Suricata User Guide

Release 3.2.1

OISF

February 15, 2017

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.

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.

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.

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*.

Source

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

Basic steps:

```
tar xzvf suricata-3.2beta1.tar.gz
cd suricata-3.2beta1
./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`

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-geopip
 Enables GeoIP support for detection.

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

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
```

Extra for iptables/nftables IPS integration:

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

Binary packages

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 sudo apt-get install suricata
```

Debian

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
```

Fedora

```
dnf install suricata
```

RHEL/CentOS

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

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

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

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

The -v option enables more verbosity of Suricata's output. Supply multiple times for more verbosity.

-r <path>
Run in pcap offline mode reading files from pcap file.

-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.

--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 deamon (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.

--mpipe
> Enable packet capture using the TileGX mpipe interface.

--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.

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.

SURICATA RULES

Rules Introduction

Contents

- *Rules Introduction*
    - *Action*
    - *Protocol*
    - *Source and destination*
    - *Ports (source-and destination-port)*
    - *Direction*
    - *Rule options*

Signatures play a very important role in Suricata. In most occasions people are using existing rulesets. The most used are Emerging Threats, Emerging Threats Pro and Sourcefire's VRT. A way to install rules is described in Rule Management with Oinkmaster. This Suricata Rules document explains all about signatures; how to read-, adjust-and create them.

A rule/signature consists of the following:

> The action, header and rule-options.

Example of 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"; classtype:trojan-activity;
reference:url,doc.emergingthreats.net/2008124;
reference:url,www.emergingthreats.net/cgi-
bin/cvsweb.cgi/sigs/VIRUS/TROJAN_IRC_Bots;
sid:2008124; rev:2;)
```

■ Action

■ Header

■ Rule options

Action

For more information read 'Action Order' see *Action-order*.

Example:

```
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"; classtype:trojan-activity;
reference:url,doc.emergingthreats.net/2008124;
reference:url,www.emergingthreats.net/cgi-
bin/cvsweb.cgi/sigs/VIRUS/TROJAN_IRC_Bots;
sid:2008124; rev:2;)
```

In this example the red, bold-faced part is the action.

Protocol

This keyword in a signature tells Suricata which protocol it concerns. You can choose between four settings. tcp (for tcp-traffic), udp, icmp and ip. ip stands for 'all' or 'any'. Suricata adds a few protocols : http, ftp, tls (this includes ssl), smb and dns (from v2.0). These are the so-called application layer protocols or layer 7 protocols. If you have a signature with for instance a http-protocol, Suricata makes sure the signature can only match if it concerns http-traffic.

Example:

```
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"; classtype:trojan-activity;
reference:url,doc.emergingthreats.net/2008124;
reference:url,www.emergingthreats.net/cgi-
bin/cvsweb.cgi/sigs/VIRUS/TROJAN_IRC_Bots;
sid:2008124; rev:2;)
```

In this example the red, bold-faced part is the protocol.

Source and destination

In source you can assign IP-addresses; IPv4 and IPv6 combined as well as separated. You can also set variables such as HOME_NET. (For more information see *Rule-vars*. In the Yaml-file you can set IP-addresses for variables such as EXTERNAL_NET and HOME_NET. These settings will be used when you use these variables in a rule. In source and destination you can make use of signs like ! And [ ].

For example:

```
! 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)
[.....,[....]]
[.... ,![.....]]
```

Pay attention to the following:

If your settings in Yaml are:

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

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

Example of source and destination 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"; classtype:trojan-activity;
reference:url,doc.emergingthreats.net/2008124;
reference:url,www.emergingthreats.net/cgi-
bin/cvsweb.cgi/sigs/VIRUS/TROJAN_IRC_Bots;
sid:2008124; rev:2;)
```

The red, bold-faced part is the source.

```
drop tcp $HOME_NET any -> SEXTERNAL_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"; classtype:trojan-activity;
reference:url,doc.emergingthreats.net/2008124;
reference:url,www.emergingthreats.net/cgi-
bin/cvsweb.cgi/sigs/VIRUS/TROJAN_IRC_Bots;
sid:2008124; rev:2;)
```

The red, bold-faced part is the destination.

Ports (source-and destination-port)

Traffic comes in and goes out through ports. Different ports have different port-numbers. The HTTP-port for example is 80 while 443 is the port for HTTPS and MSN makes use of port 1863. Commonly the Source port will be set as 'any'. This will be influenced by the protocol. The source port is designated at random by the operating system. Sometimes it is possible to filter/screen on the source In setting ports you can make use of special signs as well, like described above at 'source'. Signs like:

```
!        exception/negation
:        range
[]       signs to make clear which parts belong together
,        separation
```

Example:

```
[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]]
[....[.....]]
```

Example of ports 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"; classtype:trojan-activity;
reference:url,doc.emergingthreats.net/2008124;
reference:url,www.emergingthreats.net/cgi-
bin/cvsweb.cgi/sigs/VIRUS/TROJAN_IRC_Bots;
sid:2008124; rev:2;)
```

```
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"; classtype:trojan-activity;
reference:url,doc.emergingthreats.net/2008124;
reference:url,www.emergingthreats.net/cgi-
bin/cvsweb.cgi/sigs/VIRUS/TROJAN_IRC_Bots;
sid:2008124; rev:2;)
```

In this example, the red, bold-faced part is the port.

Direction

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.

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

Example:

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

Example 1 tcp-session
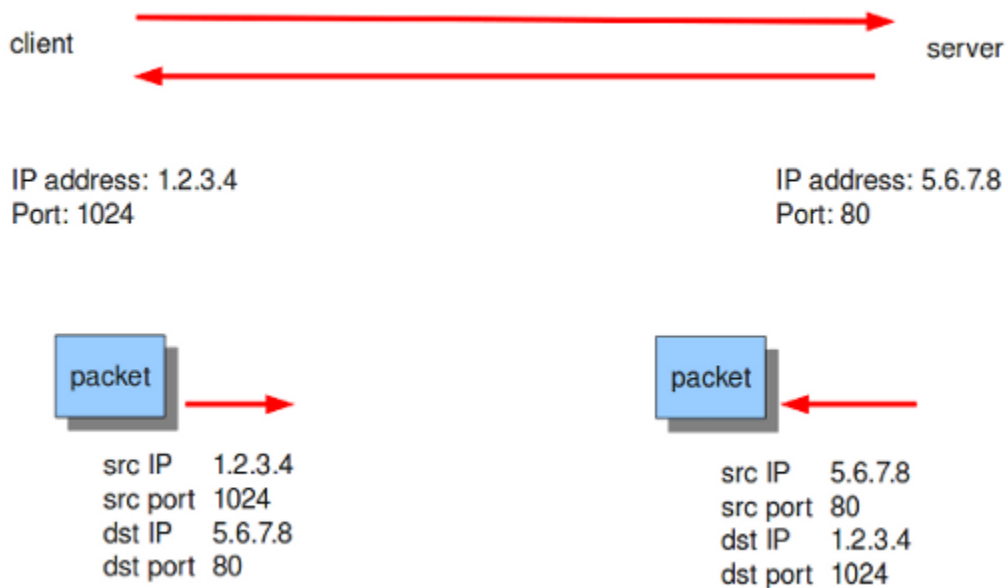
In this example there will only be a match if the signature has the same order/direction as the payload.

Example of direction 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"; classtype:trojan-activity;
reference:url,doc.emergingthreats.net/2008124;
reference:url,www.emergingthreats.net/cgi-
bin/cvsweb.cgi/sigs/VIRUS/TROJAN_IRC_Bots;
sid:2008124; rev:2;)
```

In this example the red, bold-faced part is the direction.

Rule options

Keywords have a set format:

```
name: settings;
```

Sometimes it is just the name of the setting followed by ; . Like nocase;

There are specific settings for:

- meta-information.

- headers

- payloads

- flows

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\;";
```

For more information about these settings, you can click on the following headlines:

- Meta-settings

- Payload Keywords

- HTTP Keywords

- DNS Keywords

- Flow Keywords

- IP Reputation Rules

Meta-settings

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

msg (message)

The keyword msg gives more information about the signature and the possible alert. The first part shows the class of the signature. It is a convention that part is written in uppercase characters.

The format of msg is:

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

Examples:

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

It is a convention that msg is always the first keyword of a signature.

Another example of msg in a signature:

In this example the red, bold-faced part is the msg.

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

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"; classtype:trojan-activity;
reference:url,doc.emergingthreats.net/2008124;
reference:url,www.emergingthreats.net/cgi-
bin/cvsweb.cgi/sigs/VIRUS/TROJAN_IRC_Bots;
sid:2008124; rev:2;)
```

In this example the red, bold-faced part is the sid.

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;
```

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

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"; classtype:trojan-activity;
reference:url,doc.emergingthreats.net/2008124;
reference:url,www.emergingthreats.net/cgi-
bin/cvsweb.cgi/sigs/VIRUS/TROJAN_IRC_Bots;
sid:2008124; rev:2;)
```

In this example the red, bold-faced part is the rev.

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 a signature:

```
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
```

This is an example from the fast.log. In the part [1:2008124:2], 1 is the gid (2008124 is the the sid and 2 the rev).

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.

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

Example classtype:

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

Signature	Classification.config	Alert
web-attack	web-attack, Web Application Attack, priority:1	Web Application Attack
not-suspicious	not-suspicious, Not Suspicious Traffic, priority:3	Not Suspicious Traffic

In this example you see how classtype appears in signatures, the classification.config and the alert.

Another example of classtype 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"; **classtype:trojan-activity;**
reference:url,doc.emergingthreats.net/2008124;
reference:url,www.emergingthreats.net/cgi-
bin/cvsweb.cgi/sigs/VIRUS/TROJAN_IRC_Bots;
sid:2008124; rev:2;)

In this example the red, bold-faced part is the classtype.

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: url, www.info.nl
```

In this example url is the type of reference. After that comes the actual reference (notice here you can not use http before the url).

There are different types of references:

type:

```
system              URL Prefix
bugtraq             http://www.securityfocus.com/bid
cve                 http://cve.mitre.org/cgi-bin/cvename.cgi?name=
nessus              http://cgi.nessus.org/plugins/dump.php3?id=
arachnids           (No longer available but you might still encounter this in signatures.)
                    http://www.whitehats.com/info/IDS
```

```
mcafee              http://vil.nai.com/vil/dispVirus.asp?virus_k=
url                 http://
```

For example bugtraq will be replaced by the full url:

```
reference: bugtraq, 123; http://www.securityfocus.com/bid
```

Example of reference 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"; classtype:trojan-activity;
reference:url,doc.emergingthreats.net/2008124;
reference:url,www.emergingthreats.net/cgi-
bin/cvsweb.cgi/sigs/VIRUS/TROJAN_IRC_Bots;
sid:2008124; rev:2;)
```

In this example the red, bold-faced part is the action.

