tenant-handler 1 vlan 1000
```

unregister-tenant-handler <tenant id> vlan <vlan id>

```
unregister-tenant-handler 4 vlan 1111
unregister-tenant-handler 1 vlan 1000
```

The registration of tenant and tenant handlers can be done on a running engine.

## 10.5 Dropping Privileges After Startup

Currently, libcap-ng is needed for dropping privileges on Suricata after startup. For libcap, see status of feature request number #276 – Libcap support for dropping privileges.

Most distributions have `libcap-ng` in their repositories.

To download the current version of libcap-ng from upstream, see also http://people.redhat.com/sgrubb/libcap-ng/ChangeLog

```
wget http://people.redhat.com/sgrubb/libcap-ng/libcap-ng-0.7.8.tar.gz
tar -xzvf libcap-ng-0.7.8.tar.gz
cd libcap-ng-0.7.8
./configure
make
make install
```

Download, configure, compile and install Suricata for your particular setup. See *Installation*. Depending on your environment, you may need to add the –with-libpcap_ng-libraries and –with-libpcap_ng-includes options during the configure step. e.g:

```
./configure --with-libcap_ng-libraries=/usr/local/lib \
  --with-libcap_ng-includes=/usr/local/include
```

Now, when you run Suricata, tell it what user and/or group you want it to run as after startup with the –user and –group options. e.g. (this assumes a 'suri' user and group):

```
suricata -D -i eth0 --user=suri --group=suri
```

You will also want to make sure your user/group permissions are set so suricata can still write to its log files which are usually located in /var/log/suricata.

```
mkdir -p /var/log/suricata
chown -R root:suri /var/log/suricata
chmod -R 775 /var/log/suricata
```

CHAPTER 11

# Reputation

## 11.1 IP Reputation

### 11.1.1 IP Reputation Config

IP reputation has a few configuration directives, all disabled by default.

```
# IP Reputation
#reputation-categories-file: /etc/suricata/iprep/categories.txt
#default-reputation-path: /etc/suricata/iprep
#reputation-files:
# - reputation.list
```

#### reputation-categories-file

The categories file mapping numbered category values to short names.

```
reputation-categories-file: /etc/suricata/iprep/categories.txt
```

#### default-reputation-path

Path where reputation files from the "reputation-files" directive are loaded from by default.

```
default-reputation-path: /etc/suricata/iprep
```

#### reputation-files

YAML list of file names to load. In case of a absolute path the file is loaded directly, otherwise the path from "default-reputation-path" is pre-pended to form the final path.

```
reputation-files:
 - badhosts.list
 - knowngood.list
 - sharedhosting.list
```

#### Hosts

IP reputation information is stored in the host table, so the settings of the host table affect it.

Depending on the number of hosts reputation information is available for, the memcap and hash size may have to be increased.

#### Reloads

Sending Suricata a USR2 signal will reload the IP reputation data, along with the normal rules reload.

During the reload the host table will be updated to contain the new data. The iprep information is versioned. When the reload is complete, Suricata will automatically clean up the old iprep information.

Only the reputation files will be reloaded, the categories file won't be. If categories change, Suricata should be restarted.

#### File format

The format of the reputation files is described in the *IP Reputation Format* page.

### 11.1.2 IP Reputation Format

Description of IP Reputation file formats. For the configuration see *IP Reputation Config* and *IP Reputation Keyword* for the rule format.

#### Categories file

The categories file provides a mapping between a category number, short name, and long description. It's a simple CSV file:

```
<id>,<short name>,<description>
```

Example:

```
1,BadHosts,Known bad hosts
2,Google,Known google host
```

The maximum value for the category id is hard coded at 60 currently.

#### Reputation file

The reputation file lists a reputation score for hosts in the categories. It's a simple CSV file:

```
<ip>,<category>,<reputation score>
```

The IP is an IPv4 address in the quad-dotted notation or an IPv6 address. Both IP types support networks in CIDR notation. The category is the number as defined in the categories file. The reputation score is the confidence that this IP is in the specified category, represented by a number between 1 and 127 (0 means no data).

Example:

```
1.2.3.4,1,101
1.1.1.0/24,6,88
```

If an IP address has a score in multiple categories it should be listed in the file multiple times.

Example:

```
1.1.1.1,1,10
1.1.1.1,2,10
```

This lists 1.1.1.1 in categories 1 and 2, each with a score of 10.

The purpose of the IP reputation component is the ranking of IP Addresses within the Suricata Engine. It will collect, store, update and distribute reputation intelligence on IP Addresses. The hub and spoke architecture will allows the central database (The Hub) to collect, store and compile updated IP reputation details that are then distributed to user-side sensor databases (Spokes) for inclusion in user security systems. The reputation data update frequency and security action taken, is defined in the user security configuration.

The intent of IP Reputation is to allow sharing of intelligence regarding a vast number of IP addresses. This can be positive or negative intelligence classified into a number of categories. The technical implementation requires three major efforts; engine integration, the hub that redistributes reputation, and the communication protocol between hubs and sensors. The hub will have a number of responsibilities. This will be a separate module running on a separate system as any sensor. Most often it would run on a central database that all sensors already have communication with. It will be able to subscribe to one or more external feeds. The local admin should be able to define the feeds to be subscribed to, provide authentication credentials if required, and give a weight to that feed. The weight can be an overall number or a by category weight. This will allow the admin to minimize the influence a feed has on their overall reputation if they distrust a particular category or feed, or trust another implicitly. Feeds can be configured to accept feedback or not and will report so on connect. The admin can override and choose not to give any feedback, but the sensor should report these to the Hub upstream on connect. The hub will take all of these feeds and aggregate them into an average single score for each IP or IP Block, and then redistribute this data to all local sensors as configured. It should receive connections from sensors. The sensor will have to provide authentication and will provide feedback. The hub should redistribute that feedback from sensors to all other sensors as well as up to any feeds that accept feedback. The hub should also have an API to allow outside statistical analysis to be done to the database and fed back in to the stream. For instance a local site may choose to change the reputation on all Russian IP blocks, etc.

For more information about IP Reputation see *IP Reputation Config*, *IP Reputation Keyword* and *IP Reputation Format*.

# CHAPTER 12

# Init Scripts

For Ubuntu with Upstart, the following can be used in /etc/init/suricata.conf:

```
# suricata
description "Intruder Detection System Daemon"
start on runlevel [2345]
stop on runlevel [!2345]
expect fork
exec suricata -D --pidfile /var/run/suricata.pid -c /etc/suricata/suricata.yaml -i␣
↪eth1
```

CHAPTER 13

# Setting up IPS/inline for Linux

## 13.1 Setting up IPS with Netfilter

In this guide will be explained how to work with Suricata in layer3 inline mode and how to set iptables for that purpose.

First start with compiling Suricata with NFQ support. For instructions see Ubuntu Installation. For more information about NFQ and iptables, see *NFQ*.

To check if you have NFQ enabled in your Suricata, enter the following command:

```
suricata --build-info
```

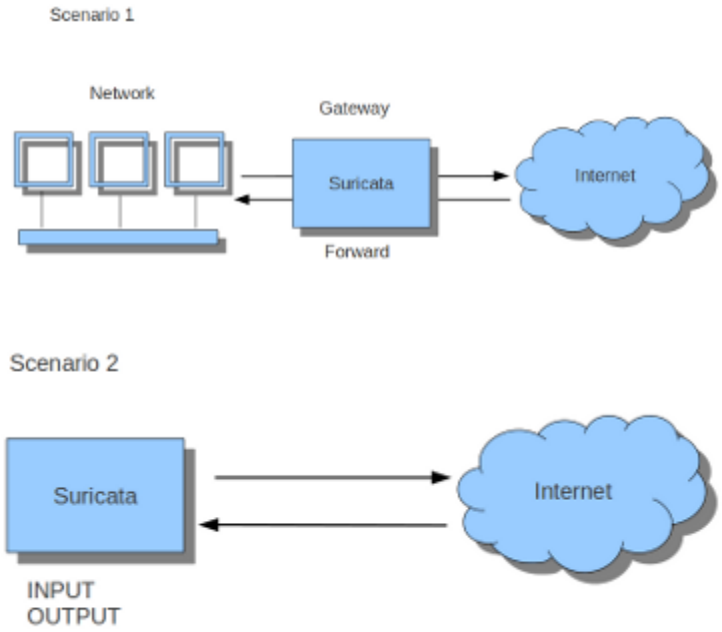
and examine if you have NFQ between the features.

To run suricata with the NFQ mode, you have to make use of the -q option. This option tells Suricata which of the queue numbers it should use.

```
sudo suricata -c /etc/suricata/suricata.yaml -q 0
```

### 13.1.1 Iptables configuration

First of all it is important to know which traffic you would like to send to Suricata. Traffic that passes your computer or traffic that is generated by your computer.




If Suricata is running on a gateway and is meant to protect the computers behind that gateway you are dealing with the first scenario: *forward_ing* . If Suricata has to protect the computer it is running on, you are dealing with the second scenario: *host* (see drawing 2). These two ways of using Suricata can also be combined.

The easiest rule in case of the gateway-scenario to send traffic to Suricata is:

```
sudo iptables -I FORWARD -j NFQUEUE
```

In this case, all forwarded traffic goes to Suricata.

In case of the host situation, these are the two most simple iptables rules;

```
sudo iptables -I INPUT -j NFQUEUE
sudo iptables -I OUTPUT -j NFQUEUE
```

It is possible to set a queue number. If you do not, the queue number will be 0 by default.

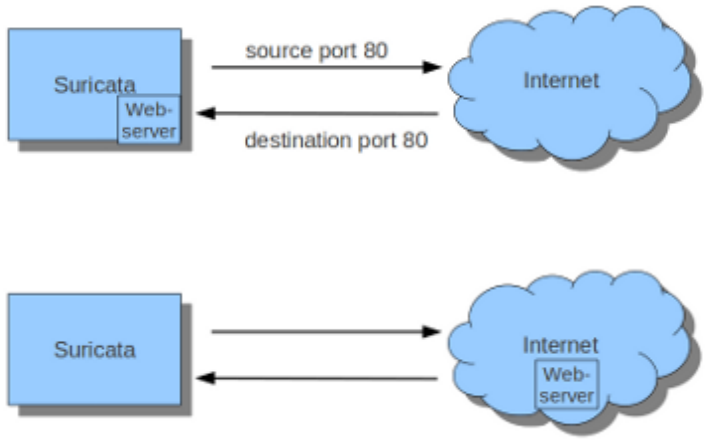
Imagine you want Suricata to check for example just TCP-traffic, or all incoming traffic on port 80, or all traffic on destination-port 80, you can do so like this:

```
sudo iptables -I INPUT -p tcp  -j NFQUEUE
sudo iptables -I OUTPUT -p tcp -j NFQUEUE
```

In this case, Suricata checks just TCP traffic.

```
sudo iptables -I INPUT -p tcp --sport 80  -j NFQUEUE
sudo iptables -I OUTPUT -p tcp --dport 80 -j NFQUEUE
```

In this example, Suricata checks all input and output on port 80.




To see if you have set your iptables rules correct make sure Suricata is running and enter:

```
sudo iptables -vnL
```

In the example you can see if packets are being logged.

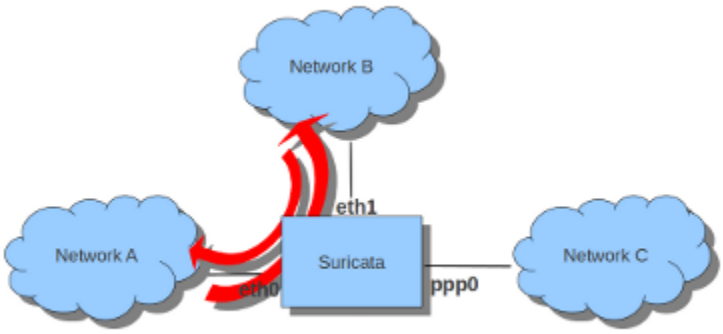
```
anne-fleur@t60:~$ sudo iptables -vnL
Chain INPUT (policy ACCEPT 258 packets, 43900 bytes)
 pkts bytes target     prot opt in     out     source               destination
 4979 5846K NFQUEUE    tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           tcp spt:80 NFQUEUE num 0

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 278 packets, 43459 bytes)
 pkts bytes target     prot opt in     out     source               destination
 5286  388K NFQUEUE    tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           tcp dpt:80 NFQUEUE num 0
anne-fleur@t60:~$
```

This description of the use of iptables is the way to use it with IPv4. To use it with IPv6 all previous mentioned commands have to start with 'ip6tables'. It is also possible to let Suricata check both kinds of traffic.

There is also a way to use iptables with multiple networks (and interface cards). Example:




```
sudo iptables -I FORWARD -i eth0 -o eth1 -j NFQUEUE
sudo iptables -I FORWARD -i eth1 -o eth0 -j NFQUEUE
```

The options -i (input) -o (output) can be combined with all previous mentioned options.

If you would stop Suricata and use internet, the traffic will not come through. To make internet work correctly, you have to erase all iptables rules.

To erase all iptables rules, enter:

```
sudo iptables -F
```

### 13.1.2 Nftables configuration

NFtables configuration is straight forward and allows mixing firewall rules with IPS. The concept is to create a dedicated chain for the IPS that will be evaluated after the firewalling rule. If your main table is named *filter* it can be created like so

```
nft> add chain filter IPS { type filter hook forward priority 10;}
```

To send all forwarded packets to Suricata one can use

```
nft> add rule filter IPS queue
```

To only do it for packets exchanged between eth0 and eth1

```
nft> add rule filter IPS iif eth0 oif eth1 queue
nft> add rule filter IPS iif eth1 oif eth0 queue
```

### 13.1.3 NFQUEUE advanced options

NFQUEUE mechanism supports some interesting options. The nftables configuration will be shown there but the features are also available in iptables.

The full syntax of the queuing mechanism looks as follow

```
nft add rule filter IPS queue num 3-5 options fanout,bypass
```

This rule sends matching packets to 3 load-balanced queues starting at 3 and ending at 5. To get the packets in Suricata with this setup, you need to specify multiple queues on command line:

```
suricata -q 3 -q 4 -q 5
```

*fanout* and *bypass* are the two available options:

- *fanout*: When used together with load balancing, this will use the CPU ID instead of connection hash as an index to map packets to the queues. The idea is that you can improve performance if there's a queue per CPU. This requires total with a number of queues superior to 1 to be specified.
- *bypass*: By default, if no userspace program is listening on an Netfilter queue, then all packets that are to be queued are dropped. When this option is used, the queue rule behaves like ACCEPT if there is no program listening, and the packet will move on to the next table.

The *bypass* option can be used to avoid downtime of link when Suricata is not running but this also means that the blocking feature will not be present.

## 13.2 Settings up IPS at Layer 2

### 13.2.1 AF_PACKET IPS mode

AF_PACKET capture method is supporting a IPS/Tap mode. In this mode, you just need the interfaces to be up. Suricata will take care of copying the packets from one interface to the other. No iptables or nftables configuration is necessary.

You need to dedicate two network interfaces for this mode. The configuration is made via configuration variable available in the description of an AF_PACKET interface.

For example, the following configuration will create a Suricata acting as IPS between interface eth0 and eth1:

```
af-packet:
  - interface: eth0
    threads: 1
    defrag: no
    cluster-type: cluster_flow
    cluster-id: 98
    copy-mode: ips
    copy-iface: eth1
    buffer-size: 64535
    use-mmap: yes
  - interface: eth1
    threads: 1
    cluster-id: 97
    defrag: no
    cluster-type: cluster_flow
    copy-mode: ips
    copy-iface: eth0
    buffer-size: 64535
    use-mmap: yes
```

Basically, we've got an af-packet configuration with two interfaces. Interface eth0 will copy all received packets to eth1 because of the copy-* configuration variable

```
copy-mode: ips
copy-iface: eth1
```

The configuration on eth1 is symmetric

```
copy-mode: ips
copy-iface: eth0
```

There is some important points to consider when setting up this mode:

- The implementation of this mode is dependent of the zero copy mode of AF_PACKET. Thus you need to set *use-mmap* to yes on both interface.
- MTU on both interfaces have to be equal: the copy from one interface to the other is direct and packet bigger then the MTU will be dropped by kernel.
- Set different values of cluster-id on both interfaces to avoid conflict.
- Any network card offloading creating bigger then physical layer datagram (like GRO, LRO, TSO) will result in packets drop as transmit path can not handle them.
- Set *stream.inline* to *auto* or *yes* so Suricata really switches to blocking mode.

The *copy-mode* variable can take the following values:

- *ips*: the drop keyword is honored and matching packets are dropped.
- *tap*: no drop occurs, Suricata acts as a bridge

Some specific care must be taken to scale the capture method on multiple threads. As we can't use defrag that will generate too big frames, the in kernel load balancing will not be correct: the IP-only fragment will not reach the same thread as the full featured packet of the same flow because the port information will not be present.

A solution is to use eBPF load balancing to get an IP pair load balancing that will be fragmentation resistant. The AF_PACKET IPS Configuration using multiple threads and eBPF load balancing looks like the following:

```
af-packet:
  - interface: eth0
    threads: 16
    defrag: no
    cluster-type: cluster_ebpf
    ebpf-lb-file: /usr/libexec/suricata/ebpf/lb.bpf
    cluster-id: 98
    copy-mode: ips
    copy-iface: eth1
    buffer-size: 64535
    use-mmap: yes
  - interface: eth1
    threads: 16
    cluster-id: 97
    defrag: no
    cluster-type: cluster_ebpf
    ebpf-lb-file: /usr/libexec/suricata/ebpf/lb.bpf
    copy-mode: ips
    copy-iface: eth0
    buffer-size: 64535
    use-mmap: yes
```

The eBPF file */usr/libexec/suricata/ebpf/lb.bpf* may not be present on disk. See *eBPF and XDP* for more information.

CHAPTER 14

# Setting up IPS/inline for Windows

This guide explains how to work with Suricata in layer 4 inline mode using WinDivert on Windows.

First start by compiling Suricata with WinDivert support. For instructions, see Windows Installation. This documentation has not yet been updated with WinDivert information, so make sure to add the following flags to *configure*:

```
--enable-windivert=yes --with-windivert-include=<include-dir> --with-windivert-
↪libraries=<libraries-dir>
```

WinDivert.dll and WinDivert.sys must be in the same directory as the Suricata executable. WinDivert automatically installs the driver when it is run. For more information about WinDivert, see https://www.reqrypt.org/windivert-doc.html.

To check if you have WinDivert enabled in your Suricata, enter the following command in an elevated command prompt or terminal:

```
suricata -c suricata.yaml --windivert [filter string]
```

For information on the WinDivert filter language, see https://www.reqrypt.org/windivert-doc.html#filter_language

If Suricata is running on a gateway and is meant to protect the network behind that gateway, you need to run WinDivert at the NETWORK_FORWARD layer. This can be achieved using the following command:

```
suricata -c suricata.yaml --windivert-forward [filter string]
```

The filter is automatically stopped and normal traffic resumes when Suricata is stopped.

A quick start is to examine all traffic, in which case you can use the following command:

```
suricata -c suricata.yaml --windivert[-forward] true
```

A few additional examples:

Only TCP traffic:

```
suricata -c suricata.yaml --windivert tcp
```

Only TCP traffic on port 80:

```
suricata -c suricata.yaml --windivert "tcp.DstPort == 80"
```

TCP and ICMP traffic:

```
suricata -c suricata.yaml --windivert "tcp or icmp"
```

CHAPTER 15

# Output

## 15.1 EVE

### 15.1.1 Eve JSON Output

The EVE output facility outputs alerts, anomalies, metadata, file info and protocol specific records through JSON.

The most common way to use this is through 'EVE', which is a firehose approach where all these logs go into a single file.

```
outputs:
  # Extensible Event Format (nicknamed EVE) event log in JSON format
  - eve-log:
      enabled: yes
      filetype: regular #regular|syslog|unix_dgram|unix_stream|redis
      filename: eve.json
      #prefix: "@cee: " # prefix to prepend to each log entry
      # the following are valid when type: syslog above
      #identity: "suricata"
      #facility: local5
      #level: Info ## possible levels: Emergency, Alert, Critical,
                   ## Error, Warning, Notice, Info, Debug
      #redis:
      #  server: 127.0.0.1
      #  port: 6379
      #  async: true ## if redis replies are read asynchronously
      #  mode: list ## possible values: list|lpush (default), rpush, channel|publish
      #             ## lpush and rpush are using a Redis list. "list" is an alias for␣
↪lpush
      #             ## publish is using a Redis channel. "channel" is an alias for␣
↪publish
      #  key: suricata ## key or channel to use (default to suricata)
      # Redis pipelining set up. This will enable to only do a query every
      # 'batch-size' events. This should lower the latency induced by network
```

(continues on next page)

(continued from previous page)

```
      # connection at the cost of some memory. There is no flushing implemented
      # so this setting as to be reserved to high traffic suricata.
      #  pipelining:
      #    enabled: yes ## set enable to yes to enable query pipelining
      #    batch-size: 10 ## number of entry to keep in buffer

      # Include top level metadata. Default yes.
      #metadata: no

      types:
        - alert:
            # payload: yes             # enable dumping payload in Base64
            # payload-buffer-size: 4kb # max size of payload buffer to output in eve-
↪log
            # payload-printable: yes   # enable dumping payload in printable (lossy)␣
↪format
            # packet: yes              # enable dumping of packet (without stream␣
↪segments)
            # http-body: yes           # Requires metadata; enable dumping of http␣
↪body in Base64
            # http-body-printable: yes # Requires metadata; enable dumping of http␣
↪body in printable format

            # Enable the logging of tagged packets for rules using the
            # "tag" keyword.
            tagged-packets: yes

            # Configure the metadata to be logged along with an
            # alert. The following shows the default configuration
            # which is used if this field is not provided or simply
            # set to a truthful value. Setting of this section is only
            # required if you wish to enable/disable specific fields.
            #metadata:

              # Include the decoded application layer (ie. http, dns)
              app-layer: true

              # Log the the current state of the flow record.
              flow: true

              rule:
                # Log the metadata field from the rule in a structured
                # format.
                metadata: true

                # Log the raw rule text.
                raw: false

            # HTTP X-Forwarded-For support by adding an extra field or overwriting
            # the source or destination IP address (depending on flow direction)
            # with the one reported in the X-Forwarded-For HTTP header. This is
            # helpful when reviewing alerts for traffic that is being reverse
            # or forward proxied.
            xff:
              enabled: no
              # Two operation modes are available, "extra-data" and "overwrite".
              mode: extra-data
```

(continues on next page)

(continued from previous page)

```
        # Two proxy deployments are supported, "reverse" and "forward". In
        # a "reverse" deployment the IP address used is the last one, in a
        # "forward" deployment the first IP address is used.
        deployment: reverse
        # Header name where the actual IP address will be reported, if more
        # than one IP address is present, the last IP address will be the
        # one taken into consideration.
        header: X-Forwarded-For
    - http:
        extended: yes     # enable this for extended logging information
        # custom allows additional http fields to be included in eve-log
        # the example below adds three additional fields when uncommented
        #custom: [Accept-Encoding, Accept-Language, Authorization]
    - dns:
        # Use version 2 logging with the new format:
        # dns answers will be logged in one single event
        # rather than an event for each of the answers.
        # Without setting a version the version
        # will fallback to 1 for backwards compatibility.
        version: 2

        # Enable/disable this logger. Default: enabled.
        #enabled: no

        # Control logging of requests and responses:
        # - requests: enable logging of DNS queries
        # - responses: enable logging of DNS answers
        # By default both requests and responses are logged.
        #requests: no
        #responses: no

        # Format of answer logging:
        # - detailed: array item per answer
        # - grouped: answers aggregated by type
        # Default: all
        #answer-format: [detailed, grouped]

        # Answer types to log.
        # Default: all
        #answer-types: [a, aaaa, cname, mx, ns, ptr, txt]
    - dns:
        # Version 1 (deprecated) DNS logger.
        version: 1

        enabled: no
        # control logging of queries and answers
        # default yes, no to disable
        query: yes     # enable logging of DNS queries
        answer: yes    # enable logging of DNS answers
        # control which RR types are logged
        # all enabled if custom not specified
        #custom: [a, aaaa, cname, mx, ns, ptr, txt]
    - tls:
        extended: yes     # enable this for extended logging information
        # output TLS transaction where the session is resumed using a
        # session id
        #session-resumption: no
```

(continues on next page)

(continued from previous page)