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 class type. This can be overruled with the keyword priority. The format of priority is:

```
priority:1;
```

Metadata

Suricata ignores the words behind meta data. Suricata supports this keyword because it is part of the signature language. The format is:

```
metadata:......;
```

Header Keywords

IP-keywords

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; classtype:misc-activity;
reference:url,support.microsoft.com/default.aspx?scid=kb#-#-
EN-US#-#-q138268; reference:url,www.isi.edu/in-
notes/rfc1122.txt; sid:1321; rev:8;)
```

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 ;
classtype:bad-unknown; reference:arachnids,422; sid:502;
rev:2;)
```

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; classtype:bad-unknown;
reference:bugtraq,2666; reference:cve,1999-0016;
reference:url,www.cert.org/advisories/CA-1997-28.html;
sid:527; rev:8;)
```

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
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; classtype:non-standard-protocol;
reference:bugtraq,8211; reference:cve,2003-0567;
sid:2189; rev:3;)
```

The named variante of that example would be:

```
ip_proto:PIM
```

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 uses 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 SCAN 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; classtype: misc-activity;
reference:url,www.f5.com/f5products/v9intro/index.html;
reference:url,doc.emergingthreats.net/2001609;
reference:url,www.emergingthreats.net/cgi-
bin/cvsweb.cgi/sigs/SCAN/SCAN_F5_BIG-IP_Probe;
sid:2001609; rev:12;)
```

Geoip

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

```
both: both directions have to match with the given geoip (geopip's)
any: one of the directions have to match with the given geoip ('s).
dest: if the destination matches with the given geoip.
src: the source matches with the given geoip.
```

Fragments

Fragbits

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;
classtype: bad-unknown;
reference:url,doc.emergingthreats.net/bin/view/Main/2
001022; reference:url,www.emergingthreats.net/cgi-
bin/cvsweb.cgi/sigs/EXPLOIT/EXPLOIT_Invalid_TCP
_Fragments; sid:2001022; rev:5;)

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;**
classtype: bad-unknown;
reference:url,doc.emergingthreats.net/bin/view/Main/2
001022; reference:url,www.emergingthreats.net/cgi-
bin/cvsweb.cgi/sigs/EXPLOIT/EXPLOIT_Invalid_TCP
_Fragments; sid:2001022; rev:5;)

TCP keywords

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:

Example of seq in a packet (Wireshark):

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;
classtype:attempted-recon;
reference:arachnids,4; sid:623; rev:6;)
```

Example of ack in a packet (Wireshark):
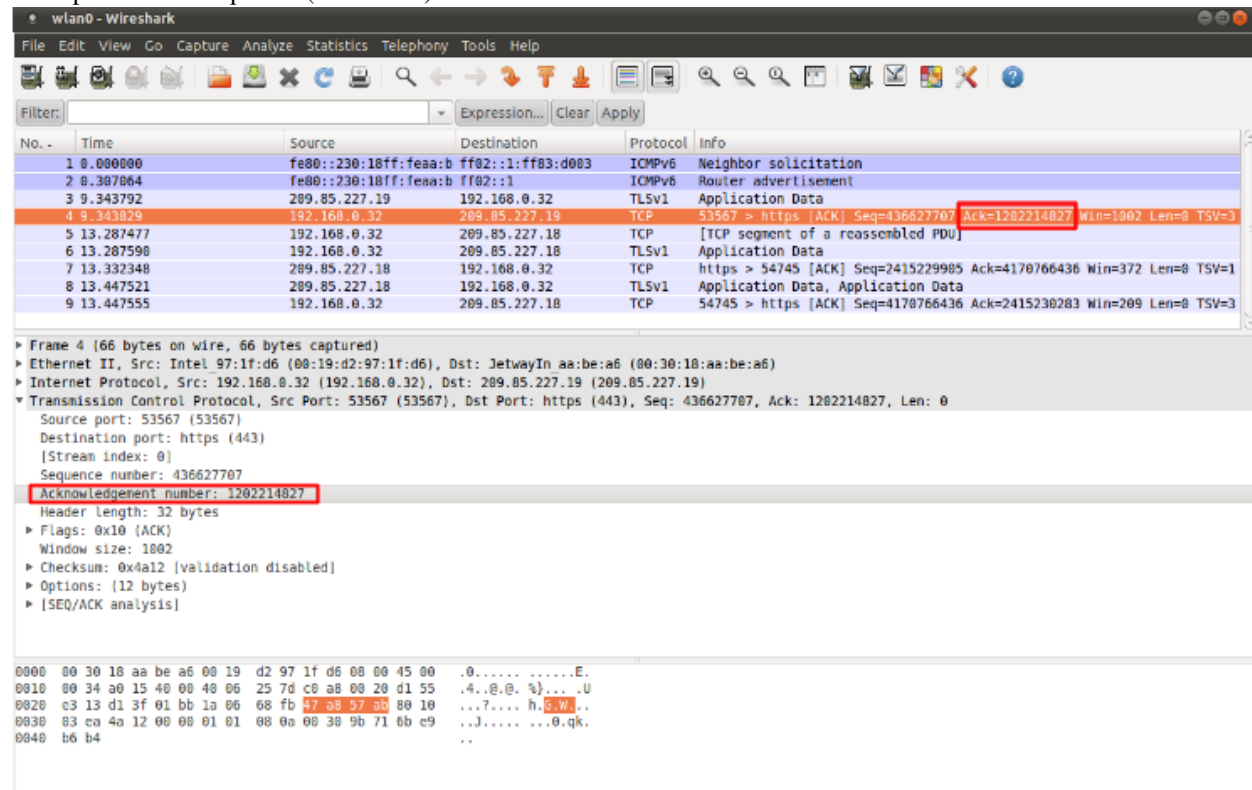
window

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;
classtype:trojan-activity; reference:mcafee,100406;
sid:2182; rev:8;)
```

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.

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:478; rev:3;)
```

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:449; rev:6;)
```

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:478; rev:3;)
```

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:478; rev:3;)
```

Message types and numbers:

Type	Name
0	Echo Reply
3	Destination Unreachable
4	Source Quench
5	Redirect
8	Echo
11	Time Exceeded
12	Parameter Problem
13	Timestamp
14	Timestamp Reply
15	Information Request
16	Information Reply
17	Address Mask Request
18	Adress Mask Reply

Meaning of type-numbers en codes combined:

Type	Code	Description
0	0	Echo Reply
3	0	Network Unreachable
3	1	Host Unreachable
3	2	Protocol Unreachable
3	3	Port Unreachable
3	4	Fragmentation needed but no fragment bit set
3	5	Source routing failed
3	6	Destination network unknown
3	7	Destination host unknown
3	8	Source host isolated (obsolete)
3	9	Destination network administratively prohibited
3	10	Destination host administratively prohibited
3	11	Network unreachable for TOS
3	12	Host unreachable for TOS
3	13	Communication administratively prohibited by filtering
3	14	Host precedence violation
3	15	Precedence cutoff in effect
4	0	Source quench
5	0	Redirect for network
5	1	Redirect for host
5	2	Redirect for TOS and network
5	3	Redirect for TOS and Host
8	0	Echo request
9	0	Router advertisement
10	0	Route solicitation
11	0	TTL equals 0 during transit
11	1	TTL equals 0 during reassembly
12	0	IP header bad (catchall error)
12	1	Required options missing
13	0	Timestamp request (obsolete)
14		Timestamp reply (obsolete)
15	0	Information request (obsolete)
16	0	Information reply (obsolete)
17	0	Address mask request
18	0	Address mask reply

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*

Payload Keywords

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:

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

Example of pcre:

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

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

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"; classtype:trojan-activity;
reference:url,doc.emergingthreats.net/2008124;
reference:url,www.emergingthreats.net/cgi-
bin/cvsweb.cgi/sigs/VIRUS/TROJAN_IRC_Bots;
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.
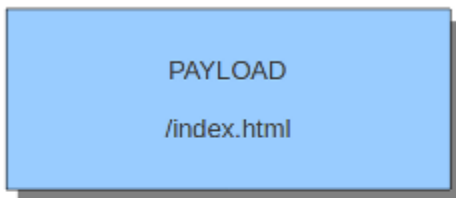
Note: The following characters must be escaped inside the content: `;` `\` `"`

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.

```
PAYLOAD

/index.html
```
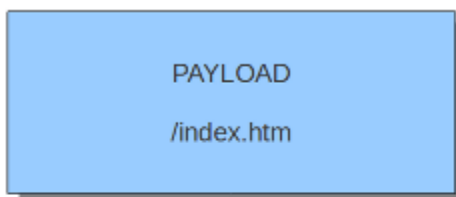
content:"/index."; http_uri; content:"htm"; http_uri; distance:0; ✔

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

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

```
PAYLOAD

/index.htm
```

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"; ✔

PAYLOAD

/index.htmo

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";   ✘

PAYLOAD

/index.abc.htm
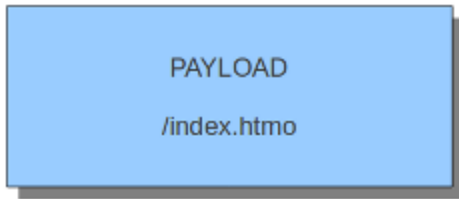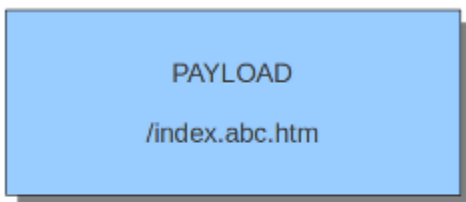
content:"/index."; http_uri; content:"htm"; http_uri; distance:0;   ✘

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.

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)	DE-TECT_SM_LIST_PMATCH	1
3	12	http_method	DE-TECT_SM_LIST_HMDMATCH	12
3	13	http_stat_code	DE-TECT_SM_LIST_HSCDMATCH	9
3	14	http_stat_msg	DE-TECT_SM_LIST_HSMDMATCH	8
2	1 (first)	http_client_body	DE-TECT_SM_LIST_HCBDMATCH	4
2	2	http_server_body	DE-TECT_SM_LIST_HSBDMATCH	5
2	3	http_header	DE-TECT_SM_LIST_HHDMATCH	6
2	4	http_raw_header	DE-TECT_SM_LIST_HRHDMATCH	7
2	5	http_uri	DE-TECT_SM_LIST_UMATCH	2
2	6	http_raw_uri	DE-TECT_SM_LIST_HRUDMATCH	3
2	7	http_host	DE-TECT_SM_LIST_HHHDMATCH	10
2	8	http_raw_host	DE-TECT_SM_LIST_HRHHDMATCH	11
2	9	http_cookie	DE-TECT_SM_LIST_HCDMATCH	13
2	10	http_user_agent	DE-TECT_SM_LIST_HUADMATCH	14
2	15 (last)	dns_query	DE-TECT_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
 *  \param patlen length of the patternn
 *
 *  \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.

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

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
|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.

Example:

```
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"; classtype:trojan-activity;
reference:url,doc.emergingthreats.net/2008124;
reference:url,www.emergingthreats.net/cgi-
bin/cvsweb.cgi/sigs/VIRUS/TROJAN_IRC_Bots;
sid:2008124; rev:2;)
```

In this example, the red, bold-faced part is the content.

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

PAYLOAD

a b C d e f g h l j

content:"abc";   ✗
content:"aBc";   ✗
content:"abC";   ✔

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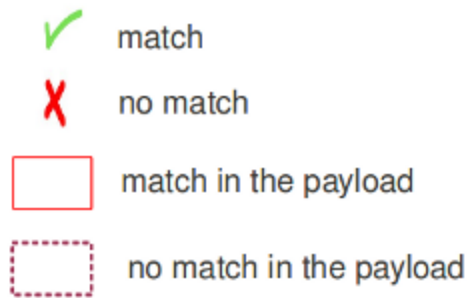
It is possible to use the ! for exceptions in contents as well.

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 the used version of Firefox is not 3.6.13.

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

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 behind the content you want to modify, like:

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

Example nocase:

It has no influence on other contents in the signature.

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:
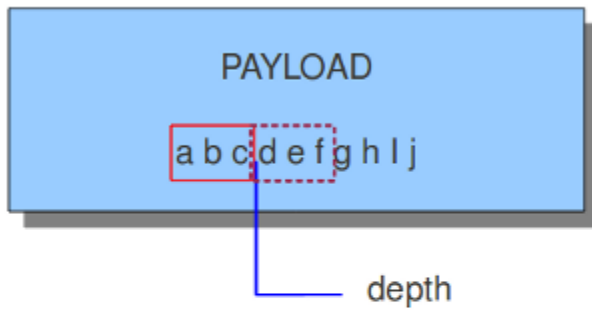
content:"def"; depth:3;   ✗

content:"abc"; depth:3;   ✔

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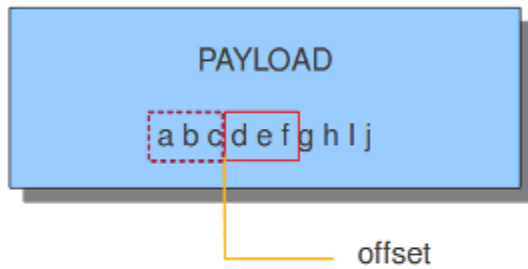
The keywords offset and depth can be combined and are often used together.

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.

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:

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

1   2   3

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

PAYLOAD

a b c d e f g h i j
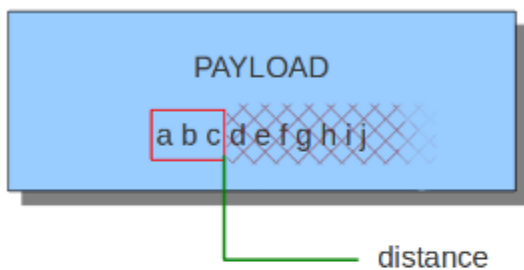
distance

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

checked area using 'distance'
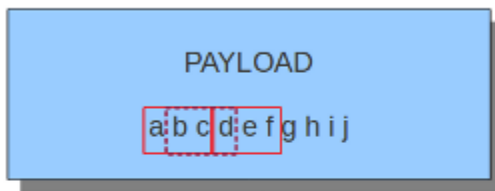
content:"abc"; content:"def"; distance:0; ✔

content:"abc"; content:"bcd"; distance:0; ✘
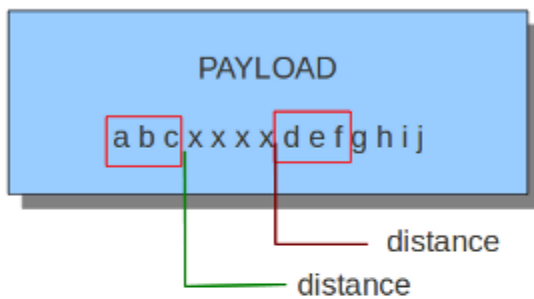
content:"abc"; content:"def"; distance:0; ✔

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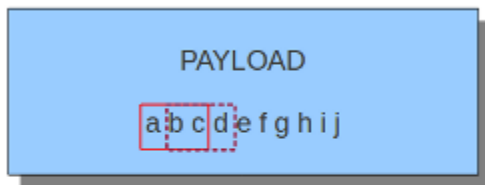
content:"abc"; content:"bcd"; distance:-2;   ✔

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:

content:"abc"; content:"klm"; within:3;

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

Example of matching with within:
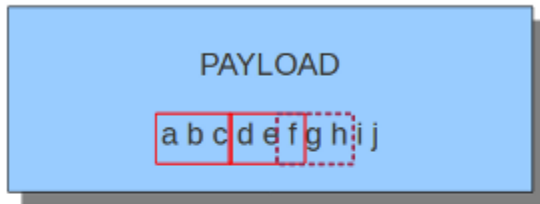
The second content has to fall/come 'within 3 ' from the first content.

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.
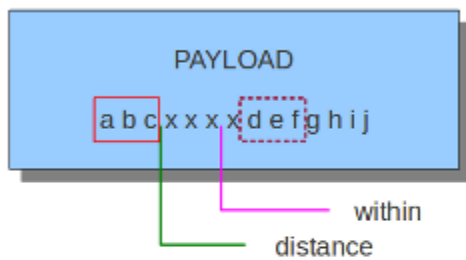
content:"abc"; content:"def"; distance:1; within:4; ✔
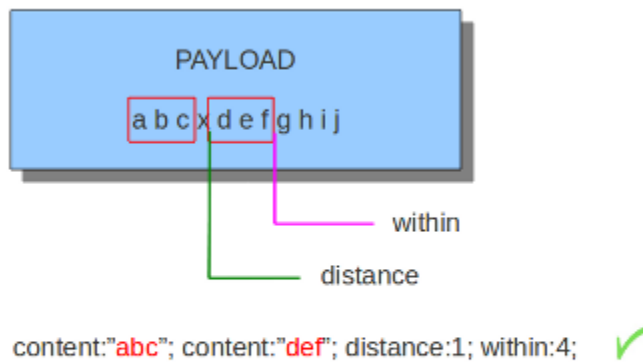
Isdataat

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.
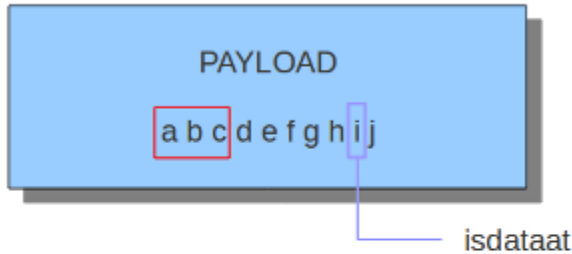
content:"abc"; isdataat:6, relative; ✔

content:"abc"; isdataat:8, relative; ✘

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; classtype: attempted-admin;
reference:bugtraq,23483; reference:cve,2007-1674;
sid:100000928; rev:1;)
```

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;)
```

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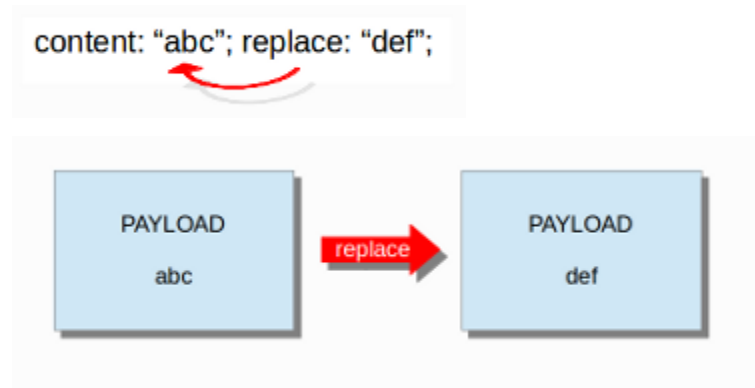
The replace modifier has to contain as many characters as the content it replaces. It can only be used with individual packets. It will not work for Normalized Buffers like HTTP uri or a content match in the reassembled stream.

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

pcre

For information about pcre check the pcre (Perl Compatible Regular Expressions) page.

fast_pattern

For information about fast_pattern check the Fast Pattern page.

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.

Types of modifiers

There are 2 types of modifiers. The older style 'content modifiers' look back in the rule.

Example:

```
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.

Example:

```
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.

The following request keywords are available:

Keyword	Sticky or Modifier	Direction
http_uri	Modifier	Request
http_raw_uri	Modifier	Request
http_method	Modifier	Request
http_request_line	Sticky Buffer	Request
http_client_body	Modifier	Request
http_header	Modifier	Both
http_raw_header	Modifier	Both
http_cookie	Modifier	Both
http_user_agent	Modifier	Request
http_host	Modifier	Request
http_raw_host	Modifier	Request

The following response keywords are available:

Keyword	Sticky or Modifier	Direction
http_stat_msg	Modifier	Response
http_stat_code	Modifier	Response
http_response_line	Sticky Buffer	Response
http_header	Modifier	Both
http_raw_header	Modifier	Both
http_cookie	Modifier	Both
http_server_body	Modifier	Response
file_data	Sticky Buffer	Response

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:

Response:

HTTP/1.1 302 Found	HTTP-version HTTP-response code, keyword: http_stat_code HTTP-response message, keyword: http_stat_msg
Location: http://www.google.nl/ Cache-Control: private Content-Type: text/html; charset=UTF-8 Set-Cookie: PREF=ID=efe36c63a3bfa6a4:FF =0:TM=1252314621:LM=129310 4406:GM=1:S=xeKyIaBhZkPrZEk N; expires=Sat, 22-Dec-2012 11:40:06 GMT; path=/; domain=.google.com Date: Thu, 23 Dec 2010 11:40:06 GMT Server: gws Content-Length: 218 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"> <TITLE>302 Moved</TITLE></HEAD><BODY > <H1>302 Moved</H1> The document has moved <A HREF="http://www.google.nl/">he re</A>. </BODY></HTML>	HTTP-response body, keywords: file_data, http_server_body

Request:

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.

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`.

Methods are: **GET**, **POST**, **PUT**, **HEAD**, **DELETE**, **TRACE**, **OPTIONS**, **CONNECT** and **PATCH**.

Example of a method in a HTTP request:

Example of the purpose of method:
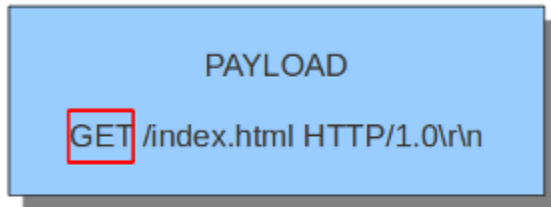
http_uri and http_raw_uri

With the `http_uri` and the `http_raw_uri` 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`.

To learn more about the difference between `http_uri` and `http_raw_uri`, please read the information about HTTP-uri normalization.

Example of the URI in a HTTP request:

Example of the purpose of `http_uri`:
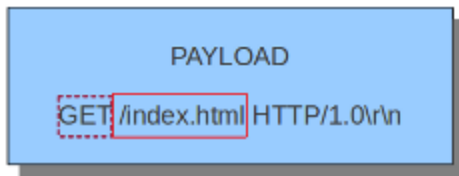
Example of the purpose of `http_raw_uri`:

#.. image:: http-keywords/raw_uri.png

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:
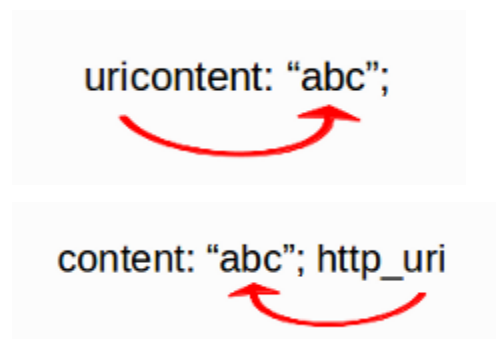
When authoring new rules, it is recommended that the `http_uri` content modifier be used rather than the deprecated `uricontent` keyword.

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:

PAYLOAD

GET /pictures/ HTTP/1.0

urilen:10;   ✔
urilen:<10;   ✘
urilen:5<>20;   ✔
urilen:20;   ✘
urilen:>4;   ✔

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).

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;)
```

http_header and http_raw_header

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: http://lists.openinfosecfoundation.org/pipermail/oisf-users/2011-October/000935.html. To avoid that, use the `http_raw_header` keyword.