```
            # custom allows to control which tls fields that are included
            # in eve-log
            #custom: [subject, issuer, session_resumed, serial, fingerprint, sni,␣
↪version, not_before, not_after, certificate, chain]
        - files:
            force-magic: no   # force logging magic on all logged files
            # force logging of checksums, available hash functions are md5,
            # sha1 and sha256
            #force-hash: [md5]
        #- drop:
        #    alerts: yes      # log alerts that caused drops
        #    flows: all       # start or all: 'start' logs only a single drop
        #                     # per flow direction. All logs each dropped pkt.
        - smtp:
            #extended: yes # enable this for extended logging information
            # this includes: bcc, message-id, subject, x_mailer, user-agent
            # custom fields logging from the list:
            #  reply-to, bcc, message-id, subject, x-mailer, user-agent, received,
            #  x-originating-ip, in-reply-to, references, importance, priority,
            #  sensitivity, organization, content-md5, date
            #custom: [received, x-mailer, x-originating-ip, relays, reply-to, bcc]
            # output md5 of fields: body, subject
            # for the body you need to set app-layer.protocols.smtp.mime.body-md5
            # to yes
            #md5: [body, subject]

        # NFS logging. Requires Rust.
        - nfs
        # IKEv2 logging. Requires Rust.
        - ikev2
        - ssh
        - stats:
            totals: yes       # stats for all threads merged together
            threads: no       # per thread stats
            deltas: no        # include delta values
        - dhcp:
            # DHCP logging requires Rust.
            enabled: yes
            # When extended mode is on, all DHCP messages are logged
            # with full detail. When extended mode is off (the
            # default), just enough information to map a MAC address
            # to an IP address is logged.
            extended: no
        # bi-directional flows
        - flow
        # uni-directional flows
        #- netflow

        # An event for logging metadata, specifically pktvars when
        # they are set, but will also include the full metadata object.
        #- metadata
```

Each alert, http log, etc will go into this one file: ‘eve.json’. This file can then be processed by 3rd party tools like Logstash (ELK) or jq.

### Output types

EVE can output to multiple methods. `regular` is a normal file. Other options are `syslog`, `unix_dgram`, `unix_stream` and `redis`.

Output types:

```
filetype: regular #regular|syslog|unix_dgram|unix_stream|redis
filename: eve.json
#prefix: "@cee: " # prefix to prepend to each log entry
# the following are valid when type: syslog above
#identity: "suricata"
#facility: local5
#level: Info ## possible levels: Emergency, Alert, Critical,
             ## Error, Warning, Notice, Info, Debug
#redis:
#  server: 127.0.0.1
#  port: 6379
#  async: true ## if redis replies are read asynchronously
#  mode: list ## possible values: list|lpush (default), rpush, channel|publish
#             ## lpush and rpush are using a Redis list. "list" is an alias for lpush
#             ## publish is using a Redis channel. "channel" is an alias for publish
#  key: suricata ## key or channel to use (default to suricata)
# Redis pipelining set up. This will enable to only do a query every
# 'batch-size' events. This should lower the latency induced by network
# connection at the cost of some memory. There is no flushing implemented
# so this setting as to be reserved to high traffic suricata.
#  pipelining:
#    enabled: yes ## set enable to yes to enable query pipelining
#    batch-size: 10 ## number of entry to keep in buffer
```

### Alerts

Alerts are event records for rule matches. They can be amended with metadata, such as the application layer record (HTTP, DNS, etc) an alert was generated for, and elements of the rule.

Metadata:

```
- alert:
    #payload: yes             # enable dumping payload in Base64
    #payload-buffer-size: 4kb # max size of payload buffer to output in eve-log
    #payload-printable: yes   # enable dumping payload in printable (lossy) format
    #packet: yes              # enable dumping of packet (without stream segments)
    #http-body: yes           # Requires metadata; enable dumping of http body in␣
↪Base64
    #http-body-printable: yes # Requires metadata; enable dumping of http body in␣
↪printable format

    # metadata:

      # Include the decoded application layer (ie. http, dns)
      #app-layer: true

      # Log the the current state of the flow record.
      #flow: true

      #rule:
```

(continues on next page)

(continued from previous page)

```
        # Log the metadata field from the rule in a structured
        # format.
        #metadata: true

        # Log the raw rule text.
        #raw: false
```

### Anomaly

Anomalies are event records created when packets with unexpected or anomalous values are handled. These events include conditions such as incorrect protocol values, incorrect protocol length values, and other conditions which render the packet suspect. Other conditions may occur during the normal progression of a stream; these are termed `stream` events are include control sequences with incorrect values or that occur out of expected sequence.

Anomalies are reported by and configured by type:

- Decode
- Stream
- Application layer

Metadata:

```
- anomaly:
    # Anomaly log records describe unexpected conditions such as truncated packets,
    # packets with invalid IP/UDP/TCP length values, and other events that render
    # the packet invalid for further processing or describe unexpected behavior on
    # an established stream. Networks which experience high occurrences of
    # anomalies may experience packet processing degradation.
    #
    # Anomalies are reported for the following:
    # 1. Decode: Values and conditions that are detected while decoding individual
    #    packets. This includes invalid or unexpected values for low-level protocol
    #    lengths as well.
    # 2. Stream: This includes stream related events (TCP 3-way handshake issues,
    #    unexpected sequence number, etc).
    # 3. Application layer: These denote application layer specific conditions that
    #    are unexpected, invalid or are unexpected given the application monitoring
    #    state.
    #
    # By default, anomaly logging is disabled. When anomaly logging is enabled,
    # application-layer anomaly reporting is enabled.
    #
    # Choose one or both types of anomaly logging and whether to enable
    # logging of the packet header for packet anomalies.
    types:
      #decode: no
      #stream: no
      #applayer: yes
    #packethdr: no
```

### HTTP

HTTP transaction logging.

Config:

```
- http:
    extended: yes     # enable this for extended logging information
    # custom allows additional http fields to be included in eve-log
    # the example below adds three additional fields when uncommented
    #custom: [Accept-Encoding, Accept-Language, Authorization]
    # set this value to one among {both, request, response} to dump all
    # http headers for every http request and/or response
    # dump-all-headers: [both, request, response]
```

List of custom fields:

| Yaml Option | HTTP Header |
|---|---|
| accept | accept |
| accept_charset | accept-charset |
| accept_encoding | accept-encoding |
| accept_language | accept-language |
| accept_datetime | accept-datetime |
| authorization | authorization |
| cache_control | cache-control |
| cookie | cookie |
| from | from |
| max_forwards | max-forwards |
| origin | origin |
| pragma | pragma |
| proxy_authorization | proxy-authorization |
| range | range |
| te | te |
| via | via |
| x_requested_with | x-requested-with |
| dnt | dnt |
| x_forwarded_proto | x-forwarded-proto |
| x_authenticated_user | x-authenticated-user |
| x_flash_version | x-flash-version |
| accept_range | accept-range |
| age | age |
| allow | allow |
| connection | connection |
| content_encoding | content-encoding |
| content_language | content-language |
| content_length | content-length |
| content_location | content-location |
| content_md5 | content-md5 |
| content_range | content-range |
| content_type | content-type |
| date | date |
| etag | etags |
| expires | expires |
| last_modified | last-modified |
| link | link |
| location | location |

Continued on next page

Table 1 – continued from previous page

| Yaml Option | HTTP Header |
|---|---|
| proxy_authenticate | proxy-authenticate |
| referrer | referrer |
| refresh | refresh |
| retry_after | retry-after |
| server | server |
| set_cookie | set-cookie |
| trailer | trailer |
| transfer_encoding | transfer-encoding |
| upgrade | upgrade |
| vary | vary |
| warning | warning |
| www_authenticate | www-authenticate |
| true_client_ip | true-client-ip |
| org_src_ip | org-src-ip |
| x_bluecoat_via | x-bluecoat-via |

In the `custom` option values from both columns can be used. The `HTTP Header` column is case insensitive.

### DNS

**Note:** As of Suricata 5.0, the version 2 format of the EVE DNS log is the default.

DNS records are logged as one entry for the request, and one entry for the response.

YAML:

```
- dns:
    # As of Suricata 5.0, version 2 of the eve dns output
    # format is the default.
    #version: 2

    # Enable/disable this logger. Default: enabled.
    #enabled: yes

    # Control logging of requests and responses:
    # - requests: enable logging of DNS queries
    # - responses: enable logging of DNS answers
    # By default both requests and responses are logged.
    #requests: no
    #responses: no

    # Format of answer logging:
    # - detailed: array item per answer
    # - grouped: answers aggregated by type
    # Default: all
    #formats: [detailed, grouped]

    # Types to log, based on the query type.
    # Default: all.
    #types: [a, aaaa, cname, mx, ns, ptr, txt]
```

### DNS v1 Format

The version 1 DNS output has been obsoleted by the version 2 output above. The v1 format logs a record per answer in the response possibly resulting in much more than 2 log records per request and response.

YAML:

```
- dns:
    # Must set the version to 1 to get the old style format.
    version: 1
    # control logging of queries and answers
    # default yes, no to disable
    query: yes     # enable logging of DNS queries
    answer: yes    # enable logging of DNS answers
    # control which RR types are logged
    # all enabled if custom not specified
    #custom: [a, aaaa, cname, mx, ns, ptr, txt]
```

To reduce verbosity the output can be filtered by supplying the record types to be logged under `custom`.

### TLS

TLS records are logged one record per session.

YAML:

```
- tls:
    extended: yes     # enable this for extended logging information
    # custom allows to control which tls fields that are included
    # in eve-log
    #custom: [subject, issuer, serial, fingerprint, sni, version, not_before, not_
↪after, certificate, chain, ja3, ja3s]
```

The default is to log certificate subject and issuer. If `extended` is enabled, then the log gets more verbose.

By using `custom` it is possible to select which TLS fields to log.

### Date modifiers in filename

It is possible to use date modifiers in the eve-log filename.

```
outputs:
  - eve-log:
      filename: eve-%s.json
```

The example above adds epoch time to the filename. All the date modifiers from the C library should be supported. See the man page for `strftime` for all supported modifiers.

### Rotate log file

Eve-log can be configured to rotate based on time.

```
outputs:
  - eve-log:
      filename: eve-%Y-%m-%d-%H:%M.json
      rotate-interval: minute
```

The example above creates a new log file each minute, where the filename contains a timestamp. Other supported `rotate-interval` values are `hour` and `day`.

In addition to this, it is also possible to specify the `rotate-interval` as a relative value. One example is to rotate the log file each X seconds.

```
outputs:
  - eve-log:
      filename: eve-%Y-%m-%d-%H:%M:%S.json
      rotate-interval: 30s
```

The example above rotates eve-log each 30 seconds. This could be replaced with `30m` to rotate every 30 minutes, `30h` to rotate every 30 hours, `30d` to rotate every 30 days, or `30w` to rotate every 30 weeks.

### Multiple Logger Instances

It is possible to have multiple 'EVE' instances, for example the following is valid:

```
outputs:
  - eve-log:
      enabled: yes
      type: file
      filename: eve-ips.json
      types:
        - alert
        - drop

  - eve-log:
      enabled: yes
      type: file
      filename: eve-nsm.json
      types:
        - http
        - dns
        - tls
```

So here the alerts and drops go into 'eve-ips.json', while http, dns and tls go into 'eve-nsm.json'.

With the exception of `drop`, you can specify multiples of the same logger type, however, `drop` can only be used once.

---

**Note:** The use of independent json loggers such as alert-json-log, dns-json-log, etc. has been deprecated and will be removed by June 2020. Please use multiple eve-log instances as documented above instead. Please see the deprecation policy for more information.

---

### File permissions

Log file permissions can be set individually for each logger. `filemode` can be used to control the permissions of a log file, e.g.:

```
outputs:
  - eve-log:
      enabled: yes
```

(continues on next page)

(continued from previous page)

```
      filename: eve.json
      filemode: 600
```

The example above sets the file permissions on `eve.json` to 600, which means that it is only readable and writable by the owner of the file.

### JSON flags

Several flags can be specified to control the JSON output in EVE:

```
outputs:
  - eve-log:
      json:
        # Sort object keys in the same order as they were inserted
        preserve-order: yes

        # Make the output more compact
        compact: yes

        # Escape all unicode characters outside the ASCII range
        ensure-ascii: yes

        # Escape the '/' characters in string with '\/'
        escape-slash: yes
```

All these flags are enabled by default, and can be modified per EVE instance.

### Community Flow ID

Often Suricata is used in combination with other tools like Bro/Zeek. Enabling the community-id option in the eve-log section adds a new `community_id` field to each output.

Example:

```
{
  "timestamp": "2003-12-16T13:21:44.891921+0000",
  "flow_id": 1332028388187153,
  "pcap_cnt": 1,
  "event_type": "alert",
  ...
  "community_id": "1:LQU9qZlK+B5F3KDmev6m5PMibrg=",
  "alert": {
    "action": "allowed",
    "gid": 1,
    "signature_id": 1,
  },
}
{
  "timestamp": "2003-12-16T13:21:45.037333+0000",
  "flow_id": 1332028388187153,
  "event_type": "flow",
  "flow": {
    "pkts_toserver": 5,
    "pkts_toclient": 4,
```

(continues on next page)

(continued from previous page)

```
    "bytes_toserver": 338,
    "bytes_toclient": 272,
    "start": "2003-12-16T13:21:44.891921+0000",
    "end": "2003-12-16T13:21:45.346457+0000",
    "age": 1,
    "state": "closed",
    "reason": "shutdown",
    "alerted": true
  },
  "community_id": "1:LQU9qZlK+B5F3KDmev6m5PMibrg=",
}
```

#### Options

The output can be enabled per instance of the EVE logger.

The `community-id` option is boolean. If set to `true` it is enabled. The `community-id-seed` option specifies a unsigned 16 bit value that is used a seed to the hash that is calculated for the `community-id` output. This must be set to the same value on all tools that output this record.

YAML:

```
- eve-log:
    # Community Flow ID
    # Adds a 'community_id' field to EVE records. These are meant to give
    # a records a predictable flow id that can be used to match records to
    # output of other tools such as Bro.
    #
    # Takes a 'seed' that needs to be same across sensors and tools
    # to make the id less predictable.

    # enable/disable the community id feature.
    community-id: false
    # Seed value for the ID output. Valid values are 0-65535.
    community-id-seed: 0
```

### 15.1.2 Eve JSON Format

Example:

```
{
    "timestamp": "2009-11-24T21:27:09.534255",
    "event_type": "alert",
    "src_ip": "192.168.2.7",
    "src_port": 1041,
    "dest_ip": "x.x.250.50",
    "dest_port": 80,
    "proto": "TCP",
    "alert": {
        "action": "allowed",
        "gid": 1,
        "signature_id" :2001999,
        "rev": 9,
        "signature": "ET MALWARE BTGrab.com Spyware Downloading Ads",
```

(continues on next page)

(continued from previous page)

```
        "category": "A Network Trojan was detected",
        "severity": 1
    }
}
```

### Common Section

All the JSON log types share a common structure:

```
{"timestamp":"2009-11-24T21:27:09.534255","event_type":"TYPE", ...tuple... ,"TYPE":{ .
↪.. type specific content ... }}
```

### Event types

The common part has a field "event_type" to indicate the log type.

```
"event_type":"TYPE"
```

### PCAP fields

If Suricata is processing a pcap file, additional fields are added:

```
"pcap_cnt": 123
```

`pcap_cnt` contains the packet number in the pcap. This can be used to look up a packet in Wireshark for example.

```
"pcap_filename":"/path/to/file.pcap"
```

`pcap_filename` contains the file name and location of the pcap that generated the event.

---

**Note:** the pcap fields are only available on "real" packets, and are omitted from internal "pseudo" packets such as flow timeout packets.

---

### Event type: Alert

### Field action

Possible values: "allowed" and "blocked"

Example:

```
"action":"allowed"
```

Action is set to "allowed" unless a rule used the "drop" action and Suricata is in IPS mode, or when the rule used the "reject" action.

It can also contain information about Source and Target of the attack in the alert.source and alert.target field it target keyword is used in the signature.

```
"alert": {
  "action": "allowed",
  "gid": 1,
  "signature_id": 1,
  "rev": 1,
  "app_proto": "http",
  "signature": "HTTP body talking about corruption",
  "severity": 3,
  "source": {
    "ip": "192.168.43.32",
    "port": 36292
  },
  "target": {
    "ip": "179.60.192.3",
    "port": 80
  },
```

### Event type: Anomaly

Events with type "anomaly" report unexpected conditions such as truncated packets, packets with invalid values, events that render the packet invalid for further processing or unexpected behaviors.

Networks which experience high occurrences of anomalies may experience packet processing degradation when anomaly logging is enabled.

### Fields

- "type": Either "decode", "stream" or "applayer". In rare cases, type will be "unknown". When this occurs, an additional field named "code" will be present. Events with type "applayer" are detected by the application layer parsers.
- "event" The name of the anomalous event. Events of type "decode" are prefixed with "decoder"; events of type "stream" are prefixed with "stream".
- "code" If "type" is "unknown", than "code" contains the unrecognized event code. Otherwise, this field is not present.

The following field is included when "type" has the value "applayer":

- "layer" Indicates the handling layer that detected the event. This will be "proto_parser" (protocol parser), "proto_detect" (protocol detection) or "parser."

When `packethdr` is enabled, the first 32 bytes of the packet are included as a byte64-encoded blob in the main part of record. This applies to events of "type" "packet" or "stream" only.

### Examples

```
"anomaly": {
  "type": "decode",
  "event": "decoder.icmpv4.unknown_type"
}

"anomaly": {
  "type": "decode",
```

(continues on next page)

(continued from previous page)


```
  "event": "decoder.udp.pkt_too_small"
}

"anomaly": {
  "type": "decode",
  "event": "decoder.ipv4.wrong_ip_version"
}

"anomaly": {
  "type": "stream",
  "event": "stream.pkt_invalid_timestamp"
}

{
  "timestamp": "1969-12-31T16:04:21.000000-0800",
  "pcap_cnt": 9262,
  "event_type": "anomaly",
  "src_ip": "208.21.2.184",
  "src_port": 0,
  "dest_ip": "10.1.1.99",
  "dest_port": 0,
  "proto": "UDP",
  "packet": "////////AQEBAQEBCABFAAA8xZ5AAP8R1+DQFQK4CgE=",
  "packet_info": {
    "linktype": 1
  },
  "anomaly": {
    "type": "decode",
    "event": "decoder.udp.pkt_too_small"
  }
}

{
  "timestamp": "2016-01-11T05:10:54.612110-0800",
  "flow_id": 412547343494194,
  "pcap_cnt": 1391293,
  "event_type": "anomaly",
  "src_ip": "192.168.122.149",
  "src_port": 49324,
  "dest_ip": "69.195.71.174",
  "dest_port": 443,
  "proto": "TCP",
  "app_proto": "tls",
  "anomaly": {
    "type": "applayer",
    "event": "APPLAYER_DETECT_PROTOCOL_ONLY_ONE_DIRECTION",
    "layer": "proto_detect"
  }
}

{
  "timestamp": "2016-01-11T05:10:52.828802-0800",
  "flow_id": 201217772575257,
  "pcap_cnt": 1391281,
  "event_type": "anomaly",
  "src_ip": "192.168.122.149",
  "src_port": 49323,
```


(continues on next page)

(continued from previous page)

```
  "dest_ip": "69.195.71.174",
  "dest_port": 443,
  "proto": "TCP",
  "tx_id": 0,
  "app_proto": "tls",
  "anomaly": {
    "type": "applayer",
    "event": "INVALID_RECORD_TYPE",
    "layer": "proto_parser"
  }
}
```

### Event type: HTTP

### Fields

- "hostname": The hostname this HTTP event is attributed to
- "url": URL at the hostname that was accessed
- "http_user_agent": The user-agent of the software that was used
- "http_content_type": The type of data returned (ex: application/x-gzip)
- "cookie"

In addition to these fields, if the extended logging is enabled in the suricata.yaml file the following fields are (can) also included:

- "length": The content size of the HTTP body
- "status": HTTP status code
- "protocol": Protocol / Version of HTTP (ex: HTTP/1.1)
- "http_method": The HTTP method (ex: GET, POST, HEAD)
- "http_refer": The referrer for this action

In addition to the extended logging fields one can also choose to enable/add from more than 50 additional custom logging HTTP fields enabled in the suricata.yaml file. The additional fields can be enabled as following:

```
- eve-log:
    enabled: yes
    type: file #file|syslog|unix_dgram|unix_stream
    filename: eve.json
    # the following are valid when type: syslog above
    #identity: "suricata"
    #facility: local5
    #level: Info ## possible levels: Emergency, Alert, Critical,
                 ## Error, Warning, Notice, Info, Debug
    types:
      - alert
      - http:
          extended: yes     # enable this for extended logging information
          # custom allows additional http fields to be included in eve-log
          # the example below adds three additional fields when uncommented
          #custom: [Accept-Encoding, Accept-Language, Authorization]
```

(continues on next page)

(continued from previous page)

```
          custom: [accept, accept-charset, accept-encoding, accept-language,
          accept-datetime, authorization, cache-control, cookie, from,
          max-forwards, origin, pragma, proxy-authorization, range, te, via,
          x-requested-with, dnt, x-forwarded-proto, accept-range, age,
          allow, connection, content-encoding, content-language,
          content-length, content-location, content-md5, content-range,
          content-type, date, etags, expires, last-modified, link, location,
          proxy-authenticate, referrer, refresh, retry-after, server,
          set-cookie, trailer, transfer-encoding, upgrade, vary, warning,
          www-authenticate, x-flash-version, x-authenticated-user]
```

The benefits here of using the extended logging is to see if this action for example was a POST or perhaps if a download of an executable actually returned any bytes.

It is also possible to dump every header for HTTP requests/responses or both via the keyword `dump-all-headers`.

### Examples

Event with non-extended logging:

```
"http": {
    "hostname": "www.digip.org",
    "url" :"\/jansson\/releases\/jansson-2.6.tar.gz",
    "http_user_agent": "<User-Agent>",
    "http_content_type": "application\/x-gzip"
}
```

In case the hostname shows a port number, such as in case there is a header "Host: www.test.org:1337":

```
"http": {
    "http_port": 1337,
    "hostname": "www.test.org",
    "url" :"\/this\/is\/test.tar.gz",
    "http_user_agent": "<User-Agent>",
    "http_content_type": "application\/x-gzip"
}
```

Event with extended logging:

```
"http": {
    "hostname": "direkte.vg.no",
    "url":".....",
    "http_user_agent": "<User-Agent>",
    "http_content_type": "application\/json",
    "http_refer": "http:\/\/www.vg.no\/",
    "http_method": "GET",
    "protocol": "HTTP\/1.1",
    "status":"200",
    "length":310
}
```

Event with `dump-all-headers` set to "both":

```
"http": {
    "hostname": "test.co.uk",
```

(continues on next page)

(continued from previous page)


```
    "url":"\/test\/file.json",
    "http_user_agent": "<User-Agent>",
    "http_content_type": "application\/json",
    "http_refer": "http:\/\/www.test.com\/",
    "http_method": "GET",
    "protocol": "HTTP\/1.1",
    "status":"200",
    "length":310,
    "request_headers": [
        {
            "name": "User-Agent",
            "value": "Wget/1.13.4 (linux-gnu)"
        },
        {
            "name": "Accept",
            "value": "*/*"
        },
    ],
    "response_headers": [
        {
            "name": "Date",
            "value": "Wed, 25 Mar 2015 15:40:41 GMT"
        },
    ]
}
```


### Event type: DNS

A new version of dns logging has been introduced to improve how dns answers are logged.

With that new version, dns answers are logged in one event rather than an event for each answer.

It's possible to customize how a dns answer will be logged with the following formats:

- "detailed": "rrname", "rrtype", "rdata" and "ttl" fields are logged for each answer
- "grouped": answers logged are aggregated by their type (A, AAAA, NS, . . . )

It will be still possible to use the old DNS logging format, you can control it with "version" option in dns configuration section.

### Fields

Outline of fields seen in the different kinds of DNS events:

- "type": Indicating DNS message type, can be "answer" or "query".
- "id": Identifier field
- "version": Indicating DNS logging version in use
- "flags": Indicating DNS answer flag, in hexadecimal (ex: 8180 , please note 0x is not output)
- "qr": Indicating in case of DNS answer flag, Query/Response flag (ex: true if set)
- "aa": Indicating in case of DNS answer flag, Authoritative Answer flag (ex: true if set)
- "tc": Indicating in case of DNS answer flag, Truncation flag (ex: true if set)

- "rd": Indicating in case of DNS answer flag, Recursion Desired flag (ex: true if set)
- "ra": Indicating in case of DNS answer flag, Recursion Available flag (ex: true if set)
- "rcode": (ex: NOERROR)
- "rrname": Resource Record Name (ex: a domain name)
- "rrtype": Resource Record Type (ex: A, AAAA, NS, PTR)
- "rdata": Resource Data (ex. IP that domain name resolves to)
- "ttl": Time-To-Live for this resource record

One can also control which RR types are logged explicitly from additional custom field enabled in the suricata.yaml file. If custom field is not specified, all RR types are logged. More than 50 values can be specified with the custom field and can be used as following:

```
- eve-log:
    enabled: yes
    type: file #file|syslog|unix_dgram|unix_stream
    filename: eve.json
    # the following are valid when type: syslog above
    #identity: "suricata"
    #facility: local5
    #level: Info ## possible levels: Emergency, Alert, Critical,
                 ## Error, Warning, Notice, Info, Debug
    types:
      - alert
      - dns:
        # control logging of queries and answers
        # default yes, no to disable
        query: yes     # enable logging of DNS queries
        answer: yes    # enable logging of DNS answers
        # control which RR types are logged
        # all enabled if custom not specified
        #custom: [a, aaaa, cname, mx, ns, ptr, txt]
        custom: [a, ns, md, mf, cname, soa, mb, mg, mr, null,
        wks, ptr, hinfo, minfo, mx, txt, rp, afsdb, x25, isdn,
        rt, nsap, nsapptr, sig, key, px, gpos, aaaa, loc, nxt,
        srv, atma, naptr, kx, cert, a6, dname, opt, apl, ds,
        sshfp, ipseckey, rrsig, nsec, dnskey, dhcid, nsec3,
        nsec3param, tlsa, hip, cds, cdnskey, spf, tkey,
        tsig, maila, any, uri]
```

### Examples

Example of a DNS query for the IPv4 address of "twitter.com" (resource record type 'A'):

```
"dns": {
    "type": "query",
    "id": 16000,
    "rrname": "twitter.com",
    "rrtype":"A"
}
```

Example of a DNS answer with "detailed" format:

```
"dns": {
    "version": 2,
    "type": "answer",
    "id": 45444,
    "flags": "8180",
    "qr": true,
    "rd": true,
    "ra": true,
    "rcode": "NOERROR",
    "answers": [
      {
        "rrname": "www.suricata-ids.org",
        "rrtype": "CNAME",
        "ttl": 3324,
        "rdata": "suricata-ids.org"
      },
      {
        "rrname": "suricata-ids.org",
        "rrtype": "A",
        "ttl": 10,
        "rdata": "192.0.78.24"
      },
      {
        "rrname": "suricata-ids.org",
        "rrtype": "A",
        "ttl": 10,
        "rdata": "192.0.78.25"
      }
    ]
}
```

Example of a DNS answer with "grouped" format:

```
"dns": {
    "version": 2,
    "type": "answer",
    "id": 18523,
    "flags": "8180",
    "qr": true,
    "rd": true,
    "ra": true,
    "rcode": "NOERROR",
    "grouped": {
      "A": [
        "192.0.78.24",
        "192.0.78.25"
      ],
      "CNAME": [
        "suricata-ids.org"
      ]
    }
}
```

Example of a old DNS answer with an IPv4 (resource record type 'A') return:

```
"dns": {
    "type": "answer",
```

(continues on next page)

(continued from previous page)

```
    "id":16000,
    "flags":"8180",
    "qr":true,
    "rd":true,
    "ra":true,
    "rcode":"NOERROR"
    "rrname": "twitter.com",
    "rrtype":"A",
    "ttl":8,
    "rdata": "199.16.156.6"
}
```

### Event type: FTP

### Fields

- "command": The FTP command.
- "command_data": The data accompanying the command.
- "reply": The command reply, which may contain multiple lines, in array format.
- "completion_code": The 3-digit completion code. The first digit indicates whether the response is good, bad or incomplete. This is also in array format and may contain multiple completion codes matching multiple reply lines.
- "dynamic_port": The dynamic port established for subsequent data transfers, when applicable, with a "PORT" or "EPRT" command.
- "mode": The type of FTP connection. Most connections are "passive" but may be "active".
- "reply_received": Indicates whether a response was matched to the command. In some non-typical cases, a command may lack a response.

### Examples

Example of regular FTP logging:

```
"ftp": {
  "command": "RETR",
  "command_data": "100KB.zip",
  "reply": [
    "Opening BINARY mode data connection for 100KB.zip (102400 bytes).",
    "Transfer complete."
  ],
  "completion_code": [
    "150",
    "226"
  ],
```

Example showing all fields:

```
"ftp": {
  "command": "EPRT",
  "command_data": "|2|2a01:e34:ee97:b130:8c3e:45ea:5ac6:e301|41813|",
```

(continues on next page)

(continued from previous page)

```
  "reply": [
    "EPRT command successful. Consider using EPSV."
  ],
  "completion_code": [
    "200"
  ],
  "dynamic_port": 41813,
  "mode": "active",
  "reply_received": "yes"
}
```

### Event type: FTP_DATA

#### Fields

- "command": The FTP command associated with the event.
- "filename": The name of the involved file.

#### Examples

Example of FTP_DATA logging:

```
"ftp_data": {
  "filename": "temp.txt",
  "command": "RETR"
}
```

### Event type: TLS

#### Fields

- "subject": The subject field from the TLS certificate
- "issuer": The issuer field from the TLS certificate
- "session_resumed": This field has the value of "true" if the TLS session was resumed via a session id. If this field appears, "subject" and "issuer" do not appear, since a TLS certificate is not seen.

If extended logging is enabled the following fields are also included:

- "serial": The serial number of the TLS certificate
- "fingerprint": The (SHA1) fingerprint of the TLS certificate
- "sni": The Server Name Indication (SNI) extension sent by the client
- "version": The SSL/TLS version used
- "not_before": The NotBefore field from the TLS certificate
- "not_after": The NotAfter field from the TLS certificate
- "ja3": The JA3 fingerprint consisting of both a JA3 hash and a JA3 string
- "ja3s": The JA3S fingerprint consisting of both a JA3 hash and a JA3 string

JA3 must be enabled in the Suricata config file (set 'app-layer.protocols.tls.ja3-fingerprints' to 'yes').

In addition to this, custom logging also allows the following fields:

- "certificate": The TLS certificate base64 encoded
- "chain": The entire TLS certificate chain base64 encoded

### Examples

Example of regular TLS logging:

```
"tls": {
    "subject": "C=US, ST=California, L=Mountain View, O=Google Inc, CN=*.google.com",
    "issuerdn": "C=US, O=Google Inc, CN=Google Internet Authority G2"
}
```

Example of regular TLS logging for resumed sessions:

```
"tls": {
    "session_resumed": true
}
```

Example of extended TLS logging:

```
"tls": {
    "subject": "C=US, ST=California, L=Mountain View, O=Google Inc, CN=*.google.com",
    "issuerdn": "C=US, O=Google Inc, CN=Google Internet Authority G2",
    "serial": "0C:00:99:B7:D7:54:C9:F6:77:26:31:7E:BA:EA:7C:1C",
    "fingerprint": "8f:51:12:06:a0:cc:4e:cd:e8:a3:8b:38:f8:87:59:e5:af:95:ca:cd",
    "sni": "calendar.google.com",
    "version": "TLS 1.2",
    "notbefore": "2017-01-04T10:48:43",
    "notafter": "2017-03-29T10:18:00"
}
```

Example of certificate logging using TLS custom logging (subject, sni, certificate):

```
"tls": {
    "subject": "C=US, ST=California, L=Mountain View, O=Google Inc, CN=*.googleapis.
↪com
    "sni": "www.googleapis.com",
    "certificate": "MIIE3TCCA8WgAwIBAgIIQPsvobRZN0gwDQYJKoZIhvcNAQELBQAwSTELMA [...]"
 }
```

### Event type: TFTP

### Fields

- "packet": The operation code, can be "read" or "write" or "error"
- "file": The filename transported with the tftp protocol
- "mode": The mode field, can be "octet" or "mail" or "netascii" (or any combination of upper and lower case)

Example of TFTP logging:

```
"tftp": {
    "packet": "write",
    "file": "rfc1350.txt",
    "mode": "octet"
 }
```

### Event type: SMB

### SMB Fields

- "id" (integer): internal transaction id
- "dialect" (string): the negotiated protocol dialect, or "unknown" if missing
- "command" (string): command name. E.g. SMB2_COMMAND_CREATE or SMB1_COMMAND_WRITE_ANDX
- "status" (string): status string. Can be both NT_STATUS or DOS_ERR and other variants
- "status_code" (string): status code as hex string
- "session_id" (integer): SMB2+ session_id. SMB1 user id.
- "tree_id" (integer): Tree ID
- "filename" (string): filename for CREATE and other commands.
- "disposition" (string): requested disposition. E.g. FILE_OPEN, FILE_CREATE and FILE_OVERWRITE. See https://msdn.microsoft.com/en-us/library/ee442175.aspx#Appendix_A_Target_119
- "access" (string): indication of how the file was opened. "normal" or "delete on close" (field is subject to change)
- "created", "accessed", "modified", "changed" (integer): timestamps in seconds since unix epoch
- "size" (integer): size of the requested file
- "fuid" (string): SMB2+ file GUID. SMB1 FID as hex.
- "share" (string): share name.
- "share_type" (string): FILE, PIPE, PRINT or unknown.
- "client_dialects" (array of strings): list of SMB dialects the client speaks.
- "client_guid" (string): client GUID
- "server_guid" (string): server GUID
- "request.native_os" (string): SMB1 native OS string
- "request.native_lm" (string): SMB1 native Lan Manager string
- "response.native_os" (string): SMB1 native OS string
- "response.native_lm" (string): SMB1 native Lan Manager string

Examples of SMB logging:

Pipe open:

```
"smb": {
  "id": 1,
  "dialect": "unknown",
  "command": "SMB2_COMMAND_CREATE",
  "status": "STATUS_SUCCESS",
  "status_code": "0x0",
  "session_id": 4398046511201,
  "tree_id": 1,
  "filename": "atsvc",
  "disposition": "FILE_OPEN",
  "access": "normal",
  "created": 0,
  "accessed": 0,
  "modified": 0,
  "changed": 0,
  "size": 0,
  "fuid": "0000004d-0000-0000-0005-0000ffffffff"
}
```

File/pipe close:

```
"smb": {
  "id": 15,
  "dialect": "2.10",
  "command": "SMB2_COMMAND_CLOSE",
  "status": "STATUS_SUCCESS",
  "status_code": "0x0",
  "session_id": 4398046511121,
  "tree_id": 1,
}
```

Tree connect (share open):

```
"smb": {
  "id": 3,
  "dialect": "2.10",
  "command": "SMB2_COMMAND_TREE_CONNECT",
  "status": "STATUS_SUCCESS",
  "status_code": "0x0",
  "session_id": 4398046511121,
  "tree_id": 1,
  "share": "\\\\admin-pc\\c$",
  "share_type": "FILE"
}
```

Dialect negotiation from SMB1 to SMB2 dialect 2.10:

```
"smb": {
  "id": 1,
  "dialect": "2.??",
  "command": "SMB1_COMMAND_NEGOTIATE_PROTOCOL",
  "status": "STATUS_SUCCESS",
  "status_code": "0x0",
  "session_id": 0,
  "tree_id": 0,
  "client_dialects": [
    "PC NETWORK PROGRAM 1.0",
```

(continues on next page)

(continued from previous page)


```
    "LANMAN1.0",
    "Windows for Workgroups 3.1a",
    "LM1.2X002",
    "LANMAN2.1",
    "NT LM 0.12",
    "SMB 2.002",
    "SMB 2.???"
  ],
  "server_guid": "aec6e793-2b11-4019-2d95-55453a0ad2f1"
}
"smb": {
  "id": 2,
  "dialect": "2.10",
  "command": "SMB2_COMMAND_NEGOTIATE_PROTOCOL",
  "status": "STATUS_SUCCESS",
  "status_code": "0x0",
  "session_id": 0,
  "tree_id": 0,
  "client_dialects": [
    "2.02",
    "2.10"
  ],
  "client_guid": "601985d2-aad9-11e7-8494-00088bb57f27",
  "server_guid": "aec6e793-2b11-4019-2d95-55453a0ad2f1"
}
```


SMB1 partial SMB1_COMMAND_SESSION_SETUP_ANDX:


```
"request": {
  "native_os": "Unix",
  "native_lm": "Samba 3.9.0-SVN-build-11572"
},
"response": {
  "native_os": "Windows (TM) Code Name \"Longhorn\" Ultimate 5231",
  "native_lm": "Windows (TM) Code Name \"Longhorn\" Ultimate 6.0"
}
```


### DCERPC fields

- "request" (string): command. E.g. REQUEST, BIND.
- "response" (string): reply. E.g. RESPONSE, BINDACK or FAULT.
- "opnum" (integer): the opnum
- "call_id" (integer): the call id
- "frag_cnt" (integer): the number of fragments for the stub data
- "stub_data_size": total stub data size
- "interfaces" (array): list of interfaces
- "interfaces.uuid" (string): string representation of the UUID
- "interfaces.version" (string): interface version
- "interfaces.ack_result" (integer): ack result

- "interfaces.ack_reason" (integer): ack reason

DCERPC REQUEST/RESPONSE:


```
"smb": {
  "id": 4,
  "dialect": "unknown",
  "command": "SMB2_COMMAND_IOCTL",
  "status": "STATUS_SUCCESS",
  "status_code": "0x0",
  "session_id": 4398046511201,
  "tree_id": 0,
  "dcerpc": {
    "request": "REQUEST",
    "response": "RESPONSE",
    "opnum": 0,
    "req": {
      "frag_cnt": 1,
      "stub_data_size": 136
    },
    "res": {
      "frag_cnt": 1,
      "stub_data_size": 8
    },
    "call_id": 2
  }
}
```


DCERPC BIND/BINDACK:


```
"smb": {
  "id": 53,
  "dialect": "2.10",
  "command": "SMB2_COMMAND_WRITE",
  "status": "STATUS_SUCCESS",
  "status_code": "0x0",
  "session_id": 35184439197745,
  "tree_id": 1,
  "dcerpc": {
    "request": "BIND",
    "response": "BINDACK",
    "interfaces": [
      {
        "uuid": "12345778-1234-abcd-ef00-0123456789ac",
        "version": "1.0",
        "ack_result": 2,
        "ack_reason": 0
      },
      {
        "uuid": "12345778-1234-abcd-ef00-0123456789ac",
        "version": "1.0",
        "ack_result": 0,
        "ack_reason": 0
      },
      {
        "uuid": "12345778-1234-abcd-ef00-0123456789ac",
        "version": "1.0",
        "ack_result": 3,
```


(continues on next page)

(continued from previous page)

```
        "ack_reason": 0
      }
    ],
    "call_id": 2
  }
```

### NTLMSSP fields

- "domain" (string): the Windows domain.
- "user" (string): the user.
- "host" (string): the host.

Example:

```
"ntlmssp": {
  "domain": "VNET3",
  "user": "administrator",
  "host": "BLU"
}
```

More complete example:

```
"smb": {
  "id": 3,
  "dialect": "NT LM 0.12",
  "command": "SMB1_COMMAND_SESSION_SETUP_ANDX",
  "status": "STATUS_SUCCESS",
  "status_code": "0x0",
  "session_id": 2048,
  "tree_id": 0,
  "ntlmssp": {
    "domain": "VNET3",
    "user": "administrator",
    "host": "BLU"
  },
  "request": {
    "native_os": "Unix",
    "native_lm": "Samba 3.9.0-SVN-build-11572"
  },
  "response": {
    "native_os": "Windows (TM) Code Name \"Longhorn\" Ultimate 5231",
    "native_lm": "Windows (TM) Code Name \"Longhorn\" Ultimate 6.0"
  }
}
```

### Kerberos fields

- "kerberos.realm" (string): the Kerberos Realm.
- "kerberos.snames (array of strings): snames.

Example:

```
"smb": {
  "dialect": "2.10",
  "command": "SMB2_COMMAND_SESSION_SETUP",
  "status": "STATUS_SUCCESS",
  "status_code": "0x0",
  "session_id": 35184439197745,
  "tree_id": 0,
  "kerberos": {
    "realm": "CONTOSO.LOCAL",
    "snames": [
      "cifs",
      "DC1.contoso.local"
    ]
  }
}
```

### Event type: SSH

### Fields

- "proto_version": The protocol version transported with the ssh protocol (1.x, 2.x)
- "software_version": The software version used by end user

Example of SSH logging:

```
"ssh": {
  "client": {
      "proto_version": "2.0",
      "software_version": "OpenSSH_6.7",
   },
  "server": {
      "proto_version": "2.0",
      "software_version": "OpenSSH_6.7",
   }
}
```

### Event type: Flow

### Fields

- "pkts_toserver": total number of packets to server, include bypassed packets
- "pkts_toclient": total number of packets to client
- "bytes_toserver": total bytes count to server
- "bytes_toclient": total bytes count to client
- "bypassed.pkts_toserver": number of bypassed packets to server
- "bypassed.pkts_toclient": number of bypassed packets to client
- "bypassed.bytes_toserver": bypassed bytes count to server
- "bypassed.bytes_toclient": bypassed bytes count to client
- "start": date of start of the flow

- "end": date of end of flow (last seen packet)
- "age": duration of the flow
- "bypass": if the flow has been bypassed, it is set to "local" (internal bypass) or "capture"
- "state": display state of the flow (include "new", "established", "closed", "bypassed")
- "reason": mechanism that did trigger the end of the flow (include "timeout", "forced" and "shutdown")
- "alerted": "true" or "false" depending if an alert has been seen on flow

Example


```
"flow": {
  "pkts_toserver": 23,
  "pkts_toclient": 21,
  "bytes_toserver": 4884,
  "bytes_toclient": 7392,
  "bypassed": {
    "pkts_toserver": 10,
    "pkts_toclient": 8,
    "bytes_toserver": 1305,
    "bytes_toclient": 984
  },
  "start": "2019-05-28T23:32:29.025256+0200",
  "end": "2019-05-28T23:35:28.071281+0200",
  "age": 179,
  "bypass": "capture",
  "state": "bypassed",
  "reason": "timeout",
  "alerted": false
}
```


### Event type: RDP

Initial negotiations between RDP client and server are stored as transactions and logged.

Each RDP record contains a per-flow incrementing "tx_id" field.

The "event_type" field indicates an RDP event subtype. Possible values:

- "initial_request"
- "initial_response"
- "connect_request"
- "connect_response"
- "tls_handshake"

### RDP type: Initial Request

The optional "cookie" field is a string identifier the RDP client has chosen to provide.

The optional "flags" field is a list of client directives. Possible values:

- "restricted_admin_mode_required"
- "redirected_authentication_mode_required"

- "correlation_info_present"

### RDP type: Initial Response

In the event of a standard initial response:

The "protocol" field is the selected protocol. Possible values:

- "rdp"
- "ssl"
- "hybrid"
- "rds_tls"
- "hybrid_ex"

The optional "flags" field is a list of support server modes. Possible values:

- "extended_client_data"
- "dynvc_gfx"
- "restricted_admin"
- "redirected_authentication"

Alternatively, in the event of an error-indicating initial response:

There will be no "protocol" or "flags" fields.

The "error_code" field will contain the numeric code provided by the RDP server.

The "reason" field will contain a text summary of this code. Possible values:

- "ssl required by server" (error code 0x1)
- "ssl not allowed by server" (error code 0x2)
- "ssl cert not on server" (error code 0x3)
- "inconsistent flags" (error code 0x4)
- "hybrid required by server" (error code 0x5)
- "ssl with user auth required by server" (error code 0x6)

### RDP type: Connect Request

The optional "channel" field is a list of requested data channel names.

Common channels:

- "rdpdr" (device redirection)
- "cliprdr" (shared clipboard)
- "rdpsnd" (sound)

The optional "client" field is a sub-object that may contain the following:

- "version": RDP protocol version. Possible values are "v4", "v5", "v10.0", "v10.1", "v10.2", "v10.3", "v10.4", "v10.5", "v10.6", "v10.7", "unknown".
- "desktop_width": Numeric desktop width value.

- "desktop_height": Numeric desktop height value.
- "color_depth": Numeric color depth. Possible values are 4, 8, 15, 16, 24.
- "keyboard_layout": Locale identifier name, e.g., "en-US".
- "build": OS and SP level, e.g., "Windows XP", "Windows 7 SP1".
- "client_name": Client computer name.
- "keyboard_type": Possible values are "xt", "ico", "at", "enhanced", "1050", "9140", "jp".
- "keyboard_subtype": Numeric code for keyboard.
- "function_keys": Number of function keys on client keyboard.
- "ime": Input method editor (IME) file name.
- "product_id": Product id string.
- "serial_number": Numeric value.
- "capabilities": List of any of the following: "support_errinfo_pdf", "want_32bpp_session", "support_statusinfo_pdu", "strong_asymmetric_keys", "valid_connection_type", "support_monitor_layout_pdu", "support_netchar_autodetect", "support_dynvc_gfx_protocol", "support_dynamic_time_zone", "support_heartbeat_pdu".
- "id": Client product id string.
- "connection_hint": Possible values are "modem", "low_broadband", "satellite", "high_broadband", "wan", "lan", "autodetect".
- "physical_width": Numeric phyical width of display.
- "physical_height": Numeric physical height of display.
- "desktop_orientation": Numeric angle of orientation.
- "scale_factor": Numeric scale factor of desktop.
- "device_scale_factor": Numeric scale factor of display.

#### RDP type: Connect Response

With this event, the initial RDP negotiation is complete in terms of tracking and logging.

#### RDP type: TLS Handshake

With this event, the initial RDP negotiation is complete in terms of tracking and logging.

The session will use TLS encryption.

The "x509_serials" field is a list of observed certificate serial numbers, e.g., "16ed2aa0495f259d4f5d99edada570d1".

#### Examples

RDP logging:

```
"rdp": {
  "tx_id": 0,
  "event_type": "initial_request",
  "cookie": "A70067"
}

"rdp": {
  "tx_id": 1,
  "event_type": "initial_response"
}

"rdp": {
  "tx_id": 2,
  "event_type": "connect_request",
  "client": {
    "version": "v5",
    "desktop_width": 1152,
    "desktop_height": 864,
    "color_depth": 15,
    "keyboard_layout": "en-US",
    "build": "Windows XP",
    "client_name": "ISD2-KM84178",
    "keyboard_type": "enhanced",
    "function_keys": 12,
    "product_id": 1,
    "capabilities": [
      "support_errinfo_pdf"
    ],
    "id": "55274-OEM-0011903-00107"
  },
  "channels": [
    "rdpdr",
    "cliprdr",
    "rdpsnd"
  ]
}

"rdp": {
  "tx_id": 3,
  "event_type": "connect_response"
}
```

RDP logging, with transition to TLS:

```
"rdp": {
  "tx_id": 0,
  "event_type": "initial_request",
  "cookie": "AWAKECODI"
}

"rdp": {
  "tx_id": 1,
  "event_type": "initial_response",
  "server_supports": [
    "extended_client_data"
  ],
  "protocol": "hybrid"
```

(continues on next page)

(continued from previous page)

```
}

"rdp": {
  "tx_id": 2,
  "event_type": "tls_handshake",
  "x509_serials": [
    "16ed2aa0495f259d4f5d99edada570d1"
  ]
}
```

### 15.1.3 Eve JSON 'jq' Examples

The jq tool is very useful for quickly parsing and filtering JSON files. This page is contains various examples of how it can be used with Suricata's Eve.json.

The basics are discussed here:

- https://www.stamus-networks.com/2015/05/18/looking-at-suricata-json-events-on-command-line/

#### Colorize output

```
tail -f eve.json | jq -c '.'
```

#### DNS NXDOMAIN

```
tail -f eve.json|jq -c 'select(.dns.rcode=="NXDOMAIN")'
```

#### Unique HTTP User Agents

```
cat eve.json | jq -s '[.[]|.http.http_user_agent]|group_by(.)|map({key:.[0],value:(.
↪|length)})|from_entries'
```

Source: https://twitter.com/mattarnao/status/601807374647750657

#### Data use for a host

```
tail -n500000 eve.json | jq -s 'map(select(.event_type=="netflow" and .dest_ip=="192.
↪168.1.3").netflow.bytes)|add'|numfmt --to=iec
1.3G
```

Note: can use a lot of memory. Source: https://twitter.com/pkt_inspector/status/605524218722148352

#### Monitor part of the stats

```
$ tail -f eve.json | jq -c 'select(.event_type=="stats")|.stats.decoder'
```

#### Inspect Alert Data

```
cat eve.json | jq -r -c 'select(.event_type=="alert")|.payload'|base64 --decode
```

#### Top 10 Destination Ports