Example of a header in a HTTP request:

Example of the purpose of `http_header`:
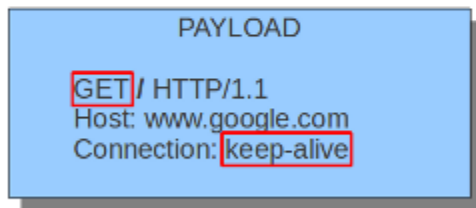
http_cookie

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`:


```
PAYLOAD

GET / HTTP/1.1
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=
0.3Cookie:PREF=ID=efe36c63a3bfa6a4:U
=aa0cf39996084d7e:TM
=1252314621:LM=1292956821:GM
=1:S=dYtecyNBioerA47b
```
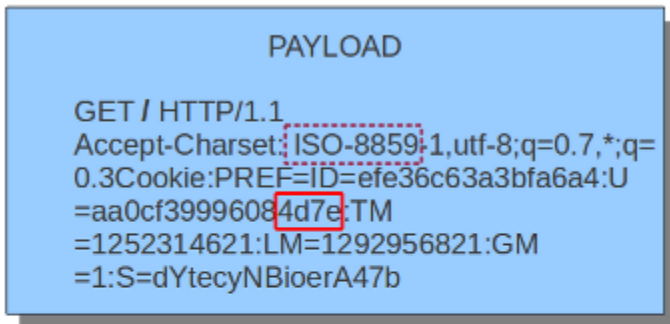
content:"4d7e"; http_uri;   ✔

content:"ISO-8859"; http_uri;   ✘

content:"4d7e"; http_cookie; depth: 13;   ✘

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.

An analysis into the performance of `http_user_agent` vs. `http_header` is found at: http://blog.inliniac.net/2012/07/09/suricata-http_user_agent-vs-http_header/

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`:

`content:"Mozilla/5.0"; http_user_agent;` ✔

`content:"google.com"; http_user_agent;` ✘

http_client_body

With the `http_client_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_client_body` in a HTTP request:

```
Host: nowhereasdfasdf.com
Connection: Keep-Alive
Cache-Control: no-cache

type=playerStart&position=tidal
```

Example of the purpose of `http_client_body`:

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

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:

Example of the purpose of `http_stat_code`:
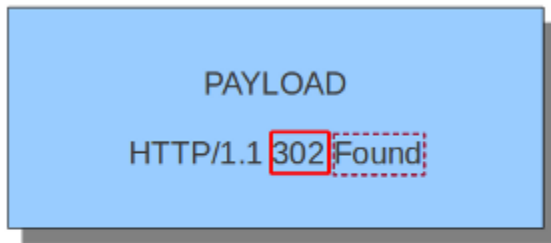
http_stat_msg

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:

Example of the purpose of `http_stat_msg`:

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;)
```

http_server_body

With the `http_server_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.

http_host and http_raw_host

With the `http_host` content modifier, it is possible to match specifically and only the normalized hostname. The `http_raw_host` 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 ist not allowed anymore. Keep in mind that you need to specify a lowercase pattern.
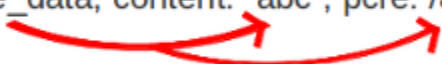
file_data

With `file_data`, the HTTP response body is inspected, just like with `http_server_body`. The `file_data` keyword works a bit differently from the normal content modifiers; when used in a rule, all content matches following it in the rule are affected (modified) by it.

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.

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

pcre

For information about the `pcre` keyword, check the pcre (Perl Compatible Regular Expressions) page.

fast_pattern

For information about the `fast_pattern` keyword, check the Fast Pattern page.

Flow Keywords

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:
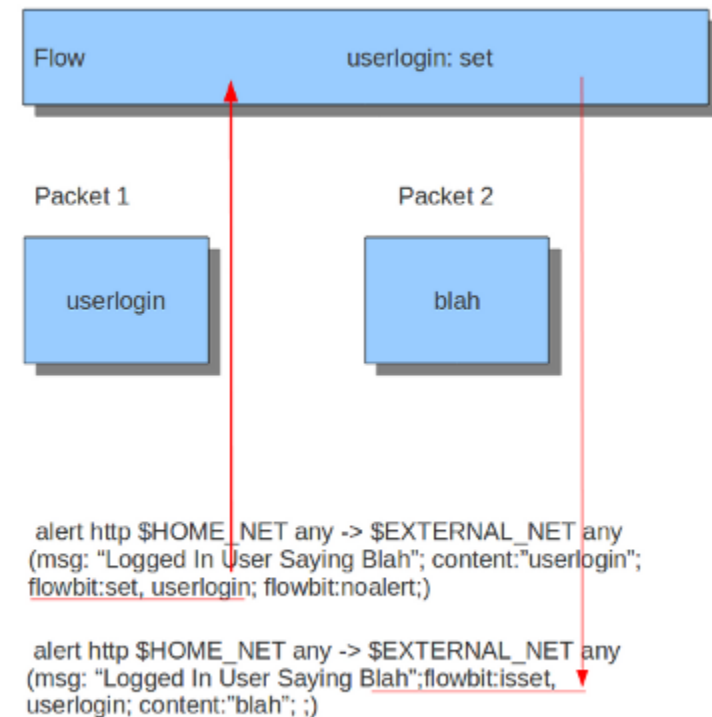
When you take a look at the first rule you will notice it would generate an alert if it would match, if it were not for the 'flowbits: noalert' at the end of that rule. The purpose of this rule is to check for a match on 'userlogin' and mark that in the flow. So, there is no need for generating an alert. 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.

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 reeassembled.

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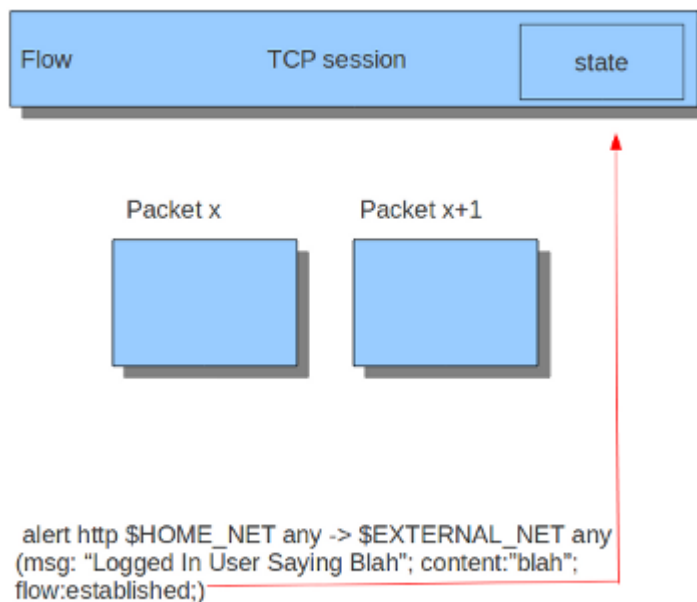
- For other protocols (for example UDP), the connection will be considered established after seeing traffic from both sides of the connection.
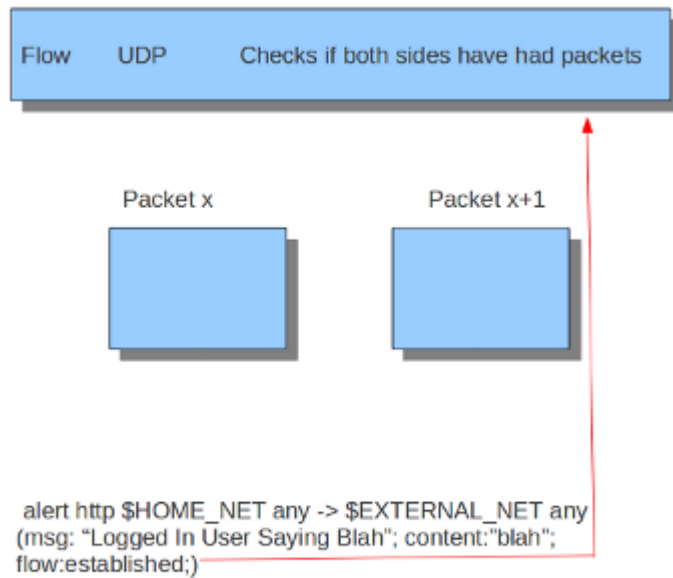
Flowint

For information, read the information on the Flowint page.

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:

Flowint

Flowint is a precursor to the Global Variables task we will be adding to the engine very soon, which will allow the capture, storage and comparison of data in a variable. It will be as the name implies Global. So you can compare data from packets in unrelated streams.

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: , ;

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

flowint: , , ;

flowint: , < +,-,=,>,<,>=,<=,==, != >, ;

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:"if the flowint counter is 3 create a new counter"; content:"Unauth
flowint:myvar, isset; flowint:myvar,==,3; flowint:cntpackets,notset; flowint:cntpackets, =, 0;)
```

```
alert tcp any any -> any any (msg:"and count the rest of the packets received without generating ale
flowint:cntpackets,isset; flowint:cntpackets, +, 1; noalert;)
```

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

Xbits

Set, unset, toggle and check for bits stored per host or ip_pair.

Syntax:

```
xbits:noalert;
xbits:<set|unset|isset|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>];
```

YAML settings

Bits that are stored per host are stored in the Host table.

Bits that are stored per IP pair are stored in the IPPair table.

Threading

Due to subtle timing issues between threads the order of sets and checks can be slightly unpredictible.

Example: create 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;)
```

File Keywords

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

filename

Matches on the file name.

Syntax:

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

Example:

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

fileext

Matches on the extension of a file name.

Syntax:

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

Example:

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

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.

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.

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: http://blog.inliniac.net/2012/06/09/suricata-md5-blacklisting/

filesize

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

Syntax:

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

Examples:

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

Note: For files that are not completely tracked because of packet loss or stream.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.

Rule Thresholding

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*

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
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 - Signif
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;                          \
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.*

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-ora
    flow:established,to_server; content:"GET"; http_method; content:"WebResource.axd"; http_uri; noc
    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/
    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

DNS Keywords

(from v2.0)
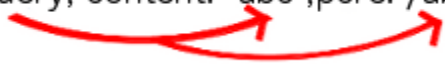
There are some more content modifiers (If you are unfamiliar with content modifiers, please visit the page Payload Keywords These ones make sure the signature checks a specific part of the network-traffic.

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.

SSL/TLS Keywords

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

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_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_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_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;)
```

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;)
```

tls_cert_expired

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

Usage:

```
tls_cert_expired;
```

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;
```

tls.version

Match on negotiated TLS/SSL version.