```
cat eve.json | jq -c 'select(.event_type=="flow")|[.proto, .dest_port]'|sort |uniq -
↪c|sort -nr|head -n10
```

## 15.2 Lua Output

Suricata offers the possibility to get more detailed output on specific kinds of network traffic via pluggable lua scripts. You can write these scripts yourself and only need to define four hook functions.

For lua output scripts suricata offers a wide range of lua functions. They all return information on specific engine internals and aspects of the network traffic. They are described in the following sections, grouped by the event/traffic type. But let's start with a example explaining the four hook functions, and how to make suricata load a lua output script.

### 15.2.1 Script structure

A lua output script needs to define 4 hook functions: init(), setup(), log(), deinit()

- init() – registers where the script hooks into the output engine
- setup() – does per output thread setup
- log() – logging function
- deinit() – clean up function

Example:

```
function init (args)
    local needs = {}
    needs["protocol"] = "http"
    return needs
end

function setup (args)
    filename = SCLogPath() .. "/" .. name
    file = assert(io.open(filename, "a"))
    SCLogInfo("HTTP Log Filename " .. filename)
    http = 0
end

function log(args)
    http_uri = HttpGetRequestUriRaw()
    if http_uri == nil then
        http_uri = "<unknown>"
    end
    http_uri = string.gsub(http_uri, "%c", ".")
```

(continues on next page)

(continued from previous page)

```
    http_host = HttpGetRequestHost()
    if http_host == nil then
        http_host = "<hostname unknown>"
    end
    http_host = string.gsub(http_host, "%c", ".")

    http_ua = HttpGetRequestHeader("User-Agent")
    if http_ua == nil then
        http_ua = "<useragent unknown>"
    end
    http_ua = string.gsub(http_ua, "%g", ".")

    timestring = SCPacketTimeString()
    ip_version, src_ip, dst_ip, protocol, src_port, dst_port = SCFlowTuple()

    file:write (timestring .. " " .. http_host .. " [**] " .. http_uri .. " [**] " ..
           http_ua .. " [**] " .. src_ip .. ":" .. src_port .. " -> " ..
           dst_ip .. ":" .. dst_port .. "\n")
    file:flush()

    http = http + 1
end

function deinit (args)
    SCLogInfo ("HTTP transactions logged: " .. http);
    file:close(file)
end
```

### 15.2.2 YAML

To enable the lua output, add the 'lua' output and add one or more scripts like so:

```
outputs:
  - lua:
      enabled: yes
      scripts-dir: /etc/suricata/lua-output/
      scripts:
        - tcp-data.lua
        - flow.lua
```

The scripts-dir option is optional. It makes Suricata load the scripts from this directory. Otherwise scripts will be loaded from the current workdir.

### 15.2.3 Developing lua output script

You can use functions described in *Lua Functions*

## 15.3 Syslog Alerting Compatibility

Suricata can alert via sylog which is a very handy feature for central log collection, compliance, and reporting to a SIEM. Instructions on setting this up can be found in the .yaml file in the section where you can configure what type of alert (and other) logging you would like.

However, there are different syslog daemons and there can be parsing issues with the syslog format a SIEM expects and what syslog format Suricata sends. The syslog format from Suricata is dependent on the syslog daemon running on the Suricata sensor but often the format it sends is not the format the SIEM expects and cannot parse it properly.

### 15.3.1 Popular syslog daemons

- **syslogd** - logs system messages
- **syslog-ng** - logs system messages but also supports TCP, TLS, and other enhanced enterprise features
- **rsyslogd** - logs system messages but also support TCP, TLS, multi-threading, and other enhanced features
- **klogd** - logs kernel messages
- **sysklogd** - basically a bundle of syslogd and klogd

If the syslog format the Suricata sensor is sending is not compatible with what your SIEM or syslog collector expects, you will need to fix this. You can do this on your SIEM if it is capable of being able to be configured to interpret the message, or by configuring the syslog daemon on the Suricata sensor itself to send in a format you SIEM can parse. The latter can be done by applying a template to your syslog config file.

### 15.3.2 Finding what syslog daemon you are using

There are many ways to find out what syslog daemon you are using but here is one way:

```
cd /etc/init.d
ls | grep syslog
```

You should see a file with the word syslog in it, e.g. "syslog", "rsyslogd", etc. Obviously if the name is "rsyslogd" you can be fairly confident you are running rsyslogd. If unsure or the filename is just "syslog", take a look at that file. For example, if it was "rsyslogd", run:

```
less rsyslogd
```

At the top you should see a comment line that looks something like this:

```
# rsyslog        Starts rsyslogd/rklogd.
```

Locate those files and look at them to give you clues as to what syslog daemon you are running. Also look in the *start()* section of the file you ran "less" on and see what binaries get started because that can give you clues as well.

### 15.3.3 Example

Here is an example where the Suricata sensor is sending syslog messages in rsyslogd format but the SIEM is expecting and parsing them in a sysklogd format. In the syslog configuration file (usually in /etc with a filename like rsyslog.conf or syslog.conf), first add the template:

```
$template sysklogd, "<%PRI%>%syslogtag:1:32%%msg:::sp-if-no-1st-sp%%msg%"
```

Then send it to the syslog server with the template applied:

```
user.alert @10.8.75.24:514;sysklogd
```

Of course this is just one example and it will probably be different in your environment depending on what syslog daemons and SIEM you use but hopefully this will point you in the right direction.

## 15.4 Custom http logging

As of Suricata 1.3.1 you can enable a custom http logging option.

In your Suricata.yaml, find the http-log section and edit as follows:

```
- http-log:
      enabled: yes
      filename: http.log
      custom: yes # enable the custom logging format (defined by custom format)
      customformat: "%{%D-%H:%M:%S}t.%z %{X-Forwarded-For}i %{User-agent}i %H %m %h
↪%u %s %B %a:%p -> %A:%P"
      append: no
      #extended: yes     # enable this for extended logging information
      #filetype: regular # 'regular', 'unix_stream' or 'unix_dgram'
```

And in your http.log file you would get the following, for example:

```
8/28/12-22:14:21.101619 - Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:11.0) Gecko/
↪20100101 Firefox/11.0  HTTP/1.1 GET us.cnn.com /video/data/3.0/video/world/2012/08/
↪28/hancocks-korea-typhoon-bolavan.cnn/index.xml 200 16856 192.168.1.91:45111 -> 157.
↪166.255.18:80
```

```
08/28/12-22:14:30.693856 - Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:11.0) Gecko/
↪20100101 Firefox/11.0  HTTP/1.1 GET us.cnn.com /video/data/3.0/video/showbiz/2012/
↪08/28/conan-reports-from-rnc-convention.teamcoco/index.xml 200 15789 192.168.1.
↪91:45108 -> 157.166.255.18:80
```

The list of supported format strings is the following:

- %h - Host HTTP Header (remote host name). ie: google.com
- %H - Request Protocol. ie: HTTP/1.1
- %m - Request Method. ie: GET
- %u - URL including query string. ie: /search?q=suricata
- %{header_name}i - contents of the defined HTTP Request Header name. ie:
- %{User-agent}i: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:11.0) Gecko/20100101 Firefox/11.0
- %{X-Forwarded-For}i: outputs the IP address contained in the X-Forwarded-For HTTP header (inserted by a reverse proxy)
- %s - return status code. In the case of 301 and 302 it will print the url in brackets. ie: 200
- %B - response size in bytes. ie: 15789
- %{header_name}o - contents of the defined HTTP Response Header name
- %{strftime_format]t - timestamp of the HTTP transaction in the selected strftime format. ie: 08/28/12-22:14:30
- %z - precision time in useconds. ie: 693856
- %a - client IP address
- %p - client port number
- %A - server IP address
- %P - server port number

Any non printable character will be represented by its byte value in hexadecimal format (|XX|, where XX is the hex code)

## 15.5 Custom tls logging

In your Suricata.yaml, find the tls-log section and edit as follows:

```
- tls-log:
    enabled: yes       # Log TLS connections.
    filename: tls.log # File to store TLS logs.
    append: yes
    custom: yes        # enabled the custom logging format (defined by customformat)
    customformat: "%{%D-%H:%M:%S}t.%z %a:%p -> %A:%P %v %n %d %D"
```

And in your tls.log file you would get the following, for example:

```
12/03/16-19:20:14.85859 10.10.10.4:58274 -> 192.0.78.24:443 VERSION='TLS 1.2'␣
↪suricata-ids.org NOTBEFORE='2016-10-27T20:36:00' NOTAFTER='2017-01-25T20:36:00'
```

:: 12/03/16-19:20:20.36849 10.10.10.4:39472 -> 192.30.253.113:443 VERSION='TLS 1.2' github.com NOTBEFORE='2016-03-10T00:00:00' NOTAFTER='2018-05-17T12:00:00'

The list of supported format strings is the following:

- %n - client SNI
- %v - TLS/SSL version
- %d - certificate date not before
- %D - certificate date not after
- %f - certificate fingerprint SHA1
- %s - certificate subject
- %i - certificate issuer dn
- %E - extended format
- %{strftime_format}t - timestamp of the TLS transaction in the selected strftime format. ie: 08/28/12-22:14:30
- %z - precision time in useconds. ie: 693856
- %a - client IP address
- %p - client port number
- %A - server IP address
- %P - server port number

Any non printable character will be represented by its byte value in hexadecimal format (|XX|, where XX is the hex code)

## 15.6 Log Rotation

All outputs in the *outputs* section of the configuration file can be subject to log rotation.

For most outputs an external tool like *logrotate* is required to rotate the log files in combination with sending a SIGHUP to Suricata to notify it that the log files have been rotated.

On receipt of a SIGHUP, Suricata simply closes all open log files and then re-opens them in append mode. If the external tool has renamed any of the log files, new files will be created, otherwise the files will be re-opened and new data will be appended to them with no noticeable affect.

The following is an example *logrotate* configuration file that will rotate Suricata log files then send Suricata a SIGHUP triggering Suricata to open new files:

```
/var/log/suricata/*.log /var/log/suricata/*.json
{
    rotate 3
    missingok
    nocompress
    create
    sharedscripts
    postrotate
            /bin/kill -HUP `cat /var/run/suricata.pid 2>/dev/null` 2>/dev/null || true
    endscript
}
```

**Note:** The above *logrotate* configuration file depends on the existence of a Suricata PID file. If running in daemon mode a PID file will be created by default, otherwise the `--pidfile` option should be used to create a PID file.

In addition to the SIGHUP style rotation discussed above, some outputs support their own time and date based rotation, however removal of old log files is still the responsibility of external tools. These outputs include:

- *Eve*
- *Unified2*
- *PCAP log*

CHAPTER 16

# Lua support

## 16.1 Lua usage in Suricata

Lua scripting can be used in two components of Suricata. The first is in output an the second one in rules in the detection engine.

Both features are using a list of functions to access to data extracted by Suricata. You can get the list of functions in the *Lua functions* page.

### 16.1.1 Lua output

Lua can be used to write arbitrary output. See *Lua Output* for more information.

### 16.1.2 Lua detection

Lua script can be used as a filter condition in signatures. See *Lua Scripting* for more information.

## 16.2 Lua functions

### 16.2.1 packet

Initialize with:

```
function init (args)
    local needs = {}
    needs["type"] = "packet"
    return needs
end
```

#### SCPacketTimestamp

Get packets timestamp as 2 numbers: seconds & microseconds elapsed since 1970-01-01 00:00:00 UTC.

```
function log(args)
    local sec, usec = SCPacketTimestamp()
end
```

#### SCPacketTimeString

Add SCPacketTimeString to get the packets time string in the format: 11/24/2009-18:57:25.179869

```
function log(args)
    ts = SCPacketTimeString()
```

#### SCPacketTuple

```
ipver, srcip, dstip, proto, sp, dp = SCPacketTuple()
```

#### SCPacketPayload

```
p = SCPacketPayload()
```

### 16.2.2 flow

```
function init (args)
    local needs = {}
    needs["type"] = "flow"
    return needs
end
```

#### SCFlowTimestamps

Get timestamps (seconds and microseconds) of the first and the last packet from the flow.

```
startts, lastts = SCFlowTimestamps()
startts_s, lastts_s, startts_us, lastts_us = SCFlowTimestamps()
```

#### SCFlowTimeString

```
startts = SCFlowTimeString()
```

#### SCFlowTuple

```
ipver, srcip, dstip, proto, sp, dp = SCFlowTuple()
```

#### SCFlowAppLayerProto

Get alprotos as string from the flow. If a alproto is not (yet) known, it returns "unknown".

Example:

```
function log(args)
    alproto = SCFlowAppLayerProto()
    if alproto ~= nil then
        print (alproto)
    end
end
```

Returns 5 values: <alproto> <alproto_ts> <alproto_tc> <alproto_orig> <alproto_expect>

Orig and expect are used when changing and upgrading protocols. In a SMTP STARTTLS case, orig would normally be set to "smtp" and expect to "tls".

#### SCFlowHasAlerts

Returns true if flow has alerts.

Example:

```
function log(args)
    has_alerts = SCFlowHasAlerts()
    if has_alerts then
        -- do something
    end
end
```

#### SCFlowStats

Gets the packet and byte counts per flow.

```
tscnt, tsbytes, tccnt, tcbytes = SCFlowStats()
```

#### SCFlowId

Gets the flow id.

```
id = SCFlowId()
```

Note that simply printing 'id' will likely result in printing a scientific notation. To avoid that, simply do:

```
id = SCFlowId()
idstr = string.format("%.0f",id)
print ("Flow ID: " .. idstr .. "\n")
```

### 16.2.3 http

Init with:

```
function init (args)
    local needs = {}
    needs["protocol"] = "http"
    return needs
end
```

### HttpGetRequestBody and HttpGetResponseBody.

Make normalized body data available to the script through HttpGetRequestBody and HttpGetResponseBody.

There no guarantees that all of the body will be availble.

Example:

```
function log(args)
    a, o, e = HttpGetResponseBody();
    --print("offset " .. o .. " end " .. e)
    for n, v in ipairs(a) do
        print(v)
    end
end
```

### HttpGetRequestHost

Get the host from libhtp's tx->request_hostname, which can either be the host portion of the url or the host portion of the Host header.

Example:

```
http_host = HttpGetRequestHost()
if http_host == nil then
    http_host = "<hostname unknown>"
end
```

### HttpGetRequestHeader

```
http_ua = HttpGetRequestHeader("User-Agent")
if http_ua == nil then
    http_ua = "<useragent unknown>"
end
```

### HttpGetResponseHeader

```
server = HttpGetResponseHeader("Server");
print ("Server: " .. server);
```

### HttpGetRequestLine

```
rl = HttpGetRequestLine();
print ("Request Line: " .. rl);
```

#### HttpGetResponseLine

```
rsl = HttpGetResponseLine();
print ("Response Line: " .. rsl);
```

#### HttpGetRawRequestHeaders

```
rh = HttpGetRawRequestHeaders();
print ("Raw Request Headers: " .. rh);
```

#### HttpGetRawResponseHeaders

```
rh = HttpGetRawResponseHeaders();
print ("Raw Response Headers: " .. rh);
```

#### HttpGetRequestUriRaw

```
http_uri = HttpGetRequestUriRaw()
if http_uri == nil then
    http_uri = "<unknown>"
end
```

#### HttpGetRequestUriNormalized

```
http_uri = HttpGetRequestUriNormalized()
if http_uri == nil then
    http_uri = "<unknown>"
end
```

#### HttpGetRequestHeaders

```
a = HttpGetRequestHeaders();
for n, v in pairs(a) do
    print(n,v)
end
```

#### HttpGetResponseHeaders

```
a = HttpGetResponseHeaders();
for n, v in pairs(a) do
    print(n,v)
end
```

### 16.2.4 DNS

#### DnsGetQueries

```
dns_query = DnsGetQueries();
if dns_query ~= nil then
    for n, t in pairs(dns_query) do
        rrname = t["rrname"]
        rrtype = t["type"]

        print ("QUERY: " .. ts .. " " .. rrname .. " [**] " .. rrtype .. " [**] " ..
               "TODO" .. " [**] " .. srcip .. ":" .. sp .. " -> " ..
               dstip .. ":" .. dp)
    end
end
```

returns a table of tables

#### DnsGetAnswers

```
dns_answers = DnsGetAnswers();
if dns_answers ~= nil then
    for n, t in pairs(dns_answers) do
        rrname = t["rrname"]
        rrtype = t["type"]
        ttl = t["ttl"]

        print ("ANSWER: " .. ts .. " " .. rrname .. " [**] " .. rrtype .. " [**] " ..
               ttl .. " [**] " .. srcip .. ":" .. sp .. " -> " ..
               dstip .. ":" .. dp)
    end
end
```

returns a table of tables

#### DnsGetAuthorities

```
dns_auth = DnsGetAuthorities();
if dns_auth ~= nil then
    for n, t in pairs(dns_auth) do
        rrname = t["rrname"]
        rrtype = t["type"]
        ttl = t["ttl"]

        print ("AUTHORITY: " .. ts .. " " .. rrname .. " [**] " .. rrtype .. " [**] "␣
↪..
               ttl .. " [**] " .. srcip .. ":" .. sp .. " -> " ..
               dstip .. ":" .. dp)
    end
end
```

returns a table of tables

#### DnsGetRcode

```
rcode = DnsGetRcode();
if rcode == nil then
    return 0
end
print (rcode)
```

returns a lua string with the error message, or nil

#### DnsGetRecursionDesired

```
if DnsGetRecursionDesired() == true then
    print ("RECURSION DESIRED")
end
```

returns a bool

### 16.2.5 TLS

Initialize with:

```
function init (args)
    local needs = {}
    needs["protocol"] = "tls"
    return needs
end
```

#### TlsGetVersion

Get the negotiated version in a TLS session as a string through TlsGetVersion.

Example:

```
function log (args)
    version = TlsGetVersion()
    if version then
        -- do something
    end
end
```

#### TlsGetCertInfo

Make certificate information available to the script through TlsGetCertInfo.

Example:

```
function log (args)
    version, subject, issuer, fingerprint = TlsGetCertInfo()
    if version == nil then
        return 0
    end
end
```

### TlsGetCertChain

Make certificate chain available to the script through TlsGetCertChain.

The output is an array of certificate with each certificate being an hash with *data* and *length* keys.

Example:

```
-- Use debian lua-luaossl coming from https://github.com/wahern/luaossl
local x509 = require"openssl.x509"

   chain = TlsGetCertChain()
   for k, v in pairs(chain) do
      -- v.length is length of data
      -- v.data is raw binary data of certificate
      cert = x509.new(v["data"], "DER")
      print(cert:text() .. "\n")
   end
```

### TlsGetCertNotAfter

Get the Unix timestamp of end of validity of certificate.

Example:

```
function log (args)
    notafter = TlsGetCertNotAfter()
    if notafter < os.time() then
        -- expired certificate
    end
end
```

### TlsGetCertNotBefore

Get the Unix timestamp of beginning of validity of certificate.

Example:

```
function log (args)
    notbefore = TlsGetCertNotBefore()
    if notbefore > os.time() then
        -- not yet valid certificate
    end
end
```

### TlsGetCertSerial

Get TLS certificate serial number through TlsGetCertSerial.

Example:

```
function log (args)
    serial = TlsGetCertSerial()
    if serial then
        -- do something
```

(continues on next page)

(continued from previous page)

```
    end
end
```

#### TlsGetSNI

Get the Server name Indication from a TLS connection.

Example:

```
function log (args)
    asked_domain = TlsGetSNI()
    if string.find(asked_domain, "badguys") then
        -- ok connection to bad guys let's do someting
    end
end
```

### 16.2.6 JA3

JA3 must be enabled in the Suricata config file (set 'app-layer.protocols.tls.ja3-fingerprints' to 'yes').

Initialize with:

```
function init (args)
    local needs = {}
    needs["protocol"] = "tls"
    return needs
end
```

#### Ja3GetHash

Get the JA3 hash (md5sum of JA3 string) through Ja3GetHash.

Example:

```
function log (args)
    hash = Ja3GetHash()
    if hash == nil then
        return
    end
end
```

#### Ja3GetString

Get the JA3 string through Ja3GetString.

Example:

```
function log (args)
    str = Ja3GetString()
    if str == nil then
        return
```

(continues on next page)

(continued from previous page)

```
    end
end
```

#### Ja3SGetHash

Get the JA3S hash (md5sum of JA3S string) through JA3SGetHash.

Example:

```
function log (args)
    hash = Ja3SGetHash()
    if hash == nil then
        return
    end
end
```

#### JA3SGetString

Get the JA3S string through Ja3SGetString.

Example:

```
function log (args)
    str = Ja3SGetString()
    if str == nil then
        return
    end
end
```

### 16.2.7 SSH

Initialize with:

```
function init (args)
    local needs = {}
    needs["protocol"] = "ssh"
    return needs
end
```

#### SshGetServerProtoVersion

Get SSH protocol version used by the server through SshGetServerProtoVersion.

Example:

```
function log (args)
    version = SshGetServerProtoVersion()
    if version == nil then
        return 0
    end
end
```

#### SshGetServerSoftwareVersion

Get SSH software used by the server through SshGetServerSoftwareVersion.

Example:

```
function log (args)
    software = SshGetServerSoftwareVersion()
    if software == nil then
        return 0
    end
end
```

#### SshGetClientProtoVersion

Get SSH protocol version used by the client through SshGetClientProtoVersion.

Example:

```
function log (args)
    version = SshGetClientProtoVersion()
    if version == nil then
        return 0
    end
end
```

#### SshGetClientSoftwareVersion

Get SSH software used by the client through SshGetClientSoftwareVersion.

Example:

```
function log (args)
    software = SshGetClientSoftwareVersion()
    if software == nil then
        return 0
    end
end
```

### 16.2.8 Files

To use the file logging API, the script's init() function needs to look like:

```
function init (args)
    local needs = {}
    needs['type'] = 'file'
    return needs
end
```

#### SCFileInfo

```
fileid, txid, name, size, magic, md5 = SCFileInfo()
```

returns fileid (number), txid (number), name (string), size (number), magic (string), md5 in hex (string)

#### SCFileState

```
state, stored = SCFileState()
```

returns state (string), stored (bool)

### 16.2.9 Alerts

Alerts are a subset of the 'packet' logger:

```
function init (args)
    local needs = {}
    needs["type"] = "packet"
    needs["filter"] = "alerts"
    return needs
end
```

#### SCRuleIds

```
sid, rev, gid = SCRuleIds()
```

#### SCRuleMsg

```
msg = SCRuleMsg()
```

#### SCRuleClass

```
class, prio = SCRuleClass()
```

### 16.2.10 Streaming Data

Streaming data can currently log out reassembled TCP data and normalized HTTP data. The script will be invoked for each consecutive data chunk.

In case of TCP reassembled data, all possible overlaps are removed according to the host OS settings.

```
function init (args)
    local needs = {}
    needs["type"] = "streaming"
    needs["filter"] = "tcp"
    return needs
end
```

In case of HTTP body data, the bodies are unzipped and dechunked if applicable.

```
function init (args)
    local needs = {}
    needs["type"] = "streaming"
    needs["protocol"] = "http"
    return needs
end
```

#### SCStreamingBuffer

```
function log(args)
    data = SCStreamingBuffer()
    hex_dump(data)
end
```

### 16.2.11 Flow variables

It is possible to access, define and modify Flow variables from Lua. To do so, you must use the functions described in this section and declare the counter in init function:

```
function init(args)
    local needs = {}
    needs["tls"] tostring(true)
    needs["flowint"] = {"tls-cnt"}
    return needs
end
```

Here we define a *tls-cnt* Flowint that can now be used in output or in a signature via dedicted functions. The access to the Flow variable is done by index so in our case we need to use 0.

```
function match(args)
    a = ScFlowintGet(0);
    if a then
        ScFlowintSet(0, a + 1)
    else
        ScFlowintSet(0, 1)
    end
```

#### ScFlowintGet

Get the Flowint at index given by the parameter.

#### ScFlowintSet

Set the Flowint at index given by the first parameter. The second parameter is the value.

#### ScFlowintIncr

Increment Flowint at index given by the first parameter.

#### ScFlowintDecr

Decrement Flowint at index given by the first parameter.

#### ScFlowvarGet

Get the Flowvar at index given by the parameter.

#### ScFlowvarSet

Set a Flowvar. First parameter is the index, second is the data and third is the length of data.

You can use it to set string

```
function init (args)
    local needs = {}
    needs["http.request_headers"] = tostring(true)
    needs["flowvar"] = {"cnt"}
    return needs
end

function match(args)
    a = ScFlowvarGet(0);
    if a then
        a = tostring(tonumber(a)+1)
        ScFlowvarSet(0, a, #a)
    else
        a = tostring(1)
        ScFlowvarSet(0, a, #a)
    end
```

### 16.2.12 Misc

#### SCThreadInfo

```
tid, tname, tgroup = SCThreadInfo()
```

It gives: tid (integer), tname (string), tgroup (string)

#### SCLogError, SCLogWarning, SCLogNotice, SCLogInfo, SCLogDebug

Print a message. It will go into the outputs defined in the yaml. Whether it will be printed depends on the log level.

Example:

```
SCLogError("some error message")
```

#### SCLogPath

Expose the log path.