Example values: "1.0", "1.1", "1.2"

Support added in Suricata version 1.3.

tls.subject

Match TLS/SSL certificate Subject field.

example:

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

Support added in Suricata version 1.3.

Case sensitve, can't use 'nocase'.

Legacy keyword. `tls_cert_subject` is the replacement.

tls.issuerdn

match TLS/SSL certificate IssuerDN field

example:

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

Support added in Suricata version 1.3.

Case sensitve, can't use 'nocase'.

Legacy keyword. `tls_cert_issuer` is the replacement.

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"
```

Support added in Suricata version 1.4.

Case sensitive, can't use 'nocase'.

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

tls.store

store TLS/SSL certificate on disk

Support added in Suricata version 1.4.

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

Modbus Keyword

The modbus keyword can be used for matching on various properties of Modbus requests.

There are two ways of using this keyword:

- matching on functions properties with the setting "function";
- matching on directly on data access with the setting "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
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
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 | write> <discretes | coils | input | holding>
modbus: access <read | write> <discretes | coils | input | holding>, address <value>
modbus: access <read | write> <discretes | coils | input | 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 I
modbus: access write holding, address 500, value >200  # Write value greather than 200 at address 500
```

(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)

(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/

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.

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

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;
```

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.

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|;
```

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 seperated 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

Generic App Layer Keywords

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.

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*.

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.

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)
- 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.

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
```

Normalized Buffers

HTTP-uri normalization

The uri has two appearances in Suricata: the raw_uri 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 raw_uri. The raw_uri and the normalized uri are separate buffers. So, the raw_uri inspects the raw_uri buffer and can not inspect the normalized buffer.

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'.

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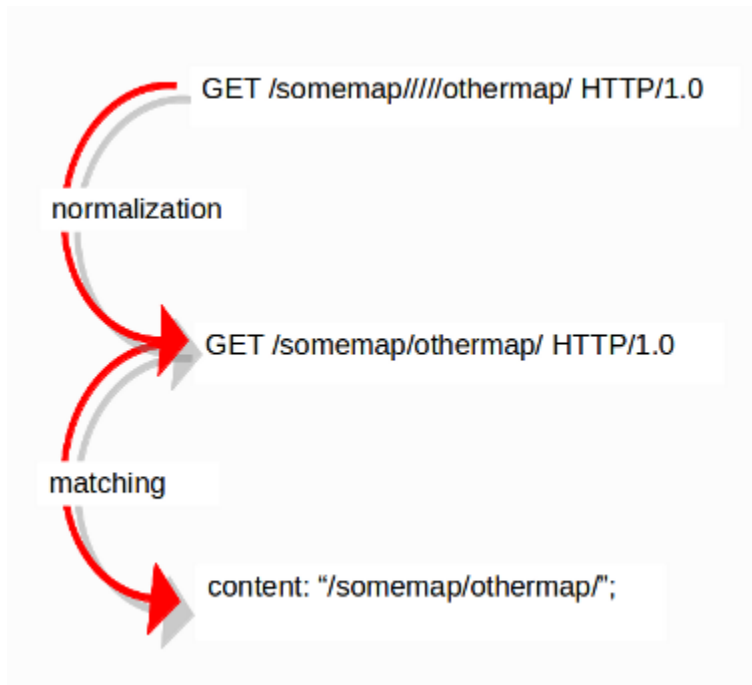
Because the data is being normalized, it is not what it used to be; it is an interpretation. Normalized buffers are: all HTTP-keywords, reassembled streams, TLS-, SSL-, SSH-, FTP- and dcerpc-buffers.

Snort Compatibility

Contents

Keyword: content

Versions affected: All versions prior to 3.0.

Prior to Suricata 3.0, the argument provided to the content keyword cannot be longer than 255 characters like it can in Snort.

Suricata 3.0 and newer can accept content arguments longer than 255 characters.

See:

- https://redmine.openinfosecfoundation.org/issues/1281

- https://github.com/inliniac/suricata/pull/1475

Keyword: urilen

Versions affected: all

In Snort the urilen range is inclusive, in Suricata it is not.

Example:

```
urilen:5<>10
```

In Snort the above will match URIs that are greater than and equal to 5 and less than and equal to 10. *Note that this is not what is documented in the Snort manual.*

In Suricata the above will match URIs that are greater than 5 and less than 10, so it will only mathch URIs that are 6, 7, 8, and 9 bytes long.

See:

- https://redmine.openinfosecfoundation.org/issues/1416

Keyword: isdataat

Versions affected: all

isdataat is off by one from Snort. In Snort the offset starts at 0 where Suricata starts at 1.

Keyword: flowbits

Versions affected: all prior to 2.0.9

Versions of Suricata prior to 2.0.9 treated leading and trailing whitespace in flowbit names as part of the flowbit name where Snort does not.

This was fixed in Suricata 2.0.9.

See:

- https://redmine.openinfosecfoundation.org/issues/1481

Keyword: flow:not_established

Versions affected: all

The not_established argument to the flow keyword is not supported.

RULE MANAGEMENT

Rule Management with Oinkmaster

It is possible to download and install rules manually, but there is a much easier and quicker way to do so. There are special programs which you can use for downloading and installing rules. There is for example Pulled Pork and Oinkmaster. In this documentation the use of Oinkmaster will be described.

To install Oinkmaster, enter:

```
sudo apt-get install oinkmaster
```

There are several rulesets. There is for example Emerging Threats (ET) Emerging Threats Pro and VRT. In this example we are using Emerging Threats.
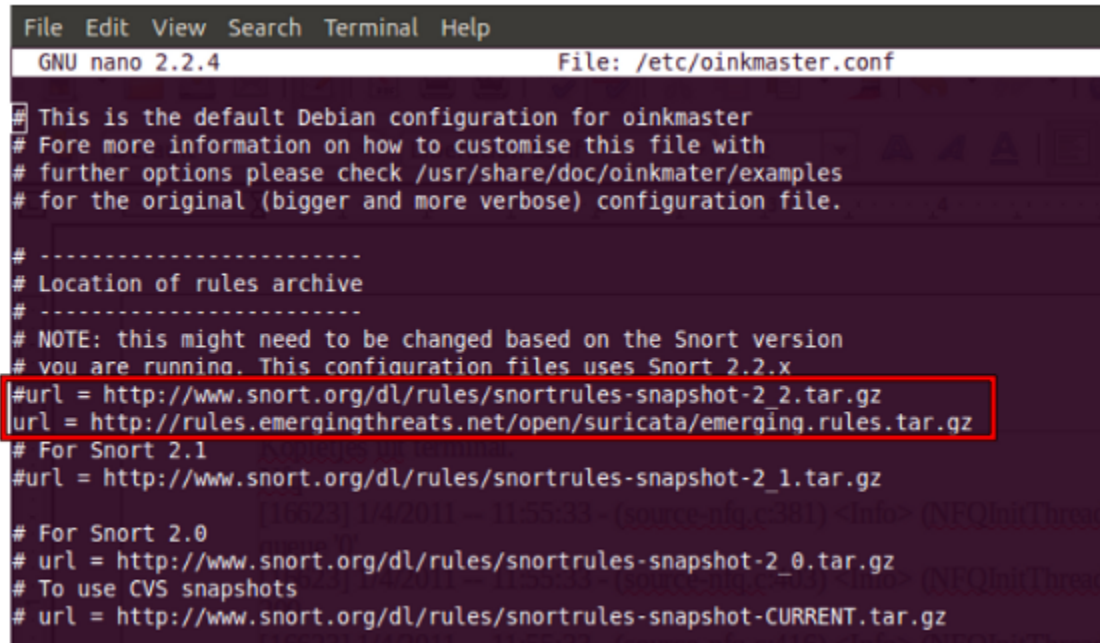
Oinkmaster has to know where the rules an be found. These rules can be found at:

```
https://rules.emergingthreats.net/open/suricata-3.2/emerging.rules.tar.gz
```

open oinkmaster.conf to add this link by entering:

```
sudo nano /etc/oinkmaster.conf
```

Place a # in front of the url that is already there and add the new url like this:

```
File  Edit  View  Search  Terminal  Help
  GNU nano 2.2.4                        File: /etc/oinkmaster.conf

# This is the default Debian configuration for oinkmaster
# Fore more information on how to customise this file with
# further options please check /usr/share/doc/oinkmater/examples
# for the original (bigger and more verbose) configuration file.

# -----------------------
# Location of rules archive
# -----------------------
# NOTE: this might need to be changed based on the Snort version
# you are running. This configuration files uses Snort 2.2.x
#url = http://www.snort.org/dl/rules/snortrules-snapshot-2_2.tar.gz
url = http://rules.emergingthreats.net/open/suricata/emerging.rules.tar.gz
# For Snort 2.1
#url = http://www.snort.org/dl/rules/snortrules-snapshot-2_1.tar.gz

# For Snort 2.0
# url = http://www.snort.org/dl/rules/snortrules-snapshot-2_0.tar.gz
# To use CVS snapshots
# url = http://www.snort.org/dl/rules/snortrules-snapshot-CURRENT.tar.gz
```

(Close oinkmaster.conf by pressing ctrl x, followed by y and enter.)

The next step is to create a directory for the new rules. Enter:

```
sudo mkdir /etc/suricata/rules
```

Next enter:

```
cd /etc
sudo oinkmaster -C /etc/oinkmaster.conf -o /etc/suricata/rules
```

In the new rules directory a classification.config and a reference.config can be found. The directories of both have to be added in the suricata.yaml file. Do so by entering:

```
sudo nano /etc/suricata/suricata.yaml
```

And add the new file locations instead of the file locations already present, like this:

To see if everything works as pleased, run Suricata:

```
suricata -c /etc/suricata/suricata.yaml -i wlan0 (or eth0)
```

You will notice there are several rule-files Suricata tries to load, but are not available. It is possible to disable those rule-sets in suricata.yaml by deleting them or by putting a # in front of them. To stop Suricata from running, press ctrl c.

Emerging Threats contains more rules than loaded in Suricata. To see which rules are available in your rules directory, enter:

```
ls /etc/suricata/rules/*.rules
```

Find those that are not yet present in suricata.yaml and add them in yaml if desired.

You can do so by entering :

```
sudo nano /etc/suricata/suricata.yaml
```

If you disable a rule in your rule file by putting a # in front of it, it will be enabled again the next time you run Oinkmaster. You can disable it through Oinkmaster instead, by entering the following:

```
cd /etc/suricata/rules
```

and find the sid of the rule(s) you want to disable.

Subsequently enter:

```
sudo nano /etc/oinkmaster.conf
```

and go all the way to the end of the file. Type there:

```
disablesid 2010495
```

Instead of 2010495, type the sid of the rule you would like to disable. It is also possible to disable multiple rules, by entering their sids separated by a comma.

If you run Oinkmaster again, you can see the amount of rules you have disabled. You can also enable rules that are disabled by default. Do so by entering:

```
ls /etc/suricata/rules
```

In this directory you can see several rule-sets Enter for example:

```
sudo nano /etc/suricata/rules/emerging-malware.rules
```

In this file you can see which rules are enabled en which are not. You can not enable them for the long-term just by simply removing the #. Because each time you will run Oinkmaster, the rule will be disabled again. Instead, look up the sid of the rule you want to enable. Place the sid in the correct place of oinkmaster.config:

```
sudo nano /etc/oinkmaster.conf
```

do so by typing:

```
enablesid: 2010495
```

Instead of 2010495, type the sid of the rule you would like to to enable. It is also possible to enable multiple rules, by entering their sids separated by a comma.

In oinkmaster.conf you can modify rules. For example, if you use Suricata as inline/IPS and you want to modify a rule that sends an alert when it matches and you would like the rule to drop the packet instead, you can do so by entering the following:

```
sudo nano oinkmaster.conf
```

At the part where you can modify rules, type:

```
modifysid 2010495 "alert" | "drop"
```

The sid 2010495 is an example. Type the sid of the rule you desire to change, instead.

Rerun Oinkmaster to notice the change.

Updating your rules

If you have already downloaded a ruleset (in the way described in this file), and you would like to update the rules, enter:

```
sudo oinkmaster -C /etc/oinkmaster.conf -o /etc/suricata/rules
```

It is recommended to update your rules frequently. Emerging Threats is modified daily, VRT is updated weekly or multiple times a week.

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 Introduction 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
```

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:

```
suricatasc -c reload-rules
```

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 favourite 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 lists helps a lot then.