```
name = "fast_lua.log"
function setup (args)
    filename = SCLogPath() .. "/" .. name
    file = assert(io.open(filename, "a"))
end
```

CHAPTER 17

# File Extraction

## 17.1 Architecture

The file extraction code works on top of the some protocols parsers. The application layer parsers runs on top of the stream reassembly engine and the UDP flow tracking.

In case of HTTP, the parser takes care of dechunking and unzipping the request and/or response data if necessary.

This means that settings in the stream engine, reassembly engine and the application layer parsers all affect the workings of the file extraction.

What files are actually extracted and stored to disk is controlled by the rule language.

Supported protocols are:

- HTTP
- SMTP
- FTP
- NFS
- SMB

## 17.2 Settings

*stream.checksum_validation* controls whether or not the stream engine rejects packets with invalid checksums. A good idea normally, but the network interface performs checksum offloading a lot of packets may seem to be broken. This setting is enabled by default, and can be disabled by setting to "no". Note that the checksum handling can be controlled per interface, see "checksum_checks" in example configuration.

*file-store.stream-depth* controls how far into a stream reassembly is done. Beyond this value no reassembly will be done. This means that after this value the HTTP session will no longer be tracked. By default a settings of 1 Megabyte is used. 0 sets it to unlimited. If set to no, it is disabled and stream.reassembly.depth is considered.

*libhtp.default-config.request-body-limit* / *libhtp.server-config.<config>.request-body-limit* controls how much of the HTTP request body is tracked for inspection by the http_client_body keyword, but also used to limit file inspection. A value of 0 means unlimited.

*libhtp.default-config.response-body-limit* / *libhtp.server-config.<config>.response-body-limit* is like the request body limit, only it applies to the HTTP response body.

## 17.3 Output

### 17.3.1 File-Store and Eve Fileinfo

There are two output modules for logging information about files extracted. The first is `eve.files` which is an `eve` sub-logger that logs `fileinfo` records. These `fileinfo` records provide metadata about the file, but not the actual file contents.

This must be enabled in the `eve` output:

```
- outputs:
    - eve-log:
        types:
          - files:
              force-magic: no
              force-hash: [md5,sha256]
```

See *Eve (Extensible Event Format)* for more details on working with the *eve* output.

The other output module, `file-store` stores the actual files to disk.

The `file-store` uses its own log directory (default: *filestore* in the default logging directory) and logs files using the SHA256 of the contents as the filename. Each file is then placed in a directory named *00* to *ff* where the directory shares the first 2 characters of the filename. For example, if the SHA256 hex string of an extracted file starts with "f9bc6d..." the file we be placed in the directory *filestore/f9*.

The size of a file that can be stored depends on `file-store.stream-depth`, if this value is reached a file can be truncated and might not be stored completely. If not enabled, `stream.reassembly.depth` will be considered.

Setting `file-store.stream-depth` to 0 permits to store any files.

`file-store.stream-depth` will always override `stream.reassembly.depth` when filestore keyword is used.

A protocol parser, like modbus, could permit to set a different store-depth value and use it rather than `file-store.stream-depth`.

Using the SHA256 for file names allows for automatic de-duplication of extracted files. However, the timestamp of a pre-existing file will be updated if the same files is extracted again, similar to the *touch* command.

Optionally a `fileinfo` record can be written to its own file sharing the same SHA256 as the file it references. To handle recording the metadata of each occurrence of an extracted file, these filenames include some extra fields to ensure uniqueness. Currently the format is:

```
<SHA256>.<SECONDS>.<ID>.json
```

where `<SECONDS>` is the seconds from the packet that triggered the stored file to be closed and `<ID>` is a unique ID for the runtime of the Suricata instance. These values should not be depended on, and are simply used to ensure uniqueness.

These `fileinfo` records are idential to the `fileinfo` records logged to the `eve` output.

See *File-store (File Extraction)* for more information on configuring the file-store output.

**Note:** This section documents version 2 of the `file-store`.

### 17.3.2 File-Store (Version 1)

**Note:** File-store version 1 has been deprecated and will be removed by June 2020. Please use file-store v2 instead. Please see the deprecation policy for more information.

```
- file-store:
    enabled: yes         # set to yes to enable
    log-dir: files       # directory to store the files
    force-magic: no      # force logging magic on all stored files
    force-hash: [md5]    # force logging of md5 checksums
    force-filestore: no # force storing of all files
    stream-depth: 1mb    # reassemble 1mb into a stream, set to no to disable
    waldo: file.waldo    # waldo file to store the file_id across runs
    max-open-files: 0    # how many files to keep open (0 means none)
    write-meta: yes      # write a .meta file if set to yes
    include-pid: yes     # include the pid in filenames if set to yes.
```

Each file that is stored will have a name "file.<id>". The id will be reset and files will be overwritten unless the waldo option is used. A "file.<id>.meta" file is generated containing file metadata if write-meta is set to yes (default). If the include-pid option is set, the files will instead have a name "file.<pid>.<id>", and metafiles will be "file.<pid>.<id>.meta". Files will additionally have the suffix ".tmp" while they are open, which is only removed when they are finalized.

## 17.4 Rules

Without rules in place no extraction will happen. The simplest rule would be:

```
alert http any any -> any any (msg:"FILE store all"; filestore; sid:1; rev:1;)
```

This will simply store all files to disk.

Want to store all files with a pdf extension?

```
alert http any any -> any any (msg:"FILE PDF file claimed"; fileext:"pdf"; filestore;␣
↪sid:2; rev:1;)
```

Or rather all actual pdf files?

```
alert http any any -> any any (msg:"FILE pdf detected"; filemagic:"PDF document";␣
↪filestore; sid:3; rev:1;)
```

Or rather only store files from black list checksum md5 ?

```
alert http any any -> any any (msg:"Black list checksum match and extract MD5";␣
↪filemd5:fileextraction-chksum.list; filestore; sid:4; rev:1;)
```

Or only store files from black list checksum sha1 ?

```
alert http any any -> any any (msg:"Black list checksum match and extract SHA1";␣
↪filesha1:fileextraction-chksum.list; filestore; sid:5; rev:1;)
```

Or finally store files from black list checksum sha256 ?

```
alert http any any -> any any (msg:"Black list checksum match and extract SHA256";␣
↪filesha256:fileextraction-chksum.list; filestore; sid:6; rev:1;)
```

Bundled with the Suricata download is a file with more example rules. In the archive, go to the rules/ directory and check the files.rules file.

## 17.5 MD5

Suricata can calculate MD5 checksums of files on the fly and log them. See *Storing MD5s checksums* for an explanation on how to enable this.

### 17.5.1 Storing MD5s checksums

#### Configuration

In the suricata yaml:

```
- file-store:
    enabled: yes       # set to yes to enable
    log-dir: files     # directory to store the files
    force-magic: yes   # force logging magic on all stored files
    force-hash: [md5]  # force logging of md5 checksums
    #waldo: file.waldo # waldo file to store the file_id across runs
```

For JSON output:

```
outputs:
  - eve-log:
    enabled: yes
    filetype: regular #regular|syslog|unix_dgram|unix_stream|redis
    filename: eve.json
    types:
      - files:
        force-magic: no   # force logging magic on all logged files
        # force logging of checksums, available hash functions are md5,
        # sha1 and sha256
        #force-hash: [md5]
```

Other settings affecting *File Extraction*

```
stream:
  memcap: 64mb
  checksum-validation: yes      # reject wrong csums
  inline: no                    # no inline mode
  reassembly:
    memcap: 32mb
    depth: 0                    # reassemble all of a stream
    toserver-chunk-size: 2560
    toclient-chunk-size: 2560
```

Make sure we have *depth: 0* so all files can be tracked fully.

```
libhtp:
  default-config:
    personality: IDS
    # Can be specified in kb, mb, gb.  Just a number indicates
    # it's in bytes.
    request-body-limit: 0
    response-body-limit: 0
```

Make sure we have *request-body-limit: 0* and *response-body-limit: 0*

### Testing

For the purpose of testing we use this rule only in a file.rules (a test/example file):

```
alert http any any -> any any (msg:"FILE store all"; filestore; sid:1; rev:1;)
```

This rule above will save all the file data for files that are opened/downloaded through HTTP

Start Suricata (-S option loads ONLY the specified rule file, with disregard if any other rules that are enabled in suricata.yaml):

```
suricata -c /etc/suricata/suricata.yaml -S file.rules -i eth0
```

Meta data:

```
TIME:              05/01/2012-11:09:52.425751
SRC IP:            2.23.144.170
DST IP:            192.168.1.91
PROTO:             6
SRC PORT:          80
DST PORT:          51598
HTTP URI:          /en/US/prod/collateral/routers/ps5855/prod_
↪brochure0900aecd8019dc1f.pdf
HTTP HOST:         www.cisco.com
HTTP REFERER:      http://www.cisco.com/c/en/us/products/routers/3800-series-
↪integrated-services-routers-isr/index.html
FILENAME:          /en/US/prod/collateral/routers/ps5855/prod_
↪brochure0900aecd8019dc1f.pdf
MAGIC:             PDF document, version 1.6
STATE:             CLOSED
MD5:               59eba188e52467adc11bf2442ee5bf57
SIZE:              9485123
```

and in files-json.log (or eve.json) :

```
{ "id": 1, "timestamp": "05\/01\/2012-11:10:27.693583", "ipver": 4, "srcip": "2.23.
↪144.170", "dstip": "192.168.1.91", "protocol": 6, "sp": 80, "dp": 51598, "http_uri
↪": "\/en\/US\/prod\/collateral\/routers\/ps5855\/prod_brochure0900aecd8019dc1f.pdf",
↪ "http_host": "www.cisco.com", "http_referer": "http:\/\/www.google.com\/url?sa=t&
↪rct=j&q=&esrc=s&source=web&cd=1&ved=0CDAQFjAA&url=http%3A%2F%2Fwww.cisco.com%2Fen
↪%2FUS%2Fprod%2Fcollateral%2Frouters%2Fps5855%2Fprod_brochure0900aecd8019dc1f.pdf&
↪ei=OqyfT9eoJubi4QTyiamhAw&usg=AFQjCNGdjDBpBDfQv2r3VogSH41V6T5x9Q", "filename": "\/
↪en\/US\/prod\/collateral\/routers\/ps5855\/prod_brochure0900aecd8019dc1f.pdf",
↪"magic": "PDF document, version 1.6", "state": "CLOSED", "md5":
↪"59eba188e52467adc11bf2442ee5bf57", "stored": true, "size": 9485123 }
```

(continues on next page)

(continued from previous page)


```
{ "id": 12, "timestamp": "05\/01\/2012-11:12:57.421420", "ipver": 4, "srcip": "2.23.
↪144.170", "dstip": "192.168.1.91", "protocol": 6, "sp": 80, "dp": 51598, "http_uri
↪": "\/en\/US\/prod\/collateral\/routers\/ps5855\/prod_brochure0900aecd8019dc1f.pdf",
↪ "http_host": "www.cisco.com", "http_referer": "http:\/\/www.google.com\/url?sa=t&
↪rct=j&q=&esrc=s&source=web&cd=1&ved=0CDAQFjAA&url=http%3A%2F%2Fwww.cisco.com%2Fen
↪%2FUS%2Fprod%2Fcollateral%2Frouters%2Fps5855%2Fprod_brochure0900aecd8019dc1f.pdf&
↪ei=OqyfT9eoJubi4QTyiamhAw&usg=AFQjCNGdjDBpBDfQv2r3VogSH41V6T5x9Q", "filename": "\/
↪en\/US\/prod\/collateral\/routers\/ps5855\/prod_brochure0900aecd8019dc1f.pdf",
↪"magic": "PDF document, version 1.6", "state": "CLOSED", "md5":
↪"59eba188e52467adc11bf2442ee5bf57", "stored": true, "size": 9485123 }
```


#### Log all MD5s without any rules

If you would like to log MD5s for everything and anything that passes through the traffic that you are inspecting with Suricata, but not log the files themselves, all you have to do is disable file-store and enable only the JSON output with forced MD5s - in suricata.yaml like so:

```
- file-store:
    enabled: no       # set to yes to enable
    log-dir: files    # directory to store the files
    force-magic: yes   # force logging magic on all stored files
    force-hash: [md5]  # force logging of md5 checksums
    #waldo: file.waldo # waldo file to store the file_id across runs

- file-log:
    enabled: yes
    filename: files-json.log
    append: no
    #filetype: regular # 'regular', 'unix_stream' or 'unix_dgram'
    force-magic: yes   # force logging magic on all logged files
    force-hash: [md5]  # force logging of md5 checksums
```

### 17.5.2 Public SHA1 MD5 data sets

National Software Reference Library - http://www.nsrl.nist.gov/Downloads.htm

# CHAPTER 18

## Public Data Sets

Collections of pcaps for testing, profiling.

DARPA sets: http://www.ll.mit.edu/mission/communications/cyber/CSTcorpora/ideval/data/

MAWI sets (pkt headers only, no payloads): http://mawi.wide.ad.jp/mawi/samplepoint-F/2012/

MACCDC: http://www.netresec.com/?page=MACCDC

Netresec: http://www.netresec.com/?page=PcapFiles

Wireshark: https://wiki.wireshark.org/SampleCaptures

Security Onion collection: https://github.com/security-onion-solutions/security-onion/wiki/Pcaps

Stratosphere IPS. Malware Capture Facility Project: https://stratosphereips.org/category/dataset.html

# CHAPTER 19

# Using Capture Hardware

## 19.1 Endace DAG

Suricata comes with native Endace DAG card support. This means Suricata can use the *libdag* interface directly, instead of a libpcap wrapper (which should also work).

Steps:

Configure with DAG support:

```
./configure --enable-dag --prefix=/usr --sysconfdir=/etc --localstatedir=/var
make
sudo make install
```

Results in:

```
Suricata Configuration:
  AF_PACKET support:                       no
  PF_RING support:                         no
  NFQueue support:                         no
  IPFW support:                            no
  DAG enabled:                             yes
  Napatech enabled:                        no
```

Start with:

```
suricata -c suricata.yaml --dag 0:0
```

Started up!

```
[5570] 10/7/2012 -- 13:52:30 - (source-erf-dag.c:262) <Info>␣
↪(ReceiveErfDagThreadInit) -- Attached and started stream: 0 on DAG: /dev/dag0
[5570] 10/7/2012 -- 13:52:30 - (source-erf-dag.c:288) <Info>␣
↪(ReceiveErfDagThreadInit) -- Starting processing packets from stream: 0 on DAG: /
↪dev/dag0
```

## 19.2 Napatech

### 19.2.1 Contents

- Introduction
- Package Installation
- Basic Configuration
- Advanced Multithreaded Configuration

### 19.2.2 Introduction

Napatech packet capture accelerator cards can greatly improve the performance of your Suricata deployment using these hardware based features:

- On board burst buffering (up to 12GB)
- Zero-copy kernel bypass DMA
- Non-blocking PCIe performance
- Port merging
- Load distribution to up 128 host buffers
- Precise timestamping
- Accurate time synchronization

The package uses a proprietary shell script to handle the installation process. In either case, gcc, make and the kernel header files are required to compile the kernel module and install the software.

### 19.2.3 Package Installation

*Note that make, gcc, and the kernel headers are required for installation*

*Root privileges are also required*

The latest driver and tools installation package can be downloaded from: https://www.napatech.com/downloads.

*Note that you will be prompted to install the Napatech libpcap library. Answer "yes" if you would like to use the Napatech card to capture packets in Wireshark, tcpdump, or another pcap based application. Libpcap is not needed for Suricata as native Napatech API support is included*

Red Hat Based Distros:

```
$ yum install kernel-devel-$(uname -r) gcc make
       $ ./package_install_3gd.sh
```

Debian Based Distros:

```
$ apt-get install linux-headers-$(uname .r) gcc make
       $ ./package_install_3gd.sh
```

To complete installation for all distros ntservice:

```
$ /opt/napatech3/bin/ntstart.sh -m
```

### 19.2.4 Suricata Installation

After downloading and extracting the Suricata tarball, you need to run configure to enable Napatech support and prepare for compilation:

```
$ ./configure --enable-napatech --with-napatech-includes=/opt/napatech3/include --
↪with-napatech-libraries=/opt/napatech3/lib
$ make
$ make install-full
```

### 19.2.5 Suricata configuration

Now edit the suricata.yaml file to configure the system. There are three ways the system can be configured:

1. Auto-config without cpu-affinity: In this mode you specify the stream configuration in suricata.yaml file and allow the threads to roam freely. This is good for single processor systems where NUMA node configuration is not a performance concern.

2. Auto-config with cpu-affinity: In this mode you use the cpu-affinity of the worker threads to control the creation and configuration of streams. One stream and one worker thread will be created for each cpu identified in suricata.yaml. This is best in systems with multiple NUMA nodes (i.e. multi-processor systems) as the NUMA node of the host buffers is matched to the core on which the thread is running.

3. Manual-config (legacy): In this mode the underlying Napatech streams are configured by issuing NTPL commands prior to running suricata. Suricata then connects to the existing streams on startup.

### 19.2.6 Example Configuration - Auto-config without cpu-affinity:

If cpu-affinity is not used it is necessary to explicitly define the streams in the Suricata configuration file. To use this option the following options should be set in the suricata configuration file:

1. Turn off cpu-affinity
2. Enable the napatech "auto-config" option
3. Specify the streams that should be created on startup
4. Specify the ports that will provide traffic to suricata
5. Specify the hashmode used to distribute traffic to the streams

Below are the options to set:

```
threading:
  set-cpu-affinity: no
    .
    .
    .
napatech:
    auto-config: yes
    streams: ["0-3"]
    ports: [all]
    hashmode: hash5tuplesorted
```

Now modify ntservice.ini. You also need make sure that you have allocated enough host buffers in ntservice.ini for the streams. It's a good idea to also set the TimeSyncReferencePriority. To do this make the following changes to ntservice.ini:

HostBuffersRx = [4,16,-1] # [number of host buffers, Size(MB), NUMA node] TimeSyncReferencePriority = OSTime # Timestamp clock synchronized to the OS

Stop and restart ntservice after making changes to ntservice:

```
$ /opt/napatech3/bin/ntstop.sh
$ /opt/napatech3/bin/ntstart.sh
```

Now you are ready to start suricata:

```
$ suricata -c /usr/local/etc/suricata/suricata.yaml --napatech --runmode workers
```

### 19.2.7 Example Configuration - Auto-config with cpu-affinity:

This option will create a single worker-thread and stream for each CPU defined in the worker-cpu-set. To use this option make the following changes to suricata.yaml:

1. Turn on cpu-affinity
2. Specify the worker-cpu-set
3. Enable the napatech "auto-config" option
4. Specify the ports that will provide traffic to suricata
5. Specify the hashmode that will be used to control the distribution of traffic to the different streams/cpus.

When you are done it should look similar to this:

```
threading:
  set-cpu-affinity: yes
  cpu-affinity:
    management-cpu-set:
      cpu: [ 0 ]
    receive-cpu-set:
      cpu: [ 0 ]
    worker-cpu-set:
      cpu: [ all ]
      .
      .
      .
napatech:
  auto-config: yes
  ports: [all]
  hashmode: hash5tuplesorted
```

Prior to running Suricata in this mode you also need to configure a sufficient number of host buffers on each NUMA node. So, for example, if you have a two processor server with 32 total cores and you plan to use all of the cores you will need to allocate 16 host buffers on each NUMA node. It is also desirable to set the Napatech cards time source to the OS.

To do this make the following changes to ntservice.ini:

```
TimeSyncReferencePriority = OSTime  # Timestamp clock synchronized to the OS
HostBuffersRx = [16,16,0],[16,16,1] # [number of host buffers, Size(MB), NUMA node]
```

Stop and restart ntservice after making changes to ntservice:

```
$ /opt/napatech3/bin/ntstop.sh -m
$ /opt/napatech3/bin/ntstart.sh -m
```

Now you are ready to start suricata:

```
$ suricata -c /usr/local/etc/suricata/suricata.yaml --napatech --runmode workers
```

### 19.2.8 Example Configuration - Manual Configuration

For Manual Configuration the Napatech streams are created by running NTPL commands prior to running Suricata. In this example we will setup the Napatech capture accelerator to merge all physical ports, and then distribute the merged traffic to four streams that Suricata will ingest.

**The steps for this configuration are:**

1. Disable the napatech auto-config option in suricata.yaml
2. Specify the streams that suricata is to use in suricata.yaml
3. Create a file with NTPL commands to create the underlying Napatech streams.

First suricata.yaml should be configured similar to the following:

```
napatech:
  auto-config: no
  streams: ["0-3"]
```

Next you need to make sure you have enough host buffers defined in ntservice.ini. As it's also a good idea to set up the TimeSync. Here are the lines to change:

```
TimeSyncReferencePriority = OSTime      # Timestamp clock synchronized to the OS
HostBuffersRx = [4,16,-1]               # [number of host buffers, Size(MB), NUMA␣
↪node]
```

Stop and restart ntservice after making changes to ntservice:

```
$ /opt/napatech3/bin/ntstop.sh
$ /opt/napatech3/bin/ntstart.sh
```

Now that ntservice is running we need to execute a few NTPL (Napatech Programming Language) commands to complete the setup. Create a file will the following commands:

```
Delete=All                              # Delete any existing filters
Assign[streamid=(0..3)]= all    # Assign all phisical ports to stream ID 0
```

Next execute those command using the ntpl tool:

```
$ /opt/napatech3/bin/ntpl -f <my_ntpl_file>
```

Now you are ready to start suricata:

```
$ suricata -c /usr/local/etc/suricata/suricata.yaml --napatech --runmode workers
```

It is possible to specify much more elaborate configurations using this option. Simply by creating the appropriate NTPL file and attaching suricata to the streams.

### 19.2.9 Counters

For each stream that is being processed the following counters will be output in stats.log:

- nt<streamid>.pkts - The number of packets recieved by the stream.
- nt<streamid>.bytes - The total bytes received by the stream.
- nt<streamid>.drop - The number of packets that were dropped from this stream due to buffer overflow conditions.

If hba is enabled the following counter will also be provided:

- nt<streamid>.hba_drop - the number of packets dropped because the host buffer allowance high-water mark was reached.

In addition to counters host buffer utilization is tracked and logged. This is also useful for debugging. Log messages are output for both Host and On-Board buffers when reach 25, 50, 75 percent of utilization. Corresponding messages are output when utilization decreases.

Debugging:

For debugging configurations it is useful to see what traffic is flowing as well as what streams are created and receiving traffic. There are two tools in /opt/napatech3/bin that are useful for this:

- monitoring: this tool will, among other things, show what traffic is arriving at the port interfaces.
- profiling: this will show host-buffers, streams and traffic flow to the streams.

If suricata terminates abnormally stream definitions, which are normally removed at shutdown, may remain in effect. If this happens they can be cleared by issuing the "delete=all" NTPL command as follows:

```
# /opt/napatech3/bin/ntpl -e "delete=all"
```

### 19.2.10 Napatech configuration options:

These are the Napatech options available in the suricata configuration file:

```
napatech:
  # The Host Buffer Allowance for all streams
  # (-1 = OFF, 1 - 100 = percentage of the host buffer that can be held back)
  # This may be enabled when sharing streams with another application.
  # Otherwise, it should be turned off.
  #hba: -1

  # When use_all_streams is set to "yes" the initialization code will query
  # the Napatech service for all configured streams and listen on all of them.
  # When set to "no" the streams config array will be used.
  #
  # This option necessitates running the appropriate NTPL commands to create
  # the desired streams prior to running suricata.
  #use-all-streams: no

  # The streams to listen on when cpu-affinity or auto-config is disabled.
  # This can be either:
  #   a list of individual streams (e.g. streams: [0,1,2,3])
  # or
  #   a range of streams (e.g. streams: ["0-3"])
  #
```

(continues on next page)

(continued from previous page)

```
  #streams: ["0-7"]
  #
  # When auto-config is enabled the streams will be created and assigned to the
  # NUMA node where the thread resides automatically. The streams will be created
  # according to the number of worker threads specified in the worker cpu set.
  # (I.e. the value of threading.cpu-affinity.worker-cpu-set.cpu.)
  #
  # This option cannot be used simultaneous with "use-all-streams".
  #
  auto-config: yes
  #
  # Ports indicates which napatech ports are to be used in auto-config mode.
  # these are the port ID's of the ports that will merged prior to the traffic
  # being distributed to the streams.
  #
  # This can be specified in any of the following ways:
  #
  #   a list of individual ports (e.g. ports: [0,1,2,3])
  #
  #   a range of ports (e.g. ports: [0-3])
  #
  #   "all" to indicate that all ports are to be merged together
  #   (e.g. ports: [all])
  #
  # This has no effect if auto-config is disabled.
  #
  ports: [all]
  #
  # When auto-config is enabled the hashmode specifies the algorithm for
  # determining to which stream a given packet is to be delivered.
  # This can be any valid Napatech NTPL hashmode command.
  #
  # The most common hashmode commands are: hash2tuple, hash2tuplesorted,
  # hash5tuple, hash5tuplesorted and roundrobin.
  #
  # See Napatech NTPL documentation other hashmodes and details on their use.
  #
  # This has no effect if auto-config is disabled.
  #
  hashmode: hash5tuplesorted
```

*Note: hba is useful only when a stream is shared with another application. When hba is enabled packets will be dropped (i.e. not delivered to suricata) when the host-buffer utilization reaches the high-water mark indicated by the hba value. This insures that, should suricata get behind in it's packet processing, the other application will still receive all of the packets. If this is enabled without another application sharing the stream it will result in sub-optimal packet buffering.*

Make sure that there are enough host-buffers declared in ntservice.ini to accommodate the number of cores/streams being used.

### 19.2.11 Support

Contact a support engineer at: ntsupport@napatech.com

Napatech Documentation can be found at: https://docs.napatech.com (Click the search icon, with no search text, to see all documents in the portal.)

## 19.3 Myricom

From: https://blog.inliniac.net/2012/07/10/suricata-on-myricom-capture-cards/

In this guide I'll describe using the Myricom libpcap support. I'm going to assume you installed the card properly, installed the Sniffer driver and made sure that all works. Make sure that in your dmesg you see that the card is in sniffer mode:

```
[ 2102.860241] myri_snf INFO: eth4: Link0 is UP
[ 2101.341965] myri_snf INFO: eth5: Link0 is UP
```

I have installed the Myricom runtime and libraries in /opt/snf

Compile Suricata against Myricom's libpcap:

```
./configure --with-libpcap-includes=/opt/snf/include/ --with-libpcap-libraries=/opt/
↪snf/lib/ --prefix=/usr --sysconfdir=/etc --localstatedir=/var
make
sudo make install
```

Next, configure the amount of ringbuffers. I'm going to work with 8 here, as my quad core + hyper threading has 8 logical CPUs. *See below* for additional information about the buffer-size parameter.

```
pcap:
  - interface: eth5
    threads: 8
    buffer-size: 512kb
    checksum-checks: no
```

The 8 threads setting makes Suricata create 8 reader threads for eth5. The Myricom driver makes sure each of those is attached to its own ringbuffer.

Then start Suricata as follows:

```
SNF_NUM_RINGS=8 SNF_FLAGS=0x1 suricata -c suricata.yaml -i eth5 --runmode=workers
```

If you want 16 ringbuffers, update the "threads" variable in your yaml to 16 and start Suricata:

```
SNF_NUM_RINGS=16 SNF_FLAGS=0x1 suricata -c suricata.yaml -i eth5 --runmode=workers
```

Note that the pcap.buffer-size yaml setting shown above is currently ignored when using Myricom cards. The value is passed through to the pcap_set_buffer_size libpcap API within the Suricata source code. From Myricom support:

```
"The libpcap interface to Sniffer10G ignores the pcap_set_buffer_size() value.  The␣
↪call to snf_open() uses zero as the dataring_size which informs the Sniffer library␣
↪to use a default value or the value from the SNF_DATARING_SIZE environment variable.
↪"
```

The following pull request opened by Myricom in the libpcap project indicates that a future SNF software release could provide support for setting the SNF_DATARING_SIZE via the pcap.buffer-size yaml setting:

- https://github.com/the-tcpdump-group/libpcap/pull/435

Until then, the data ring and descriptor ring values can be explicitly set using the SNF_DATARING_SIZE and SNF_DESCRING_SIZE environment variables, respectively.

The SNF_DATARING_SIZE is the total amount of memory to be used for storing incoming packet data. This size is shared across all rings. The SNF_DESCRING_SIZE is the total amount of memory to be used for storing meta information about the packets (packet lengths, offsets, timestamps). This size is also shared across all rings.

Myricom recommends that the descriptor ring be 1/4 the size of the data ring, but the ratio can be modified based on your traffic profile. If not set explicitly, Myricom uses the following default values: SNF_DATARING_SIZE = 256MB, and SNF_DESCRING_SIZE = 64MB

Expanding on the 16 thread example above, you can start Suricata with a 16GB Data Ring and a 4GB Descriptor Ring using the following command:

```
SNF_NUM_RINGS=16 SNF_DATARING_SIZE=17179869184 SNF_DESCRING_SIZE=4294967296 SNF_
↪FLAGS=0x1 suricata -c suricata.yaml -i eth5 --runmode=workers
```

### 19.3.1 Debug Info

Myricom also provides a means for obtaining debug information. This can be useful for verifying your configuration and gathering additional information. Setting SNF_DEBUG_MASK=3 enables debug information, and optionally setting the SNF_DEBUG_FILENAME allows you to specify the location of the output file.

Following through with the example:

```
SNF_NUM_RINGS=16 SNF_DATARING_SIZE=17179869184 SNF_DESCRING_SIZE=4294967296 SNF_
↪FLAGS=0x1 SNF_DEBUG_MASK=3 SNF_DEBUG_FILENAME="/tmp/snf.out" suricata -c suricata.
↪yaml -i eth5 --runmode=workers
```

### 19.3.2 Additional Info

- http://www.40gbe.net/index_files/be59da7f2ab5bf0a299ab99ef441bb2e-28.html
- http://o-www.emulex.com/blogs/implementers/2012/07/23/black-hat-usa-2012-emulex-faststack-sniffer10g-product-demo-emul

## 19.4 eBPF and XDP

### 19.4.1 Introduction

eBPF stands for extended BPF. This is an extended version of Berkeley Packet Filter available in recent Linux kernel versions.

It provides more advanced features with eBPF programs developed in C and capability to use structured data shared between kernel and userspace.

eBPF is used for three things in Suricata:

- eBPF filter: any BPF like filter can be developed. An example of filter accepting only packet for some VLANs is provided. A bypass implementation is also provided.
- eBPF load balancing: provide programmable load balancing. A simple ippair load balancing is provided.
- XDP programs: suricata can load XDP programs. A bypass program is provided.

Bypass can be implemented in eBPF and XDP. The advantage of XDP is that the packets are dropped at the earliest stage possible. So performance is better. But bypassed packets don't reach the network so you can't use this on regular traffic but only on duplicated/sniffed traffic.

The bypass implementation relies on one of the most powerful concept of eBPF: maps. A map is a data structure shared between user space and kernel space/hardware. It allows user space and kernel space to interact, pass information. Maps are often implemented as arrays or hash tables that can contain arbitrary key, value pairs.

#### XDP

XDP provides another Linux native way of optimising Suricata's performance on sniffing high speed networks:

> XDP or eXpress Data Path provides a high performance, programmable network data path in the Linux kernel as part of the IO Visor Project. XDP provides bare metal packet processing at the lowest point in the software stack which makes it ideal for speed without compromising programmability. Furthermore, new functions can be implemented dynamically with the integrated fast path without kernel modification.

More info about XDP:

- IOVisor's XDP page
- Cilium's BPF and XDP reference guide

### 19.4.2 Requirements

You will need a kernel that supports XDP and, for real performance improvement, a network card that support XDP in the driver.

Suricata XDP code has been tested with 4.13.10 but 4.15 or later is necessary to use all features like the CPU redirect map.

If you are using an Intel network card, you will need to stay with in tree kernel NIC drivers. The out of tree drivers do not contain the XDP support.

Having a network card with support for RSS symmetric hashing is a good point or you will have to use the XDP CPU redirect map feature.

### 19.4.3 Prerequisites

This guide has been confirmed on Debian/Ubuntu "LTS" Linux.

#### Disable irqbalance

Irqbalance may cause issues in most setups described here, so it is recommended to deactivate it

```
systemctl stop irqbalance
systemctl disable irqbalance
```

#### Kernel

You need to run a kernel 4.13 or newer.

#### Clang and dependencies

Make sure you have clang (>=3.9) installed on the system

```
sudo apt install clang
```

Some i386 headers will also be needed as eBPF is not x86_64 and some included headers are architecture specific

```
sudo apt install libc6-dev-i386 --no-install-recommends
```

#### libbpf

Suricata uses libbpf to interact with eBPF and XDP

```
git clone https://github.com/libbpf/libbpf.git
```

Now, you can build and install the library

```
cd libbpf/src/
make && sudo make install

sudo make install_headers
sudo ldconfig
```

In some cases your system will not find the libbpf library that is installed under */usr/lib64* so you may need to modify your ldconfig configuration.

### 19.4.4 Compile and install Suricata

To get Suricata source, you can use the usual

```
git clone  https://github.com/OISF/suricata.git
cd suricata && git clone https://github.com/OISF/libhtp.git -b 0.5.x

./autogen.sh
```

Then you need to add the ebpf flags to configure and specify the Clang compiler for building all C sources, including the eBPF programs

```
CC=clang ./configure --prefix=/usr/ --sysconfdir=/etc/ --localstatedir=/var/ \
--enable-ebpf --enable-ebpf-build

make clean && make
sudo  make install-full
sudo ldconfig
sudo mkdir /usr/libexec/suricata/ebpf/
```

The `clang` compiler is needed if you want to build eBPF files as the build is done via a specific eBPF backend available only in llvm/clang suite. If you don't want to use Clang for building Suricata itself, you can still specify it separately, using the `--with-clang` parameter

```
./configure --prefix=/usr/ --sysconfdir=/etc/ --localstatedir=/var/ \
--enable-ebpf --enable-ebpf-build --with-clang=/usr/bin/clang
```

### 19.4.5 Setup bypass

If you plan to use eBPF or XDP for a kernel/hardware level bypass, you need to enable some of the following features:

First, enable *bypass* in the *stream* section in `suricata.yaml`

```
stream:
  bypass: true
```

This will bypass flows as soon as the stream depth will be reached.

If you want, you can also bypass encrypted flows by setting *encryption-handling* to *bypass* in the app-layer tls section

```
app-layer:
  protocols:
    tls:
      enabled: yes
      detection-ports:
        dp: 443

      encryption-handling: bypass
```

Another solution is to use a set of signatures using the `bypass` keyword to obtain a selective bypass. Suricata traffic ID defines flowbits that can be used in other signatures. For instance one could use

```
alert any any -> any any (msg:"bypass video"; flowbits:isset,traffic/label/video;␣
↪noalert; bypass; sid:1000000; rev:1;)
alert any any -> any any (msg:"bypass Skype"; flowbits:isset,traffic/id/skype;␣
↪noalert; bypass; sid:1000001; rev:1;)
```

### 19.4.6 Setup eBPF filter

The file *ebpf/vlan_filter.c* contains a list of vlan id in a switch that you need to edit to get something adapted to your network. Another filter dropping packets from or to a set of IPv4 address is also available in *ebpf/filter.c*. See *Pinned maps usage* for more information.

Suricata can load as eBPF filter any eBPF code exposing a `filter` section.

Once modifications and build via *make* are done, you can copy the resulting eBPF filter as needed

```
cp ebpf/vlan_filter.bpf /usr/libexec/suricata/ebpf/
```

Then setup the *ebpf-filter-file* variable in af-packet section in `suricata.yaml`

```
- interface: eth3
  threads: 16
  cluster-id: 97
  cluster-type: cluster_flow # choose any type suitable
  defrag: yes
  # eBPF file containing a 'filter' function that will be inserted into the
  # kernel and used as load balancing function
  ebpf-filter-file:  /usr/libexec/suricata/ebpf/vlan_filter.bpf
  use-mmap: yes
  ring-size: 200000
```

You can then run suricata normally

```
/usr/bin/suricata --pidfile /var/run/suricata.pid  --af-packet=eth3 -vvv
```

### 19.4.7 Setup eBPF bypass

You can also use eBPF bypass. To do that load the *bypass_filter.bpf* file and update af-packet configuration in `suricata.yaml` to set bypass to yes

```
- interface: eth3
  threads: 16
  cluster-id: 97
```

(continues on next page)

(continued from previous page)

```
  cluster-type: cluster_qm # symmetric RSS hashing is mandatory to use this mode
  # eBPF file containing a 'filter' function that will be inserted into the
  # kernel and used as packet filter function
  ebpf-filter-file:  /usr/libexec/suricata/ebpf/bypass_filter.bpf
  bypass: yes
  use-mmap: yes
  ring-size: 200000
```

Constraints on eBPF code to have a bypass compliant code are stronger than for regular filter. The filter must expose *flow_table_v4* and *flow_table_v6* per CPU array maps with similar definitions as the one available in *bypass_filter.c*. These two maps will be accessed and maintained by Suricata to handle the lists of flow to bypass.

If you are not using vlan tracking (`vlan.use-for-tracking` set to false in suricata.yaml) then you also have to set the VLAN_TRACKING define to 0 in `bypass_filter.c`.

### 19.4.8 Setup eBPF load balancing

eBPF load balancing allows to load balance the traffic on the listening sockets With any logic implemented in the eBPF filter. The value returned by the function tagged with the `loadbalancer` section is used with a modulo on the CPU count to know in which socket the packet has to be send.

An implementation of a simple symetric IP pair hashing function is provided in the `lb.bpf` file.

Copy the resulting eBPF filter as needed

```
cp ebpf/lb.bpf /usr/libexec/suricata/ebpf/
```

Then use `cluster_ebpf` as load balancing method in the interface section of af-packet and point the `ebpf-lb-file` variable to the `lb.bpf` file

```
- interface: eth3
  threads: 16
  cluster-id: 97
  cluster-type: cluster_ebpf
  defrag: yes
  # eBPF file containing a 'loadbalancer' function that will be inserted into the
  # kernel and used as load balancing function
  ebpf-lb-file:  /usr/libexec/suricata/ebpf/lb.bpf
  use-mmap: yes
  ring-size: 200000
```

### 19.4.9 Setup XDP bypass

XDP bypass will allow Suricata to tell the kernel that packets for some flows have to be dropped via the XDP mechanism. This is a really early drop that occurs before the datagram is reaching the Linux kernel network stack.

Linux 4.15 or newer are recommended to use that feature. You can use it on older kernel if you set `BUILD_CPUMAP` to 0 in `ebpf/xdp_filter.c`.

Copy the resulting xdp filter as needed:

```
cp ebpf/xdp_filter.bpf /usr/libexec/suricata/ebpf/
```

Setup af-packet section/interface in `suricata.yaml`.

We will use `cluster_qm` as we have symmetric hashing on the NIC, `xdp-mode: driver` and we will also use the `/usr/libexec/suricata/ebpf/xdp_filter.bpf` (in our example TCP offloading/bypass)

```
- interface: eth3
  threads: 16
  cluster-id: 97
  cluster-type: cluster_qm # symmetric hashing is a must!
  defrag: yes
  # Xdp mode, "soft" for skb based version, "driver" for network card based
  # and "hw" for card supporting eBPF.
  xdp-mode: driver
  xdp-filter-file:  /usr/libexec/suricata/ebpf/xdp_filter.bpf
  # if the ebpf filter implements a bypass function, you can set 'bypass' to
  # yes and benefit from these feature
  bypass: yes
  use-mmap: yes
  ring-size: 200000
  # Uncomment the following if you are using hardware XDP with
  # a card like Netronome (default value is yes)
  # use-percpu-hash: no
```

XDP bypass is compatible with AF_PACKET IPS mode. Packets from bypassed flows will be send directly from one card to the second card without going by the kernel network stack.

If you are using hardware XDP offload you may have to set `use-percpu-hash` to false and build and install the XDP filter file after setting `USE_PERCPU_HASH` to 0.

In the XDP filter file, you can set `ENCRYPTED_TLS_BYPASS` to 1 if you want to bypass the encrypted TLS 1.2 packets in the eBPF code. Be aware that this will mean that Suricata will be blind on packets on port 443 with the correct pattern.

If you are not using vlan tracking (`vlan.use-for-tracking` set to false in suricata.yaml) then you also have to set the VLAN_TRACKING define to 0 in `xdp_filter.c`.

### Intel NIC setup

Intel network card don't support symmetric hashing but it is possible to emulate it by using a specific hashing function.

Follow these instructions closely for desired result:

```
ifconfig eth3 down
```

Use in tree kernel drivers: XDP support is not available in Intel drivers available on Intel website.

Enable symmetric hashing

```
ifconfig eth3 down
ethtool -L eth3 combined 16 # if you have at least 16 cores
ethtool -K eth3 rxhash on
ethtool -K eth3 ntuple on
ifconfig eth3 up
./set_irq_affinity 0-15 eth3
ethtool -X eth3 hkey␣
↪6D:5A:6D:5A:6D:5A:
```

In the above setup you are free to use any recent `set_irq_affinity` script. It is available in any Intel x520/710 NIC sources driver download.

**NOTE:** We use a special low entropy key for the symmetric hashing. More info about the research for symmetric hashing set up

### Disable any NIC offloading

Run the following command to disable offloading

```
for i in rx tx tso ufo gso gro lro tx nocache copy sg txvlan rxvlan; do
       /sbin/ethtool -K eth3 $i off 2>&1 > /dev/null;
done
```

### Balance as much as you can

Try to use the network's card balancing as much as possible

```
for proto in tcp4 udp4 ah4 esp4 sctp4 tcp6 udp6 ah6 esp6 sctp6; do
   /sbin/ethtool -N eth3 rx-flow-hash $proto sd
done
```

This command triggers load balancing using only source and destination IPs. This may be not optimal in term of load balancing fairness but this ensures all packets of a flow will reach the same thread even in the case of IP fragmentation (where source and destination port will not be available for some fragmented packets).

### The XDP CPU redirect case

If ever your hardware is not able to do a symmetric load balancing but support XDP in driver mode, you can then use the CPU redirect map support available in the *xdp_filter.bpf* and *xdp_lb.bpf* file. In this mode, the load balancing will be done by the XDP filter and each CPU will handle the whole packet treatment including the creation of the skb structure in kernel.

You will need Linux 4.15 or newer to use that feature.

To do so set the *xdp-cpu-redirect* variable in af-packet interface configuration to a set of CPUs. Then use the *cluster_cpu* as load balancing function. You will also need to set the affinity to be certain that CPU cores that have the skb assigned are used by Suricata.

Also to avoid out of order packets, you need to set the RSS queue number to 1. So if our interface is *eth3*

```
/sbin/ethtool -L eth3 combined 1
```

In case your system has more then 64 core, you need to set *CPUMAP_MAX_CPUS* to a value greater than this number in *xdp_lb.c* and *xdp_filter.c*.

A sample configuration for pure XDP load balancing could look like

```
- interface: eth3
  threads: 16
  cluster-id: 97
  cluster-type: cluster_cpu
  xdp-mode: driver
  xdp-filter-file:  /usr/libexec/suricata/ebpf/xdp_lb.bpf
  xdp-cpu-redirect: ["1-17"] # or ["all"] to load balance on all CPUs
```

(continues on next page)

(continued from previous page)

```
  use-mmap: yes
  ring-size: 200000
```

It is possible to use *xdp_monitor* to have information about the behavior of CPU redirect. This program is available in Linux tree under the *samples/bpf* directory and will be build by the make command. Sample output is the following

```
sudo ./xdp_monitor --stats
XDP-event       CPU:to  pps           drop-pps     extra-info
XDP_REDIRECT    11      2,880,212     0            Success
XDP_REDIRECT    total   2,880,212     0            Success
XDP_REDIRECT    total   0             0            Error
cpumap-enqueue   11:0   575,954       0            5.27       bulk-average
cpumap-enqueue  sum:0   575,954       0            5.27       bulk-average
cpumap-kthread  0       575,990       0            56,409     sched
cpumap-kthread  1       576,090       0            54,897     sched
```

### Start Suricata with XDP

You can now start Suricata with XDP bypass activated

```
/usr/bin/suricata -c /etc/suricata/xdp-suricata.yaml --pidfile /var/run/suricata.pid␣
↪--af-packet=eth3 -vvv
```

Confirm you have the XDP filter engaged in the output (example):

```
...
...
(runmode-af-packet.c:220) <Config> (ParseAFPConfig) -- Enabling locked memory for␣
↪mmap on iface eth3
(runmode-af-packet.c:231) <Config> (ParseAFPConfig) -- Enabling tpacket v3 capture on␣
↪iface eth3
(runmode-af-packet.c:326) <Config> (ParseAFPConfig) -- Using queue based cluster mode␣
↪for AF_PACKET (iface eth3)
(runmode-af-packet.c:424) <Info> (ParseAFPConfig) -- af-packet will use '/usr/libexec/
↪suricata/ebpf/xdp_filter.bpf' as XDP filter file
(runmode-af-packet.c:429) <Config> (ParseAFPConfig) -- Using bypass kernel␣
↪functionality for AF_PACKET (iface eth3)
(runmode-af-packet.c:609) <Config> (ParseAFPConfig) -- eth3: enabling zero copy mode␣
↪by using data release call
(util-runmodes.c:296) <Info> (RunModeSetLiveCaptureWorkersForDevice) -- Going to use␣
↪8 thread(s)
...
...
```

## 19.4.10 Pinned maps usage

Pinned maps stay attached to the system if the creating process disappears and they can also be accessed by external tools. In Suricata bypass case, this can be used to keep active bypassed flow tables, so Suricata is not hit by previously bypassed flows when restarting. In the socket filter case, this can be used to maintain a map from tools outside of Suricata.

To use pinned maps, you first have to mount the *bpf* pseudo filesystem

```
sudo mount -t bpf none /sys/fs/bpf
```

You can also add to your */etc/fstab*

```
bpffs                       /sys/fs/bpf             bpf     defaults 0 0
```

and run *sudo mount -a*.

Pinned maps will be accessible as file from the */sys/fs/bpf* directory. Suricata will pin them under the name *suricata-$IFACE_NAME-$MAP_NAME*.

To activate pinned maps for a interface, set *pinned-maps* to *true* in the *af-packet* configuration of this interface

```
- interface: eth3
  pinned-maps: true
```

### 19.4.11 XDP and pinned-maps

This option can be used to expose the maps of a socket filter to other processes. This allows for example, the external handling of a accept list or block list of IP addresses. See bpfctrl for an example of external list handling.

In the case of XDP, the eBPF filter is attached to the interface so if you activate *pinned-maps* the eBPF will remain attached to the interface and the maps will remain accessible upon Suricata start. If XDP bypass is activated, Suricata will try at start to open the pinned maps *flow_v4_table* and *flow_v6_table*. If they are present, this means the XDP filter is still there and Suricata will just use them instead of attaching the XDP file to the interface.

So if you want to reload the XDP filter, you need to remove the files from */sys/fs/bpf/* before starting Suricata.

In case, you are not using bypass, this means that the used maps are managed from outside Suricata. As their names are not known by Suricata, you need to specify a name of a map to look for, that will be used to check for the presence of the XDP filter

```
- interface: eth3
  pinned-maps: true
  pinned-maps-name: ipv4_drop
  xdp-filter-file: /usr/libexec/suricata/ebpf/xdp_filter.bpf
```

If XDP bypass is used in IPS mode stopping Suricata will trigger an interruption in the traffic. To fix that, the provided XDP filter *xdp_filter.bpf* is containing a map that will trigger a global bypass if set to 1. You need to use *pinned-maps* to benefit from this feature.

To use it you need to set *#define USE_GLOBAL_BYPASS 1* (instead of 0) in the *xdp_filter.c* file and rebuild the eBPF code and install the eBPF file in the correct place. If you write *1* as key *0* then the XDP filter will switch to global bypass mode. Set key *0* to value *0* to send traffic to Suricata.

The switch must be activated on all sniffing interfaces. For an interface named *eth0* the global switch map will be */sys/fs/bpf/suricata-eth0-global_bypass*.

#### Pinned maps and eBPF filter

Pinned maps can also be used with regular eBPF filters. The main difference is that the map will not persist after Suricata is stopped because it is attached to a socket and not an interface which is persistent.

The eBPF filter *filter.bpf* uses a *ipv4_drop* map that contains the set of IPv4 addresses to drop. If *pinned-maps* is set to *true* in the interface configuration then the map will be pinned under */sys/fs/bpf/suricata-eth3-ipv4_drop*.

You can then use a tool like *bpfctrl* to manage the IPv4 addresses in the map.

### 19.4.12 Hardware bypass with Netronome

Netronome cards support hardware bypass. In this case the eBPF code is running in the card itself. This introduces some architectural differences compared to driver mode and the configuration and eBPF filter need to be updated.

On eBPF side, as of Linux 4.19 CPU maps and interfaces redirect are not supported and these features need to be disabled. By architecture, per CPU hash should not be used and has to be disabled. To achieve this, edit the beginning of *ebpf/xdp_filter.c* and do