For the Emerging Threats list this is: http://lists.emergingthreats.net/mailman/listinfo/emerging-sigs

For the VRT ruleset: https://lists.sourceforge.net/lists/listinfo/snort-sigs

In many cases, looking at just the alert and the packet that triggered it won't be enough to be conclusive. When running an IDS engine like Suricata, it's always recommended to combine it with full packet capturing. Using tools like Sguil or Snorby, the full TCP session or UDP flow can be inspected.

For example, if a rule fired that indicates your web application is attacked, looking at the full TCP session might reveal that the web application replied with 404 not found. This will usually mean the attack failed. Usually, not always.

Obviously there is a lot more to Incidence Response, but this should get you started.

PERFORMANCE

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.

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: auto, single, autofp. The heaviest task is the detection; a packet will be checked against thousands of signatures.

Example of the default runmode:
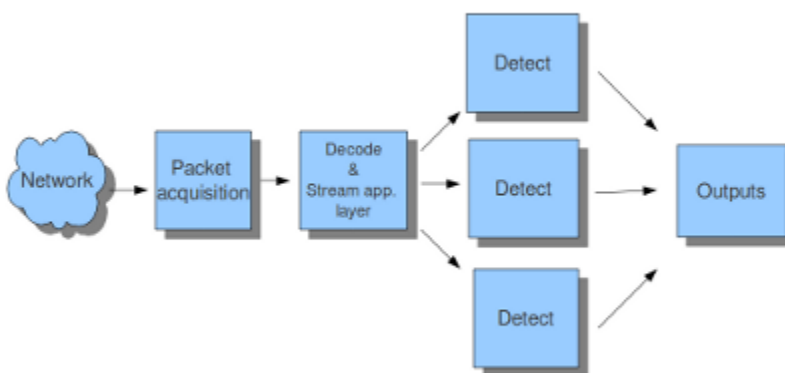
In the pfring mode, every flow follows its own fixed route in the runmode.
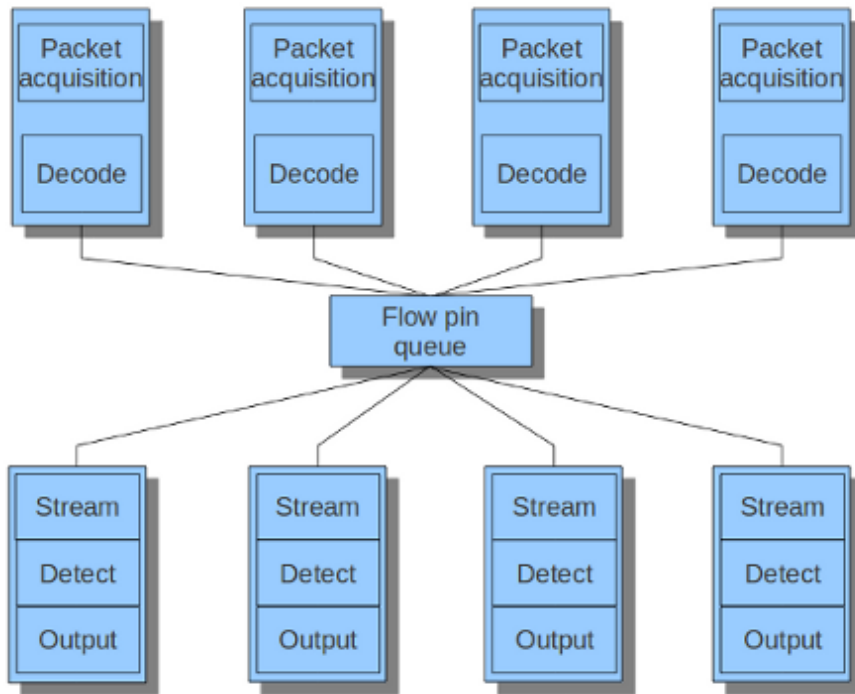
For more information about the command line options concerning the runmode, see Command Line Options.

Packet Capture

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.

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.

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.

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.

Tuning Considerations

Settings to check for optimal performance.

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: 1000 or higher. Max is ~65000.

mpm-algo: <ac|hs|ac-bs|ac-ks>

Controls the pattern matcher algorithm. AC is the default. On supported platforms, Hyperscan is the best option.

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. Higher is better but results in (much) higher memory usage.

The "custom" setting allows modification of the group sizes:

```
custom-values:
  toclient-groups: 50
  toserver-groups: 50
```

In general, increasing will improve performance, but will lead to higher memory usage.

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 and ac-bs use "single". All others "full". Setting this to "full" with AC requires a lot of memory: 32GB+ for a reasonable rule set.

Hyperscan

Introduction

"Hyperscan is a high-performance multiple regex matching library." https://01.org/hyperscan

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

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
```

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

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 http://downloads.sourceforge.net/project/boost/boost/1.60.0/boost_1_60_0.tar.gz
tar xvzf boost_1_60_0.tar.gz
cd boost_1_60_0
./bootstrap.sh --prefix=~/tmp/boost-1.60
./b2 install
```

Hyperscan

We'll install version 4.2.0.

```
git clone https://github.com/01org/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-1.60 ../
```

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
-- 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
```

High Performance Configuration

If you have enough RAM, consider the following options in suricata.yaml to off-load as much work from the CPU's as possible:

```
detect:
  profile: custom
  custom-values:
      toclient-groups: 200
      toserver-groups: 200
  sgh-mpm-context: auto
  inspection-recursion-limit: 3000
```

Be advised, however, that this may require lots of RAM for even modestly sized rule sets. Also be aware that having additional CPU's available provides a greater performance boost than having more RAM available. That is, it would be better to spend money on CPU's instead of RAM when configuring a system.

It may also lead to significantly longer rule loading times.

Statistics

The stats.log produces statistics records on a fixed interval, by default every 8 seconds.

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
      dropped_smbus: 0
```

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.

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

Ignoring Traffic

In some cases there are reasons to ignore certain traffic. Maybe a trusted host or network, or a site. This document lists some strategies for ignoring traffic.

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)".

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 http.log are still generated for this traffic.

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
```

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
```

Rule Profiling

```
----------------------------------------------------------------------
Date: 9/5/2013 -- 14:59:58
----------------------------------------------------------------------
 Num      Rule         Gid      Rev      Ticks      %      Checks   Matches  Max Ticks   Avg Ticks
-------- ------------ -------- -------- ------------ ------ -------- -------- ----------- ------------
1        2210021      1        3        12037        4.96   1        1        12037       12037.00
2        2210054      1        1        107479       44.26  12       0        35805       8956.58
3        2210053      1        1        4513         1.86   1        0        4513        4513.00
4        2210023      1        1        3077         1.27   1        0        3077        3077.00
5        2210008      1        1        3028         1.25   1        0        3028        3028.00
6        2210009      1        1        2945         1.21   1        0        2945        2945.00
7        2210055      1        1        2945         1.21   1        0        2945        2945.00
8        2210007      1        1        2871         1.18   1        0        2871        2871.00
9        2210005      1        1        2871         1.18   1        0        2871        2871.00
10       2210024      1        1        2846         1.17   1        0        2846        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

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.

Installation

On Ubuntu, install the libtcmalloc-minimal0 package:

```
apt-get install libtcmalloc-minimal0
```

On Fedora, install the gperftools-libs package:

```
yum install gperftools-libs
```

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
```

CONFIGURATION

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.

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
```

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
```

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
```

User and group

It is possible to set the user and group to run Suricata as:

```
run-as:
  user: suri
  group: suri
```

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.

Splitting configuration in multiple files

Some users might have a need or a wish to split their suricata.yaml file in to seperate 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.

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/
```

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.

```
# 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
    #  mode: list ## possible values: list (default), channel
    #  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
    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: yes                  # enable dumping of http fields
          tls: yes                   # enable dumping of tls fields
          ssh: yes                   # enable dumping of ssh fields
```

```
        smtp: yes                # enable dumping of smtp fields

        # Enable the logging of tagged packets for rules using the
        # "tag" keyword.
        tagged-packets: yes

        # 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
          # 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:
        # 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
    - 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]

    - ssh
    - stats:
        totals: yes       # stats for all threads merged together
        threads: no       # per thread stats
        deltas: no        # include delta values
    # bi-directional flows
    - flow
    # uni-directional flows
    #- netflow
```

For more advanced configuration options, see *Eve JSON Output*.

The format is documented in *Eve JSON Format*.

Log output for use with Barnyard (unified.log)

This log only supports IPv4. Its information will be stored in the default logging directory. This log is designed to be stored in a binary format on the hard disc, where it will be further processed by Barnyard. Barnyard can store the output in a database, so Suricata can work on other important tasks. Barnyard can add the files in the Mysql-database, send them to Sguil or several other output options.

There is a size-limit to the log-file: If Suricata generates an alert, it stores this alert in a unified-file. Suricata keeps continuing doing that, until the file has reached its limit. Which in the default case is at 32 MB. At that point Suricata generates a new file and the process starts all over again. Barnyard keeps on processing these files. To prevent Suricata from filling up the hard disc, a size limit is enforced. When the limit is reached, the file will 'role-over', creating a new file. Barnyard removes old files. To every file, Suricata adds a time stamp, so it is easy to see which one came first and which one is the latter.

```
-Unified-log:                       #The log-name.
   enabled: no                      #This log is not enabled. Set 'yes' to enable.
   filename: unified.log            #The name of the file in the default logging directory.
   limit: 32                        #The file size limit in megabytes.
```

This output option has been removed in Suricata 1.1rc1 (see ticket #353).

Alert output for use with Barnyard (unified.alert)

This log only supports IPv4. Its information will be stored in the default logging directory. For further information read the above information about (2) unified.log)

```
-Unified-alert:                     #The log-name.
   enabled: no                      #This log is not enabled. Set 'yes' to enable.
   filename: unified.alert          #The name of the file in the default logging directory.
   limit: 32                        #The file size limit in megabytes.
```

This output option has been removed in Suricata 1.1rc1 (see ticket #353).

Alert output for use with Barnyard2 (unified2.alert)

This log also supports IPv6 in addition to IPv4. It's information will be stored in the default logging directory. For further information read the above information about 2. unified.log.

```
- unified2-alert:                 #The log-name.
    enabled: yes                  #This log is enabled. Set 'no' to disable.
    filename: unified2.alert      #The name of the file in the default logging directory.
    limit: 32                     #The file size limit in megabytes.
```

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
```

```
- 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
                                  # overwritten while restarting Suricata.
    extended: yes                 # If set to yes more information is written about the event.
```

A line based log of DNS queries and replies (dns.log)

This log keeps track of all DNS events (queries and replies). It contains the type of DNS activity that has been performed, the requested / replied domain name and relevant data suck as client, server, ttl, resource record data. This logging can also be performed through the use of the *Eve-log capability* which offers easier parsing.

Example of the apperance of a DNS log of a query with a preceding reply:

```
07/01/2014-04:07:08.768100 [**] Query TX 14bf [**] zeustracker.abuse.ch [**] A [**] 192.168.1.6:37681
07/01/2014-04:07:08.768100 [**] Response TX 14bf [**] zeustracker.abuse.ch [**] A [**] TTL 60 [**] 20
```

Non-existant domains and other DNS errors are recorded by the text representation of the rcode field in the reply (see RFC1035 and RFC2136 for a list). In the example below a non-existent domain is resolved and the NXDOMAIN error logged:

```
02/25/2015-22:58:40.499385 [**] Query TX a3ce [**] nosuchdomainwfqwdqwdqw.com [**] A [**] 192.168.40.
02/25/2015-22:58:40.499385 [**] Response TX a3ce [**] NXDOMAIN [**] 192.168.40.2:53 -> 192.168.40.10:
02/25/2015-22:58:40.499385 [**] Response TX a3ce [**] NXDOMAIN [**] 192.168.40.2:53 -> 192.168.40.10:
```

Configuration options:

```
- dns-log:                        # The log-name
    enabled: yes                  # If this log is enabled. Set 'no' to disable
    filename: dns.log             # Name of this file this log is written to in the default logging dir
    append: yes                   # If this option is set to yes, the (if any exists) dns.log file wil
    filetype: regular / unix_stream / unix_dgram
```

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).

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
                                    # 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 / ).
    interval: 8                   #The default amount of time after which the file will be
                                  #refreshed.
    append: yes/no                #If this option is set to yes, the last filled fast.log-file will not
                                  #overwritten while restarting Suricata.
```

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
                                  #Emergency, Alert, Critical, Error, Warning, Notice, Info and Debug.
```

Drop.log, a line based information for dropped packets

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
                                   #overwritten while restarting Suricata. If set to 'no' the last fille
```

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
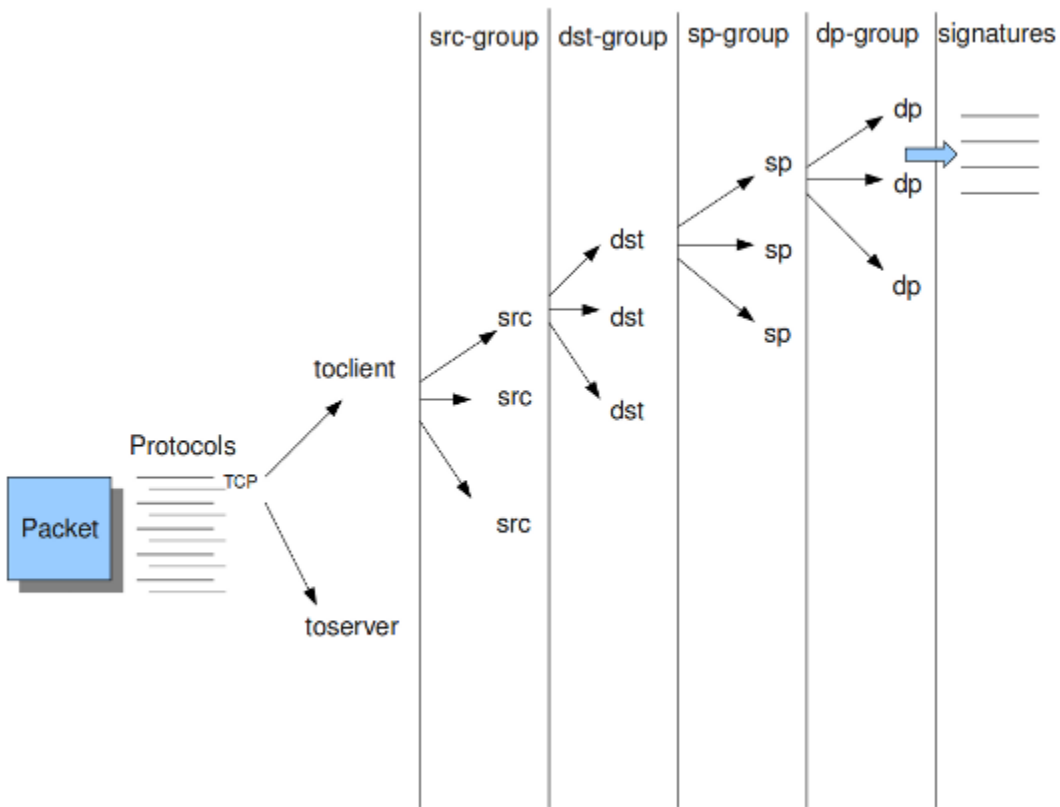
src	Stands for source IP-address.
dst	Stands for destination IP-address.
sp	Stands for source port.
dp	Stands for destination port.

Example 5 Detail grouping tree
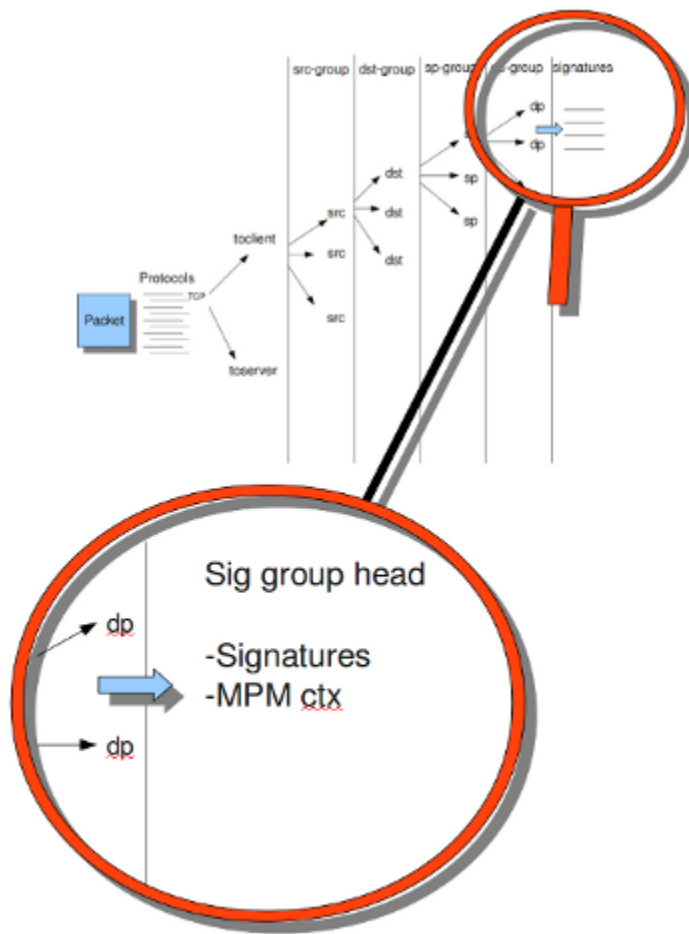
Prefilter Engines

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
```

CUDA (Compute United Device Architecture)

Suricata utilizes CUDA for offloading CPU intensive tasks to the (NVIDIA) GPU (graphics processing unit). Suricata supports an experimental multi-pattern-matcher using CUDA. Only if you have compiled Suricata with CUDA (by entering –enable-cuda in the configure stage) you can make use of these features. There are several options for CUDA. The option 'packet_buffer_limit' designates how many packets will be send to the GPU at the same time. Suricata sends packets in 'batches', meaning it sends multiple packets at once. As soon as Suricata has collected the amount of packets set in the 'packet_buffer_limit' option, it sends them to the GPU. The default amount of packets is 2400.

The option 'packet_size_limit' makes sure that packets with payloads bigger than a certain amount of bytes will not be send to the GPU. Other packets will be send to the GPU. The default setting is 1500 bytes.

The option 'packet_buffers' designates the amount of buffers that will be filled with packets and will be processed. Buffers contain the batches of packets. During the time these filled buffers are being processed, new buffers will be filled.

The option 'batching_timeout' can have all values higher than 0. If a buffers is not fully filled after a period of time (set in this option 'batching_timeout'), the buffer will be send to the GPU anyway.

The option 'page_locked' designates whether the page locked memory will or will not be used. The advantage of page locked memory is that it can not be swapped out to disk. You would not want your computer to use your hard disk for Suricata, because it lowers the performance a lot. In this option you can set whether you still want this for CUDA or not.

The option 'device_id' is an option within CUDA to determine which GPU should be turned to account.(If there is only one GPU present at your computer, there is no benefit making use of the 'device-id' option.) To detect the id of your GPU's, enter the following in your command line:

```
suricata --list-cuda-cards
```

With the option 'cuda_streams' you can determine how many cuda-streams should be used for asynchronous processing. All values > 0 are valid. For this option you need a device with Compute Capability > 1.0 and page_locked enabled to have any effect.