```
#define BUILD_CPUMAP        0
/* Increase CPUMAP_MAX_CPUS if ever you have more than 64 CPUs */
#define CPUMAP_MAX_CPUS     64

#define USE_PERCPU_HASH    0
#define GOT_TX_PEER    0
```

Then build the bpf file with *make* and install it in the expected place.

On Suricata configuration side, this is rather simple as you need to activate hardware mode and the *no-percpu-hash* option in the *af-packet* configuration of the interface

```
xdp-mode: hw
no-percpu-hash: true
```

The load balancing will be done on IP pairs inside the eBPF code, so using *cluster_qm* as cluster type is a good idea

```
cluster-type: cluster_qm
```

As of Linux 4.19, the number of threads must be a power of 2. So set *threads* variable of the *af-packet* interface to a power of 2 and in the eBPF filter set the following variable accordingly

```
#define RSS_QUEUE_NUMBERS   32
```

### 19.4.13 Getting live info about bypass

You can get information about bypass via the stats event and through the unix socket. `iface-stat` will return the number of bypassed packets (adding packets for a flow when it timeout)

```
suricatasc -c "iface-stat enp94s0np0" | jq
{
  "message": {
    "pkts": 56529854964,
    "drop": 932328611,
    "bypassed": 1569467248,
    "invalid-checksums": 0
  },
  "return": "OK"
}
```

`iface-bypassed-stats` command will return the number of elements in IPv4 and IPv6 flow tables for each interface

```
# suricatasc
>>> iface-bypassed-stats
Success:
{
```

(continues on next page)

(continued from previous page)

```
    "enp94s0np0": {
       "ipv4_fail": 0,
       "ipv4_maps_count": 2303,
       "ipv4_success": 4232,
       "ipv6_fail": 0,
       "ipv6_maps_count": 13131,
       "ipv6_success": 13500

    }
}
```

The stats entry also contains a *stats.flow_bypassed* object that has local and capture bytes and packets counters as well as a bypassed and closed flow counter

```
{
  "local_pkts": 0,
  "local_bytes": 0,
  "local_capture_pkts": 20,
  "local_capture_bytes": 25000,
  "closed": 84,
  "pkts": 4799,
  "bytes": 2975133
}
```

*local_pkts* and *local_bytes* are for Suricata bypassed flows. This can be because local bypass is used or because the capture method can not bypass more flows. *pkts* and *bytes* are counters coming from the capture method. They can take some time to appear due to the accounting at timeout. *local_capture_pkts* and *local_capture_bytes* are counters for packets that are seen by Suricata before the capture method efficiently bypass the traffic. There is almost always some for each flow because of the buffer in front of Suricata reading threads.

## 19.5 Netmap

Netmap is a high speed capture framework for Linux and FreeBSD. In Linux it is available as an external module, while in FreeBSD 11+ it is available by default.

### 19.5.1 Compiling Suricata

#### FreeBSD

On FreeBSD 11 and up, NETMAP is included and enabled by default in the kernel.

To build Suricata with NETMAP, add `--enable-netmap` to the configure line. The location of the NETMAP includes (/usr/src/sys/net/) does not have to be specified.

#### Linux

On Linux, NETMAP is not included by default. It can be pulled from github. Follow the instructions on installation included in the NETMAP repository.

When NETMAP is installed, add `--enable-netmap` to the configure line. If the includes are not added to a standard location, the location can be specified on the Suricata configure commandline.

Example:

```
./configure --enable-netmap --with-netmap-includes=/usr/local/include/netmap/
```

### 19.5.2 Starting Suricata

When opening an interface, netmap can take various special characters as options in the interface string.

**Warning:** the interface that netmap reads from will become unavailable for normal network operations. You can lock yourself out of your system.

#### IDS

Suricata can be started in 2 ways to use netmap:

```
suricata --netmap=<interface>
suricata --netmap=igb0
```

In the above example Suricata will start reading from igb0. The number of threads created depends on the number of RSS queues available on the NIC.

```
suricata --netmap
```

In the above example Suricata will take the `netmap` block from the yaml and open each of the interfaces listed.

```
netmap:
  - interface: igb0
    threads: 2
  - interface: igb1
    threads: 4
```

For the above configuration, both igb0 and igb1 would be opened. With 2 threads for igb0 and 4 capture threads for igb1.

**Warning:** This multi threaded setup only works correctly if the NIC has symmetric RSS hashing. If this is not the case, consider using the the 'lb' method below.

#### IPS

Suricata's Netmap based IPS mode is based on the concept of creating a layer 2 software bridge between 2 interfaces. Suricata reads packets on one interface and transmits them on another.

Packets that are blocked by the IPS policy, are simply not transmitted.

```
netmap:
  - interface: igb0
    copy-mode: ips
    copy-iface: igb1
  - interface: igb1
    copy-mode: ips
    copy-iface: igb0
```

### 19.5.3 Advanced setups

### 19.5.4 lb (load balance)

"lb" is a tool written by Seth Hall to allow for load balancing for single or multiple tools. One common use case is being able to run Suricata and Zeek together on the same traffic.

starting lb:

```
lb -i eth0 -p suricata:6 -p zeek:6
```

**Note:** On FreeBSD 11, the named prefix doesn't work.

yaml:

```
netmap:
  - interface: suricata
    threads: 6
```

startup:

```
suricata --netmap=netmap:suricata
```

The interface name as passed to Suricata includes a 'netmap:' prefix. This tells Suricata that it's going to read from netmap pipes instead of a real interface.

Then Zeek (formerly Bro) can be configured to load 6 instances. Both will get a copy of the same traffic. The number of netmap pipes does not have to be equal for both tools.

#### FreeBSD 11

On FreeBSD 11 the named pipe is not available.

starting lb:

```
lb -i eth0 -p 6
```

yaml:

```
netmap:
  - interface: netmap:eth0
    threads: 6
```

startup:

```
suricata --netmap
```

**Note:** "lb" is bundled with netmap.

#### Single NIC

When an interface enters NETMAP mode, it is no longer available to the OS for other operations. This can be undesirable in certain cases, but there is a workaround.

By running Suricata in a special inline mode, the interface will show it's traffic to the OS.

```
netmap:
  - interface: igb0
    copy-mode: tap
    copy-iface: igb0^
  - interface: igb0^
    copy-mode: tap
    copy-iface: igb0
```

The copy-mode can be both 'tap' and 'ips', where the former never drops packets based on the policies in use, and the latter may drop packets.

**Warning:** Misconfiguration can lead to connectivity loss. Use with care.

**Note:** This set up can also be used to mix NETMAP with firewall setups like pf or ipfw.

#### VALE switches

VALE is a virtual switch that can be used to create an all virtual network or a mix of virtual and real nics.

A simple all virtual setup:

```
vale-ctl -n vi0
vale-ctl -a vale0:vi0
vale-ctl -n vi1
vale-ctl -a vale0:vi1
```

We now have a virtual switch "vale0" with 2 ports "vi0" and "vi1".

We can start Suricata to listen on one of the ports:

```
suricata --netmap=vale0:vi1
```

Then we can

### 19.5.5 Inline IDS

The inline IDS is almost the same as the IPS setup above, but it will not enfore `drop` policies.

```
netmap:
  - interface: igb0
    copy-mode: tap
    copy-iface: igb1
  - interface: igb1
    copy-mode: tap
    copy-iface: igb0
```

The only difference with the IPS mode is that the `copy-mode` setting is set to `tap`.

CHAPTER 20

# Interacting via Unix Socket

## 20.1 Introduction

Suricata can listen to a unix socket and accept commands from the user. The exchange protocol is JSON-based and the format of the message has been done to be generic.

An example script called suricatasc is provided in the source and installed automatically when installing/updating Suricata.

The unix socket is enabled by default if libjansson is available.

You need to have libjansson installed:

- libjansson4 - C library for encoding, decoding and manipulating JSON data
- libjansson-dev - C library for encoding, decoding and manipulating JSON data (dev)
- python-simplejson - simple, fast, extensible JSON encoder/decoder for Python

Debian/Ubuntu:

```
apt-get install libjansson4 libjansson-dev python-simplejson
```

If libjansson is present on the system , unix socket will be compiled in automatically.

The creation of the socket is managed by setting enabled to 'yes' or 'auto' under unix-command in Suricata YAML configuration file:

```
unix-command:
  enabled: yes
  #filename: custom.socket # use this to specify an alternate file
```

The `filename` variable can be used to set an alternate socket filename. The filename is always relative to the local state base directory.

Clients are implemented for some language and can be used as code example to write custom scripts:

- Python: https://github.com/inliniac/suricata/blob/master/scripts/suricatasc/suricatasc.in (provided with suricata and used in this document)
- Perl: https://github.com/aflab/suricatac (a simple Perl client with interactive mode)
- C: https://github.com/regit/SuricataC (a Unix socket mode client in C without interactive mode)

## 20.2 Commands in standard running mode

You may need to install suricatasc if you have not done so, running the following command from python/suricatasc

```
sudo python setup.py install
```

The set of existing commands is the following:

- command-list: list available commands
- shutdown: this shutdown Suricata
- iface-list: list interfaces where Suricata is sniffing packets
- iface-stat: list statistic for an interface
- help: alias of command-list
- version: display Suricata's version
- uptime: display Suricata's uptime
- running-mode: display running mode (workers, autofp, simple)
- capture-mode: display capture system used
- conf-get: get configuration item (see example below)
- dump-counters: dump Suricata's performance counters
- reopen-log-files: reopen log files (to be run after external log rotation)
- ruleset-reload-rules: reload ruleset and wait for completion
- ruleset-reload-nonblocking: reload ruleset and proceed without waiting
- ruleset-reload-time: return time of last reload
- ruleset-stats: display the number of rules loaded and failed
- ruleset-failed-rules: display the list of failed rules
- memcap-set: update memcap value of an item specified
- memcap-show: show memcap value of an item specified
- memcap-list: list all memcap values available
- reload-rules: alias of ruleset-reload-rules
- register-tenant-handler: register a tenant handler with the specified mapping
- unregister-tenant-handler: unregister a tenant handler with the specified mapping
- register-tenant: register tenant with a particular ID and filename
- unregister-tenant: unregister tenant with a particular ID
- reload-tenant: reload a tenant with specified ID and filename

- add-hostbit: add hostbit on a host IP with a particular bit name and time of expiry
- remove-hostbit: remove hostbit on a host IP with specified bit name
- list-hostbit: list hostbit for a particular host IP

You can access to these commands with the provided example script which is named `suricatasc`. A typical session with `suricatasc` will looks like:

```
# suricatasc
Command list: shutdown, command-list, help, version, uptime, running-mode, capture-
↪mode, conf-get, dump-counters, iface-stat, iface-list, quit
>>> iface-list
Success: {'count': 2, 'ifaces': ['eth0', 'eth1']}
>>> iface-stat eth0
Success: {'pkts': 378, 'drop': 0, 'invalid-checksums': 0}
>>> conf-get unix-command.enabled
Success:
"yes"
```

## 20.3 Commands on the cmd prompt

You can use suricatasc directly on the command prompt:

```
root@debian64:~# suricatasc -c version
{'message': '2.1beta2 RELEASE', 'return': 'OK'}
root@debian64:~#
root@debian64:~# suricatasc -c uptime
{'message': 35264, 'return': 'OK'}
root@debian64:~#
```

**NOTE:** You need to quote commands involving more than one argument:

```
root@debian64:~# suricatasc -c "iface-stat eth0"
{'message': {'pkts': 5110429, 'drop': 0, 'invalid-checksums': 0}, 'return': 'OK'}
root@debian64:~#
```

## 20.4 Pcap processing mode

This mode is one of main motivation behind this code. The idea is to be able to ask to Suricata to treat different pcap files without having to restart Suricata between the files. This provides you a huge gain in time as you don't need to wait for the signature engine to initialize.

To use this mode, start suricata with your preferred YAML file and provide the option `--unix-socket` as argument:

```
suricata -c /etc/suricata-full-sigs.yaml --unix-socket
```

It is also possible to specify the socket filename as argument:

```
suricata --unix-socket=custom.socket
```

In this last case, you will need to provide the complete path to the socket to `suricatasc`. To do so, you need to pass the filename as first argument of `suricatasc`:

```
suricatasc custom.socket
```

Once Suricata is started, you can use the provided script `suricatasc` to connect to the command socket and ask for pcap treatment:

```
root@tiger:~# suricatasc
>>> pcap-file /home/benches/file1.pcap /tmp/file1
Success: Successfully added file to list
>>> pcap-file /home/benches/file2.pcap /tmp/file2
Success: Successfully added file to list
>>> pcap-file-continuous /home/pcaps /tmp/dirout
Success: Successfully added file to list
```

You can add multiple files without waiting the result: they will be sequentially processed and the generated log/alert files will be put into the directory specified as second arguments of the pcap-file command. You need to provide absolute path to the files and directory as Suricata doesn't know from where the script has been run. If you pass a directory instead of a file, all files in the directory will be processed. If using `pcap-file-continuous` and passing in a directory, the directory will be monitored for new files being added until you use `pcap-interrupt` or delete/move the directory.

To know how many files are waiting to get processed, you can do:

```
>>> pcap-file-number
Success: 3
```

To get the list of queued files, do:

```
>>> pcap-file-list
Success: {'count': 2, 'files': ['/home/benches/file1.pcap', '/home/benches/file2.pcap
↪']}
```

To get current processed file:

```
>>> pcap-current
Success:
"/tmp/test.pcap"
```

When passing in a directory, you can see last processed time (modified time of last file) in milliseconds since epoch:

```
>>> pcap-last-processed
Success:
1509138964000
```

To interrupt directory processing which terminates the current state:

```
>>> pcap-interrupt
Success:
"Interrupted"
```

## 20.5 Build your own client

The protocol is documented in the following page https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Unix_Socket#Protocol

The following session show what is send (SND) and received (RCV) by the server. Initial negotiation is the following:

```
# suricatasc
SND: {"version": "0.1"}
RCV: {"return": "OK"}
```

Once this is done, command can be issued:

```
>>> iface-list
SND: {"command": "iface-list"}
RCV: {"message": {"count": 1, "ifaces": ["wlan0"]}, "return": "OK"}
Success: {'count': 1, 'ifaces': ['wlan0']}
>>> iface-stat wlan0
SND: {"command": "iface-stat", "arguments": {"iface": "wlan0"}}
RCV: {"message": {"pkts": 41508, "drop": 0, "invalid-checksums": 0}, "return": "OK"}
Success: {'pkts': 41508, 'drop': 0, 'invalid-checksums': 0}
```

In pcap-file mode, this gives:

```
>>> pcap-file /home/eric/git/oisf/benches/sandnet.pcap /tmp/bench
SND: {"command": "pcap-file", "arguments": {"output-dir": "/tmp/bench", "filename": "/
↪home/eric/git/oisf/benches/sandnet.pcap"}}
RCV: {"message": "Successfully added file to list", "return": "OK"}
Success: Successfully added file to list
>>> pcap-file-number
SND: {"command": "pcap-file-number"}
RCV: {"message": 1, "return": "OK"}
>>> pcap-file-list
SND: {"command": "pcap-file-list"}
RCV: {"message": {"count": 1, "files": ["/home/eric/git/oisf/benches/sandnet.pcap"]},
↪"return": "OK"}
Success: {'count': 1, 'files': ['/home/eric/git/oisf/benches/sandnet.pcap']}
>>> pcap-file-continuous /home/eric/git/oisf/benches /tmp/bench 0 true
SND: {"command": "pcap-file", "arguments": {"output-dir": "/tmp/bench", "filename": "/
↪home/eric/git/oisf/benches/sandnet.pcap", "tenant": 0, "delete-when-done": true}}
RCV: {"message": "Successfully added file to list", "return": "OK"}
Success: Successfully added file to list
```

There is one thing to be careful about: a Suricata message is sent in multiple send operations. This result in possible incomplete read on client side. The worse workaround is to sleep a bit before trying a recv call. An other solution is to use non blocking socket and retry a recv if the previous one has failed.

Pcap-file json format is:

```
{
  "command": "pcap-file",
  "arguments": {
    "output-dir": "path to output dir",
    "filename": "path to file or directory to run",
    "tenant": 0,
    "continuous": false,
    "delete-when-done": false
  }
}
```

*output-dir* and *filename* are required. *tenant* is optional and should be a number, indicating which tenant the file or directory should run under. *continuous* is optional and should be true/false, indicating that file or directory should be run until *pcap-interrupt* is sent or ctrl-c is invoked. *delete-when-done* is optional and should be true/false, indicating that the file or files under the directory specified by *filename* should be deleted when processing is complete. *delete-when-done* defaults to false, indicating files will be kept after processing.

CHAPTER 21

# 3rd Party Integration

## 21.1 Symantec SSL Visibility (BlueCoat)

As Suricata itself cannot decrypt SSL/TLS traffic, some organizations use a decryption product to handle this. This document will offer some advice on using Suricata with the Symantec SSL Visibility appliance (formerly known as BlueCoat).

### 21.1.1 Appliance Software Version

The appliance comes with two major software version options. The 3.x and 4.x series. Suricata works best with the 4.x series.

TLS1.3 is only properly supported in the 4.x version of the appliance software.

### 21.1.2 Magic Markers

The appliance has an indicator that data is decrypted. This is done using a special magic source MAC address, or using a special VLAN header. Since Suricata can use VLANs as part of flow tracking, it is recommended to use the source MAC method.

In the 3.x version of the software these markers are always there, the config just allows setting which type will be used. In the 4.x software the markers are optional.

### 21.1.3 TCP handling

In the 3.x software, a bit of care is required in TCP stream reassembly handling in Suricata. The decrypted traffic is presented to the IDS as TCP data packets, that are not ack'd as regularly as would be expected in a regular TCP session. A large TCP window is used to not violate the TCP specs. Since in IDS mode Suricata waits for ACKs for much of its processing, this can lead to delays in detection and logging, as well as increased resource usage due to increased data buffering.

To avoid this, enable the 'stream.inline' mode, which processed data segments as they come in without waiting for the ACKs.

The 4.x software sends more regular ACKs and does not need any special handling on the Suricata side.

### 21.1.4 TLS matching in Suricata

The appliance takes care of the TLS handling and decryption, presenting only the decrypted data to Suricata. This means that Suricata will not see the TLS handshake. As a consequence of this, Suricata cannot inspect the TLS handshake or otherwise process it. This means that for decrypted TLS sessions, Suricata will not do any TLS keyword inspection (such as fingerprint matching and ja3), TLS logging or TLS certificate extraction.

If it is important to match on and/or log such information as well, the appliance facilities for matching and logging themselves will have to be used.

For TLS traffic where the appliance security policy does not lead to decryption of the traffic, the TLS handshake is presented to Suricata for analysis and logging.

### 21.1.5 IPS

When using Suricata in IPS mode with the appliance, some things will have to be considered:

- if Suricata DROPs a packet in the decrypted traffic, this will be seen by the appliance after which it will trigger a RST session teardown.
- if a packet takes more than one second to process, it will automatically be considered a DROP by the appliance. This should not happen in normal traffic, but with very inefficient Lua scripts this could perhaps happen. The appliance can also be configured to wait for 5 seconds.
- When using the Suricata 'replace' keyword to modify data, be aware that the 3.x appliance software will not pass the modification on to the destination so this will not have any effect. The 4.x appliance software does support passing on modifications that were made to the unencrypted text, by default this feature is disabled but you can enable it if you want modifications to be passed on to the destination in the re-encrypted stream. Due to how Suricata works, the size of the payloads cannot be changed.

CHAPTER 22

# Man Pages

## 22.1 Suricata

### 22.1.1 SYNOPSIS

**suricata** [OPTIONS] [BPF FILTER]

### 22.1.2 DESCRIPTION

Suricata is a high performance Network IDS, IPS and Network Security Monitoring engine. Open Source and owned by a community run non-profit foundation, the Open Information Security Foundation (OISF).

### 22.1.3 OPTIONS

**-h**
: Display a brief usage overview.

**-V**
: Displays the version of Suricata.

**-c** <path>
: Path to configuration file.

**-T**
: Test configuration.

**-v**
: Increase the verbosity of the Suricata application logging by increasing the log level from the default. This option can be passed multiple times to further increase the verbosity.

  - -v: INFO
  - -vv: PERF

- -vvv: CONFIG
- -vvvv: DEBUG

This option will not decrease the log level set in the configuration file if it is already more verbose than the level requested with this option.

**-r** <path>
Run in pcap offline mode (replay mode) reading files from pcap file. If <path> specifies a directory, all files in that directory will be processed in order of modified time maintaining flow state between files.

**--pcap-file-continuous**
Used with the -r option to indicate that the mode should stay alive until interrupted. This is useful with directories to add new files and not reset flow state between files.

**--pcap-file-delete**
Used with the -r option to indicate that the mode should delete pcap files after they have been processed. This is useful with pcap-file-continuous to continuously feed files to a directory and have them cleaned up when done. If this option is not set, pcap files will not be deleted after processing.

**-i** <interface>
After the -i option you can enter the interface card you would like to use to sniff packets from. This option will try to use the best capture method available. Can be used several times to sniff packets from several interfaces.

**--pcap[**=<device>]
Run in PCAP mode. If no device is provided the interfaces provided in the *pcap* section of the configuration file will be used.

**--af-packet[**=<device>]
Enable capture of packet using AF_PACKET on Linux. If no device is supplied, the list of devices from the af-packet section in the yaml is used.

**-q** <queue id>
Run inline of the NFQUEUE queue ID provided. May be provided multiple times.

**-s** <filename.rules>
With the -s option you can set a file with signatures, which will be loaded together with the rules set in the yaml.

**-S** <filename.rules>
With the -S option you can set a file with signatures, which will be loaded exclusively, regardless of the rules set in the yaml.

**-l** <directory>
With the -l option you can set the default log directory. If you already have the default-log-dir set in yaml, it will not be used by Suricata if you use the -l option. It will use the log dir that is set with the -l option. If you do not set a directory with the -l option, Suricata will use the directory that is set in yaml.

**-D**
Normally if you run Suricata on your console, it keeps your console occupied. You can not use it for other purposes, and when you close the window, Suricata stops running. If you run Suricata as daemon (using the -D option), it runs at the background and you will be able to use the console for other tasks without disturbing the engine running.

**--runmode** <runmode>
With the *–runmode* option you can set the runmode that you would like to use. This command line option can override the yaml runmode option.

Runmodes are: *workers*, *autofp* and *single*.

For more information about runmodes see *Runmodes* in the user guide.

**-F** <bpf filter file>
Use BPF filter from file.

**-k** [all|none]
Force (all) the checksum check or disable (none) all checksum checks.

**--user**=<user>
Set the process user after initialization. Overrides the user provided in the *run-as* section of the configuration file.

**--group**=<group>
Set the process group to group after initialization. Overrides the group provided in the *run-as* section of the configuration file.

**--pidfile** <file>
Write the process ID to file. Overrides the *pid-file* option in the configuration file and forces the file to be written when not running as a daemon.

**--init-errors-fatal**
Exit with a failure when errors are encountered loading signatures.

**--disable-detection**
Disable the detection engine.

**--dump-config**
Dump the configuration loaded from the configuration file to the terminal and exit.

**--build-info**
Display the build information the Suricata was built with.

**--list-app-layer-protos**
List all supported application layer protocols.

**--list-keywords**=[all|csv|<kword>]
List all supported rule keywords.

**--list-runmodes**
List all supported run modes.

**--set** <key>=<value>
Set a configuration value. Useful for overriding basic configuration parameters in the configuration. For example, to change the default log directory:

```
--set default-log-dir=/var/tmp
```

**--engine-analysis**
Print reports on analysis of different sections in the engine and exit. Please have a look at the conf parameter engine-analysis on what reports can be printed

**--unix-socket**=<file>
Use file as the Suricata unix control socket. Overrides the *filename* provided in the *unix-command* section of the configuration file.

**--pcap-buffer-size**=<size>
Set the size of the PCAP buffer (0 - 2147483647).

**--netmap[**=<device>]
Enable capture of packet using NETMAP on FreeBSD or Linux. If no device is supplied, the list of devices from the netmap section in the yaml is used.

**--pfring[**=<device>]
Enable PF_RING packet capture. If no device provided, the devices in the Suricata configuration will be used.

**--pfring-cluster-id** <id>
Set the PF_RING cluster ID.

**--pfring-cluster-type** <type>
Set the PF_RING cluster type (cluster_round_robin, cluster_flow).

**-d** <divert-port>
Run inline using IPFW divert mode.

**--dag** <device>
Enable packet capture off a DAG card. If capturing off a specific stream the stream can be select using a device name like "dag0:4". This option may be provided multiple times read off multiple devices and/or streams.

**--napatech**
Enable packet capture using the Napatech Streams API.

**--erf-in**=<file>
Run in offline mode reading the specific ERF file (Endace extensible record format).

**--simulate-ips**
Simulate IPS mode when running in a non-IPS mode.

### 22.1.4 OPTIONS FOR DEVELOPERS

**-u**
Run the unit tests and exit. Requires that Suricata be compiled with *–enable-unittests*.

**-U, --unittest-filter**=REGEX
With the -U option you can select which of the unit tests you want to run. This option uses REGEX. Example of use: suricata -u -U http

**--list-unittests**
List all unit tests.

**--fatal-unittests**
Enables fatal failure on a unit test error. Suricata will exit instead of continuing more tests.

**--unittests-coverage**
Display unit test coverage report.

### 22.1.5 SIGNALS

Suricata will respond to the following signals:

**SIGUSR2** Causes Suricata to perform a live rule reload.

**SIGHUP** Causes Suricata to close and re-open all log files. This can be used to re-open log files after they may have been moved away by log rotation utilities.

### 22.1.6 FILES AND DIRECTORIES

**/etc/suricata/suricata.yaml** Default location of the Suricata configuration file.

**/var/log/suricata** Default Suricata log directory.

### 22.1.7 BUGS

Please visit Suricata's support page for information about submitting bugs or feature requests.

### 22.1.8 NOTES

- Suricata Home Page

  https://suricata-ids.org/

- Suricata Support Page

  https://suricata-ids.org/support/

## 22.2 Suricata Socket Control

### 22.2.1 SYNOPSIS

**suricatasc**

### 22.2.2 DESCRIPTION

Suricata socket control tool

### 22.2.3 COMMANDS

**shutdown**
Shut Suricata instance down.

**command-list**
List available commands.

**help**
Get help about the available commands.

**version**
Print the version of Suricata instance.

**uptime**
Display the uptime of Suricata.

**running-mode**
Display running mode. This can either be *workers*, *autofp* or *single*.

**capture-mode**
Display the capture mode. This can be either of *PCAP_DEV*, *PCAP_FILE*, *PFRING(DISABLED)*, *NFQ*, *NFLOG*, *IPFW*, *ERF_FILE*, *ERF_DAG*, *AF_PACKET_DEV*, *NETMAP(DISABLED)*, *UNIX_SOCKET* or *WINDIVERT(DISABLED)*.

**conf-get <variable>**
Get configuration value for a given variable. Variable to be provided can be either of the configuration parameters that are written in suricata.yaml.

**dump-counters**
Dump Suricata's performance counters.

**ruleset-reload-rules**
Reload the ruleset and wait for completion.

**reload-rules**
Alias .. describe *ruleset-reload-rules*.

**ruleset-reload-nonblocking**
Reload ruleset and proceed without waiting.

**ruleset-reload-time**
Return time of last reload.

**ruleset-stats**
Display the number of rules loaded and failed.

**ruleset-failed-rules**
Display the list of failed rules.

**register-tenant-handler <id> <htype> [hargs]**
Register a tenant handler with the specified mapping.

**unregister-tenant-handler <id> <htype> [hargs]**
Unregister a tenant handler with the specified mapping.

**register-tenant <id> <filename>**
Register tenant with a particular ID and filename.

**reload-tenant <id> <filename>**
Reload a tenant with specified ID and filename.

**unregister-tenant <id>**
Unregister tenant with a particular ID.

**add-hostbit <ipaddress> <hostbit> <expire>**
Add hostbit on a host IP with a particular bit name and time of expiry.

**remove-hostbit <ipaddress> <hostbit>**
Remove hostbit on a host IP with specified IP address and bit name.

**list-hostbit <ipaddress>**
List hostbit for a particular host IP.

**reopen-log-files**
Reopen log files to be run after external log rotation.

**memcap-set <config> <memcap>**
Update memcap value of a specified item.

**memcap-show <config>**
Show memcap value of a specified item.

**memcap-list**
List all memcap values available.

### 22.2.4 PCAP MODE COMMANDS

**pcap-file <file> <dir> [tenant] [continuous] [delete-when-done]**
Add pcap files to Suricata for sequential processing. The generated log/alert files will be put into the directory specified as second argument. Make sure to provide absolute path to the files and directory. It is acceptable to add multiple files without waiting the result.

**pcap-file-continuous <file> <dir> [tenant] [delete-when-done]**
Add pcap files to Suricata for sequential processing. Directory will be monitored for new files being added until there is a use of **pcap-interrupt** or directory is moved or deleted.

**pcap-file-number**
Number of pcap files waiting to get processed.

**pcap-file-list**
List of queued pcap files.

**pcap-last-processed**
Processed time of last file in milliseconds since epoch.

**pcap-interrupt**
Terminate the current state by interrupting directory processing.

**pcap-current**
Currently processed file.

### 22.2.5 BUGS

Please visit Suricata's support page for information about submitting bugs or feature requests.

### 22.2.6 NOTES

- Suricata Home Page

  https://suricata-ids.org/

- Suricata Support Page

  https://suricata-ids.org/support/

## 22.3 Suricata Control

### 22.3.1 SYNOPSIS

**suricatactl** [-h] <command> [<args>]

### 22.3.2 DESCRIPTION

This tool helps control Suricata's features.

### 22.3.3 OPTIONS

**-h**

Get help about the available commands.

### 22.3.4 COMMANDS

*suricatactl-filestore(1)*

### 22.3.5 BUGS

Please visit Suricata's support page for information about submitting bugs or feature requests.

### 22.3.6 NOTES

- Suricata Home Page

  https://suricata-ids.org/

- Suricata Support Page

  https://suricata-ids.org/support/

## 22.4 Suricata Control Filestore

### 22.4.1 SYNOPSIS

**suricatactl filestore** [-h] <command> [<args>]

### 22.4.2 DESCRIPTION

This command lets you perform certain operations on Suricata filestore.

### 22.4.3 OPTIONS

**-h**

Get help about the available commands.

### 22.4.4 COMMANDS

**prune [-h|–help] [-n|–dry-run] [-v|verbose] [-q|–quiet] -d <DIRECTORY> –age <AGE>**

Prune files older than a given age.

-d <DIRECTORY> | –directory <DIRECTORY> is a required argument which tells that user must provide the suricata filestore directory on which all the specified operations are to be performed.

–age <AGE> is a required argument asking the age of the files. Files older than the age mentioned with this option shall be pruned.

-h | –help is an optional argument with which you can ask for help about the command usage.

-n | –dry-run is an optional argument which makes the utility print only what would happen

-v | –verbose is an optional argument to increase the verbosity of command.

-q | –quiet is an optional argument that helps log errors and warnings only and keep silent about everything else.

### 22.4.5 BUGS

Please visit Suricata's support page for information about submitting bugs or feature requests.

### 22.4.6 NOTES

- Suricata Home Page

  https://suricata-ids.org/

- Suricata Support Page

  https://suricata-ids.org/support/

CHAPTER 23

# Acknowledgements

Thank you to the following for their Wiki and documentation contributions that have made this user guide possible:

- Andreas Herz
- Andreas Moe
- Anne-Fleur Koolstra
- Christophe Vandeplas
- Darren Spruell
- David Cannings
- David Diallo
- David Wharton
- Eric Leblond
- god lol
- Haris Haq
- Ignacio Sanchez
- Jason Ish
- Jason Taylor
- Josh Smith
- Ken Steele
- Les Syv
- Mark Solaris
- Martin Holste
- Mats Klepsland
- Matt Jonkman

- Michael Bentley
- Michael Hrishenko
- Nathan Jimerson
- Nicolas Merle
- Peter Manev
- Philipp Buehler
- Ralph Broenink
- Rob MacGregor
- Russel Fulton
- Victor Julien
- Vincent Fang
- Zach Rasmor

CHAPTER 24

# Licenses

## 24.1 GNU General Public License

*Version 2, June 1991 Copyright © 1989, 1991 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA*

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

### 24.1.1 Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: **(1)** copyright the software, and **(2)** offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that

what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

### 24.1.2 TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

**0.** This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

**1.** You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

**2.** You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

- **a)** You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
- **b)** You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
- **c)** If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

**3.** You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

- **a)** Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- **b)** Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- **c)** Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

**4.** You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

**5.** You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

**6.** Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

**7.** If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

**8.** If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

**9.** The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

**10.** If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

### 24.1.3 NO WARRANTY

**11.** BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

**12.** IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

*END OF TERMS AND CONDITIONS*

### 24.1.4 How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> <one line to give the program's name and a brief idea of what it does.> Copyright (C) <year> <name of author>
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> Gnomovision version 69, Copyright (C) year name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items–whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the program 'Gnomovision' (which makes passes at compilers) written by James Hacker.
>
> <signature of Ty Coon>, 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License.

## 24.2 Creative Commons Attribution-NonCommercial 4.0 International Public License

By exercising the Licensed Rights (defined below), You accept and agree to be bound by the terms and conditions of this Creative Commons Attribution-NonCommercial 4.0 International Public License ("Public License"). To the extent this Public License may be interpreted as a contract, You are granted the Licensed Rights in consideration of Your acceptance of these terms and conditions, and the Licensor grants You such rights in consideration of benefits the Licensor receives from making the Licensed Material available under these terms and conditions.

### 24.2.1 Section 1 – Definitions.

a. **Adapted Material** means material subject to Copyright and Similar Rights that is derived from or based upon the Licensed Material and in which the Licensed Material is translated, altered, arranged, transformed, or otherwise modified in a manner requiring permission under the Copyright and Similar Rights held by the Licensor. For purposes of this Public License, where the Licensed Material is a musical work, performance, or sound

recording, Adapted Material is always produced where the Licensed Material is synched in timed relation with a moving image.

b. **Adapter's License** means the license You apply to Your Copyright and Similar Rights in Your contributions to Adapted Material in accordance with the terms and conditions of this Public License.

c. **Copyright and Similar Rights** means copyright and/or similar rights closely related to copyright including, without limitation, performance, broadcast, sound recording, and Sui Generis Database Rights, without regard to how the rights are labeled or categorized. For purposes of this Public License, the rights specified in Section 2(b)(1)-(2) are not Copyright and Similar Rights.

d. **Effective Technological Measures** means those measures that, in the absence of proper authority, may not be circumvented under laws fulfilling obligations under Article 11 of the WIPO Copyright Treaty adopted on December 20, 1996, and/or similar international agreements.

e. **Exceptions and Limitations** means fair use, fair dealing, and/or any other exception or limitation to Copyright and Similar Rights that applies to Your use of the Licensed Material.

f. **Licensed Material** means the artistic or literary work, database, or other material to which the Licensor applied this Public License.

g. **Licensed Rights** means the rights granted to You subject to the terms and conditions of this Public License, which are limited to all Copyright and Similar Rights that apply to Your use of the Licensed Material and that the Licensor has authority to license.

h. **Licensor** means the individual(s) or entity(ies) granting rights under this Public License.

i. **NonCommercial** means not primarily intended for or directed towards commercial advantage or monetary compensation. For purposes of this Public License, the exchange of the Licensed Material for other material subject to Copyright and Similar Rights by digital file-sharing or similar means is NonCommercial provided there is no payment of monetary compensation in connection with the exchange.

j. **Share** means to provide material to the public by any means or process that requires permission under the Licensed Rights, such as reproduction, public display, public performance, distribution, dissemination, communication, or importation, and to make material available to the public including in ways that members of the public may access the material from a place and at a time individually chosen by them.

k. **Sui Generis Database Rights** means rights other than copyright resulting from Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases, as amended and/or succeeded, as well as other essentially equivalent rights anywhere in the world.

l. **You** means the individual or entity exercising the Licensed Rights under this Public License. Your has a corresponding meaning.

### 24.2.2 Section 2 – Scope.

a. **License grant.**

1. Subject to the terms and conditions of this Public License, the Licensor hereby grants You a worldwide, royalty-free, non-sublicensable, non-exclusive, irrevocable license to exercise the Licensed Rights in the Licensed Material to:

A. reproduce and Share the Licensed Material, in whole or in part, for NonCommercial purposes only; and

B. produce, reproduce, and Share Adapted Material for NonCommercial purposes only.

2. **Exceptions and Limitations.** For the avoidance of doubt, where Exceptions and Limitations apply to Your use, this Public License does not apply, and You do not need to comply with its terms and conditions.

3. **Term.** The term of this Public License is specified in Section 6(a).
4. **Media and formats; technical modifications allowed.** The Licensor authorizes You to exercise the Licensed Rights in all media and formats whether now known or hereafter created, and to make technical modifications necessary to do so. The Licensor waives and/or agrees not to assert any right or authority to forbid You from making technical modifications necessary to exercise the Licensed Rights, including technical modifications necessary to circumvent Effective Technological Measures. For purposes of this Public License, simply making modifications authorized by this Section 2(a)(4) never produces Adapted Material.
5. **Downstream recipients.**

A. **Offer from the Licensor – Licensed Material.** Every recipient of the Licensed Material automatically receives an offer from the Licensor to exercise the Licensed Rights under the terms and conditions of this Public License.

B. **No downstream restrictions.** You may not offer or impose any additional or different terms or conditions on, or apply any Effective Technological Measures to, the Licensed Material if doing so restricts exercise of the Licensed Rights by any recipient of the Licensed Material.

6. **No endorsement.** Nothing in this Public License constitutes or may be construed as permission to assert or imply that You are, or that Your use of the Licensed Material is, connected with, or sponsored, endorsed, or granted official status by, the Licensor or others designated to receive attribution as provided in Section 3(a)(1)(A)(i).

b. **Other rights.**

1. Moral rights, such as the right of integrity, are not licensed under this Public License, nor are publicity, privacy, and/or other similar personality rights; however, to the extent possible, the Licensor waives and/or agrees not to assert any such rights held by the Licensor to the limited extent necessary to allow You to exercise the Licensed Rights, but not otherwise.
2. Patent and trademark rights are not licensed under this Public License.
3. To the extent possible, the Licensor waives any right to collect royalties from You for the exercise of the Licensed Rights, whether directly or through a collecting society under any voluntary or waivable statutory or compulsory licensing scheme. In all other cases the Licensor expressly reserves any right to collect such royalties, including when the Licensed Material is used other than for NonCommercial purposes.

### 24.2.3 Section 3 – License Conditions.

Your exercise of the Licensed Rights is expressly made subject to the following conditions.

a. **Attribution.**

1. If You Share the Licensed Material (including in modified form), You must:

A. retain the following if it is supplied by the Licensor with the Licensed Material:

i. identification of the creator(s) of the Licensed Material and any others designated to receive attribution, in any reasonable manner requested by the Licensor (including by pseudonym if designated);

ii. a copyright notice;

iii. a notice that refers to this Public License;

iv. a notice that refers to the disclaimer of warranties;

v. a URI or hyperlink to the Licensed Material to the extent reasonably practicable;

B. indicate if You modified the Licensed Material and retain an indication of any previous modifications; and

C. indicate the Licensed Material is licensed under this Public License, and include the text of, or the URI or hyperlink to, this Public License.

2. You may satisfy the conditions in Section 3(a)(1) in any reasonable manner based on the medium, means, and context in which You Share the Licensed Material. For example, it may be reasonable to satisfy the conditions by providing a URI or hyperlink to a resource that includes the required information.

3. If requested by the Licensor, You must remove any of the information required by Section 3(a)(1)(A) to the extent reasonably practicable.

4. If You Share Adapted Material You produce, the Adapter's License You apply must not prevent recipients of the Adapted Material from complying with this Public License.

### 24.2.4 Section 4 – Sui Generis Database Rights.

Where the Licensed Rights include Sui Generis Database Rights that apply to Your use of the Licensed Material:

a. for the avoidance of doubt, Section 2(a)(1) grants You the right to extract, reuse, reproduce, and Share all or a substantial portion of the contents of the database for NonCommercial purposes only;

b. if You include all or a substantial portion of the database contents in a database in which You have Sui Generis Database Rights, then the database in which You have Sui Generis Database Rights (but not its individual contents) is Adapted Material; and

c. You must comply with the conditions in Section 3(a) if You Share all or a substantial portion of the contents of the database.

For the avoidance of doubt, this Section 4 supplements and does not replace Your obligations under this Public License where the Licensed Rights include other Copyright and Similar Rights.

### 24.2.5 Section 5 – Disclaimer of Warranties and Limitation of Liability.

a. **Unless otherwise separately undertaken by the Licensor, to the extent possible, the Licensor offers the Licensed Material as-is and as-available, and makes no representations or warranties of any kind concerning the Licensed Material, whether express, implied, statutory, or other. This includes, without limitation, warranties of title, merchantability, fitness for a particular purpose, non-infringement, absence of latent or other defects, accuracy, or the presence or absence of errors, whether or not known or discoverable. Where disclaimers of warranties are not allowed in full or in part, this disclaimer may not apply to You.**

b. **To the extent possible, in no event will the Licensor be liable to You on any legal theory (including, without limitation, negligence) or otherwise for any direct, special, indirect, incidental, consequential, punitive, exemplary, or other losses, costs, expenses, or damages arising out of this Public License or use of the Licensed Material, even if the Licensor has been advised of the possibility of such losses, costs, expenses, or damages. Where a limitation of liability is not allowed in full or in part, this limitation may not apply to You.**

c. The disclaimer of warranties and limitation of liability provided above shall be interpreted in a manner that, to the extent possible, most closely approximates an absolute disclaimer and waiver of all liability.

### 24.2.6 Section 6 – Term and Termination.

a. This Public License applies for the term of the Copyright and Similar Rights licensed here. However, if You fail to comply with this Public License, then Your rights under this Public License terminate automatically.

b. Where Your right to use the Licensed Material has terminated under Section 6(a), it reinstates:

   1. automatically as of the date the violation is cured, provided it is cured within 30 days of Your discovery of the violation; or

   2. upon express reinstatement by the Licensor.

   For the avoidance of doubt, this Section 6(b) does not affect any right the Licensor may have to seek remedies for Your violations of this Public License.

c. For the avoidance of doubt, the Licensor may also offer the Licensed Material under separate terms or conditions or stop distributing the Licensed Material at any time; however, doing so will not terminate this Public License.

d. Sections 1, 5, 6, 7, and 8 survive termination of this Public License.

### 24.2.7 Section 7 – Other Terms and Conditions.

a. The Licensor shall not be bound by any additional or different terms or conditions communicated by You unless expressly agreed.

b. Any arrangements, understandings, or agreements regarding the Licensed Material not stated herein are separate from and independent of the terms and conditions of this Public License.

### 24.2.8 Section 8 – Interpretation.

a. For the avoidance of doubt, this Public License does not, and shall not be interpreted to, reduce, limit, restrict, or impose conditions on any use of the Licensed Material that could lawfully be made without permission under this Public License.

b. To the extent possible, if any provision of this Public License is deemed unenforceable, it shall be automatically reformed to the minimum extent necessary to make it enforceable. If the provision cannot be reformed, it shall be severed from this Public License without affecting the enforceability of the remaining terms and conditions.

c. No term or condition of this Public License will be waived and no failure to comply consented to unless expressly agreed to by the Licensor.

d. Nothing in this Public License constitutes or may be interpreted as a limitation upon, or waiver of, any privileges and immunities that apply to the Licensor or You, including from the legal processes of any jurisdiction or authority.

**Creative Commons is not a party to its public licenses. Notwithstanding, Creative Commons may elect to apply one of its public licenses to material it publishes and in those instances will be considered the "Licensor." Except for the limited purpose of indicating that material is shared under a Creative Commons public license or as otherwise permitted by the Creative Commons policies published at creativecommons.org/policies, Creative Commons does not authorize the use of the trademark "Creative Commons" or any other trademark or logo of Creative Commons without its prior written consent including, without limitation, in connection with any unauthorized modifications to any of its public licenses or any other arrangements, understandings, or agreements concerning use of licensed material. For the avoidance of doubt, this paragraph does not form part of the public licenses.**

**Creative Commons may be contacted at creativecommons.org.**

## 24.3 Suricata Source Code

The Suricata source code is licensed under version 2 of the *GNU General Public License*.

## 24.4 Suricata Documentation

The Suricata documentation (this documentation) is licensed under the *Creative Commons Attribution-NonCommercial 4.0 International Public License*.

# Index

## Symbols


-af-packet[=<device>]
    command line option, 14, 330
-build-info
    command line option, 15, 331
-dag <device>
    command line option, 15, 332
-disable-detection
    command line option, 15, 331
-disable-gccmarch-native
    command line option, 7
-dump-config
    command line option, 15, 331
-enable-geoip
    command line option, 8
-enable-lua
    command line option, 8
-engine-analysis
    command line option, 15, 331
-erf-in=<file>
    command line option, 15, 332
-fatal-unittests
    command line option, 16, 332
-group=<group>
    command line option, 14, 331
-init-errors-fatal
    command line option, 14, 331
-list-app-layer-protos
    command line option, 15, 331
-list-keywords=[all|csv|<kword>]
    command line option, 15, 331
-list-runmodes
    command line option, 15, 331
-list-unittests
    command line option, 16, 332
-localstatedir=/var
    command line option, 7
-napatech
    command line option, 15, 332
-netmap[=<device>]
    command line option, 15, 331
-pcap-buffer-size=<size>
    command line option, 15, 331
-pcap-file-continuous
    command line option, 13, 330
-pcap-file-delete
    command line option, 13, 330
-pcap[=<device>]
    command line option, 14, 330
-pfring-cluster-id <id>
    command line option, 15, 331
-pfring-cluster-type <type>
    command line option, 15, 332
-pfring[=<device>]
    command line option, 15, 331
-pidfile <file>
    command line option, 14, 331
-prefix=/usr/
    command line option, 7
-runmode <runmode>
    command line option, 14, 330
-set <key>=<value>
    command line option, 15, 331
-simulate-ips
    command line option, 16, 332
-sysconfdir=/etc
    command line option, 7
-unittests-coverage
    command line option, 16, 332
-unix-socket=<file>
    command line option, 15, 331
-user=<user>
    command line option, 14, 331
-D
    command line option, 14, 330
-F <bpf filter file>
    command line option, 14, 330
-S <filename.rules>
    command line option, 14, 330

-T
    command line option, 13, 329
-U, -unittest-filter=REGEX
    command line option, 16, 332
-V
    command line option, 13, 329
-c <path>
    command line option, 13, 329
-d <divert-port>
    command line option, 15, 332
-h
    command line option, 13, 329, 335, 336
-i <interface>
    command line option, 14, 330
-k [all|none]
    command line option, 14, 331
-l <directory>
    command line option, 14, 330
-q <queue id>
    command line option, 14, 330
-r <path>
    command line option, 13, 330
-s <filename.rules>
    command line option, 14, 330
-u
    command line option, 16, 332
-v
    command line option, 13, 329

## C

command line option
    -af-packet[=<device>], 14, 330
    -build-info, 15, 331
    -dag <device>, 15, 332
    -disable-detection, 15, 331
    -disable-gccmarch-native, 7
    -dump-config, 15, 331
    -enable-geoip, 8
    -enable-lua, 8
    -engine-analysis, 15, 331
    -erf-in=<file>, 15, 332
    -fatal-unittests, 16, 332
    -group=<group>, 14, 331
    -init-errors-fatal, 14, 331
    -list-app-layer-protos, 15, 331
    -list-keywords=[all|csv|<kword>], 15, 331
    -list-runmodes, 15, 331
    -list-unittests, 16, 332
    -localstatedir=/var, 7
    -napatech, 15, 332
    -netmap[=<device>], 15, 331
    -pcap-buffer-size=<size>, 15, 331
    -pcap-file-continuous, 13, 330
    -pcap-file-delete, 13, 330
    -pcap[=<device>], 14, 330
    -pfring-cluster-id <id>, 15, 331
    -pfring-cluster-type <type>, 15, 332
    -pfring[=<device>], 15, 331
    -pidfile <file>, 14, 331
    -prefix=/usr/, 7
    -runmode <runmode>, 14, 330
    -set <key>=<value>, 15, 331
    -simulate-ips, 16, 332
    -sysconfdir=/etc, 7
    -unittests-coverage, 16, 332
    -unix-socket=<file>, 15, 331
    -user=<user>, 14, 331
    -D, 14, 330
    -F <bpf filter file>, 14, 330
    -S <filename.rules>, 14, 330
    -T, 13, 329
    -U, -unittest-filter=REGEX, 16, 332
    -V, 13, 329
    -c <path>, 13, 329
    -d <divert-port>, 15, 332
    -h, 13, 329, 335, 336
    -i <interface>, 14, 330
    -k [all|none], 14, 331
    -l <directory>, 14, 330
    -q <queue id>, 14, 330
    -r <path>, 13, 330
    -s <filename.rules>, 14, 330
    -u, 16, 332
    -v, 13, 329