```
cuda:
  -mpm:
    packet_buffer_limit: 2400
    packet_size_limit: 1500
    packet_buffers: 10
    batching_timeout: 1
    page_locked: enabled
    device_id: 0
    cuda_streams: 2
```

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 analysed 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
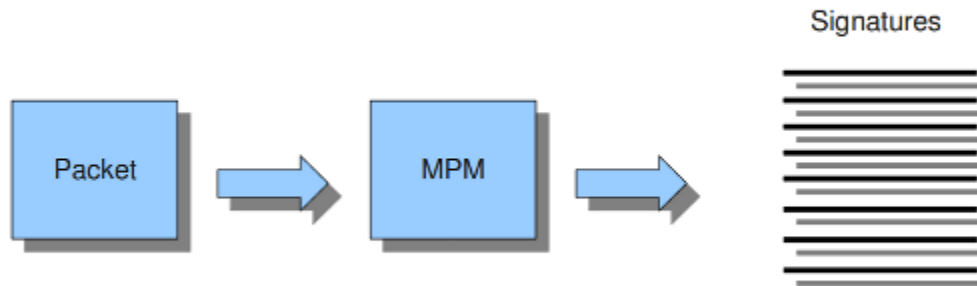
Suricata offers various implementations of different multi-pattern-matcher algorithm's. These can be found below.

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.
```

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 corre
Second: TCP-network traffic comes in as packets. The Stream-Assembly engine reconstructs the original
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
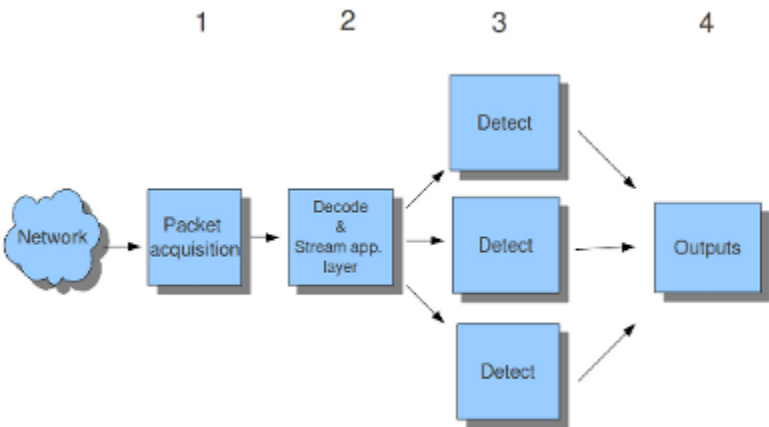

```
Packet acquisition:            Reads packets from the network
Decode:                        Decodes packets.
Stream app. Layer:             Performs stream-tracking and reassembly.
```

```
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. De detect threads running on other cores will process more packets. This is only the case after setting the option at 'yes'.

Example 7 Balancing workload
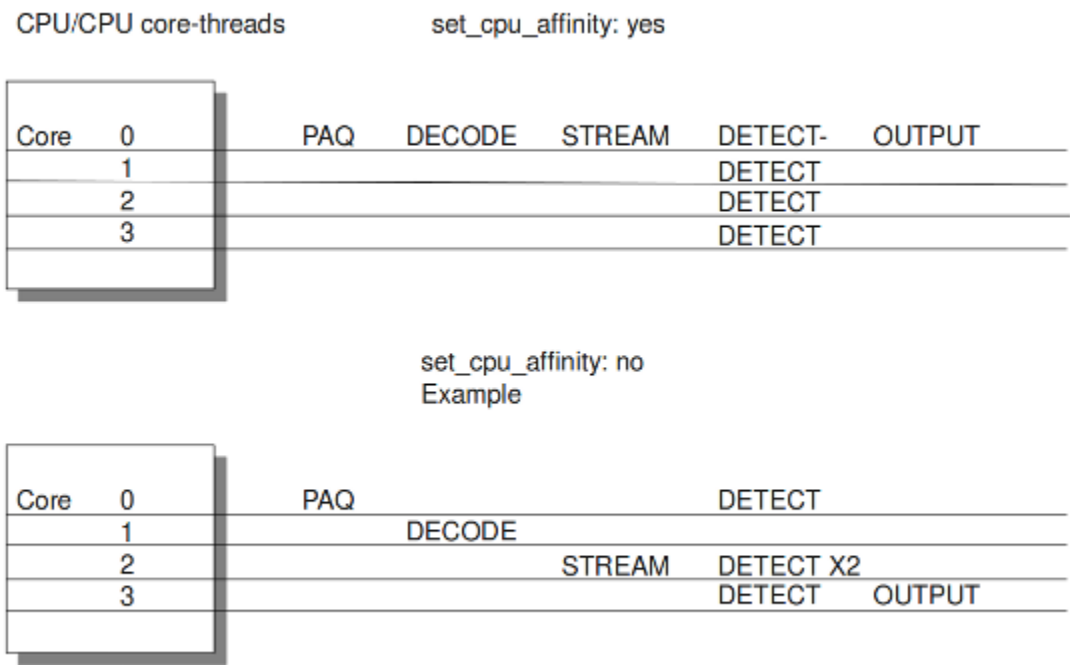
You can set the detect-thread-ratio:

```
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-, decode-, stream-, detect-, verdict-, reject- and outputs-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' 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
  - decode-cpu-set:
      cpu: [ 0, 1 ]
      mode: "balanced"
  - stream-cpu-set:
      cpu: [ "0-1" ]
  - detect-cpu-set:
      cpu: [ "all" ]
      mode: "exclusive" # run detect threads in these cpus
      # Use explicitely 3 threads and don't compute number by using
      # detect-thread-ratio variable:
      # threads: 3
      prio:
        low: [ 0 ]
        medium: [ "1-2" ]
        high: [ 3 ]
        default: "medium"
  - verdict-cpu-set:
      cpu: [ 0 ]
      prio:
        default: "high"
  - reject-cpu-set:
      cpu: [ 0 ]
      prio:
        default: "low"
  - output-cpu-set:
      cpu: [ "all" ]
      prio:
         default: "medium"
```

Relevant cpu-affinity settings for IDS/IPS modes

IDS mode

Runmode AutoFp:

```
management-cpu-set - used for management (example - flow.managers, flow.recyclers)
recive-cpu-set - used for receive and decode
detect-cpu-set - used for streamtcp,detect,output(logging),reject
```

Rumode Workers:

```
management-cpu-set - used for management (example - flow.managers, flow.recyclers)
detect-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)
recive-cpu-set - used for receive and decode
detect-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)
detect-cpu-set - used for receive,streamtcp,decode,detect,output(logging),respond/reject, verdict
```

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
```

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
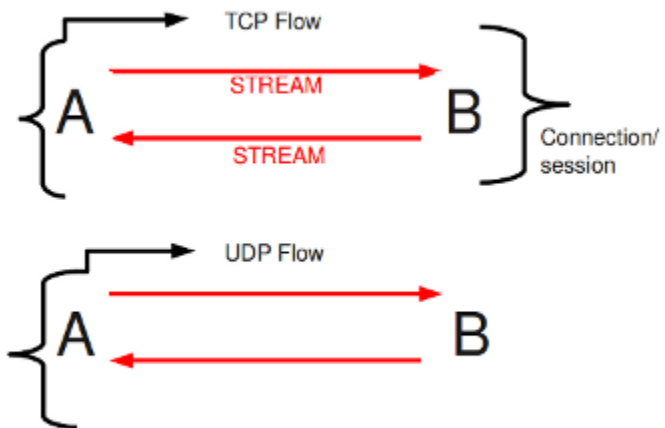
Example 10 Tuple
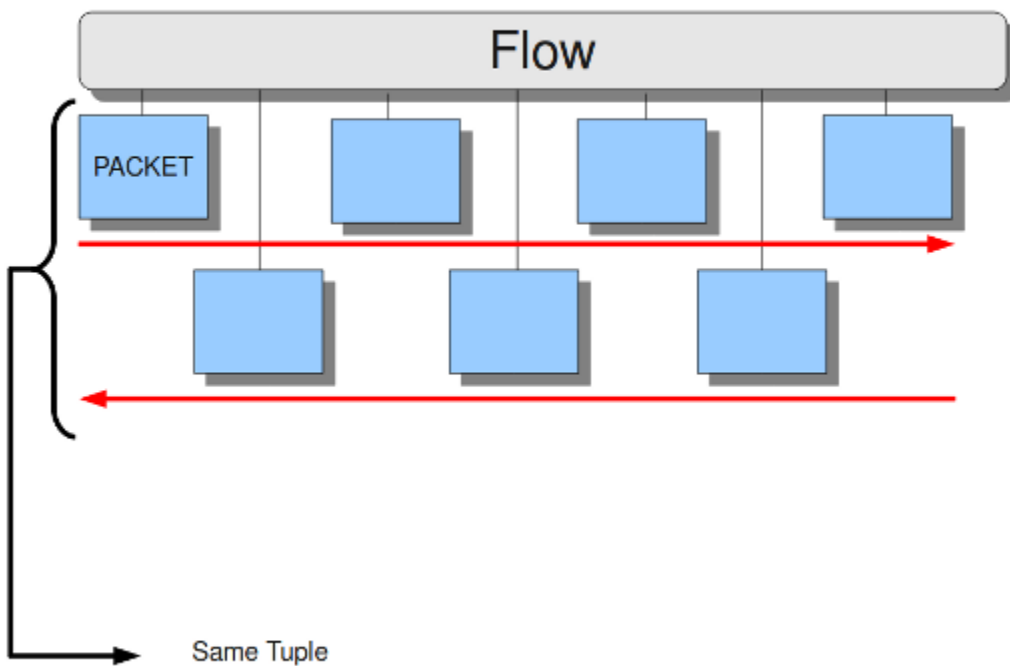
Keeping track of all these flows, uses memory. The more flows, the more memory it will cost.

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 se
                                 #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 st
    established: 300             #Time-out in seconds after the last activity in this flow in a Establ
                                 #state.
    emergency_new: 10           #Time-out in seconds after the last activity in this flow in a New st
```

```
                                         #during the emergency mode.
      emergency_established: 100  #Time-out in seconds after the last activity in this flow in a Establi
                                         #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
  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 recognised 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: 33554432             #Amount of flow-information (in bytes) to keep in memory.
  checksum_validation: yes     #Validate packet checksum, reject packets with invalid checksums.
```

The option 'max_sessions' is the limit for concurrent sessions. It prevents Suricata from using all memory for sessions.

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 running after a few 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*.

```
max_sessions: 262144          # 256k concurrent sessions
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
```

Example 11 Normal/IDS mode

Suricata inspects traffic in chunks.
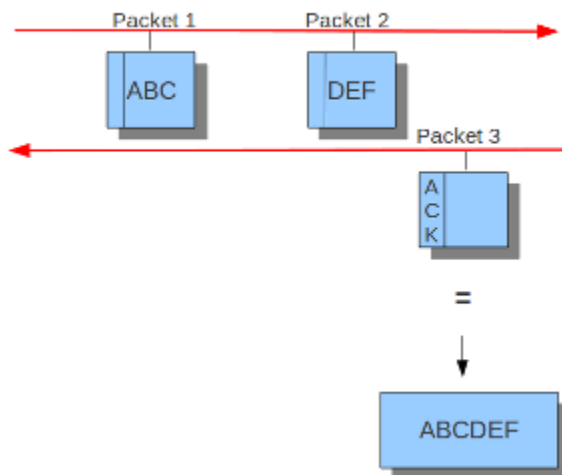
Example 12 Inline/IPS Sliding Window

Suricata inspects traffic in a sliding window manner.
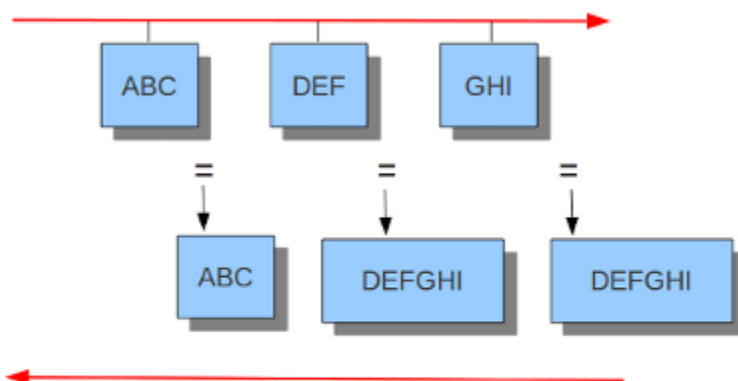
Sliding window = 6

Example 13 Normal/IDS (reasembly on ACK'D data)

Example 14 Inline/IPS (reassembly on UNACK'D data)
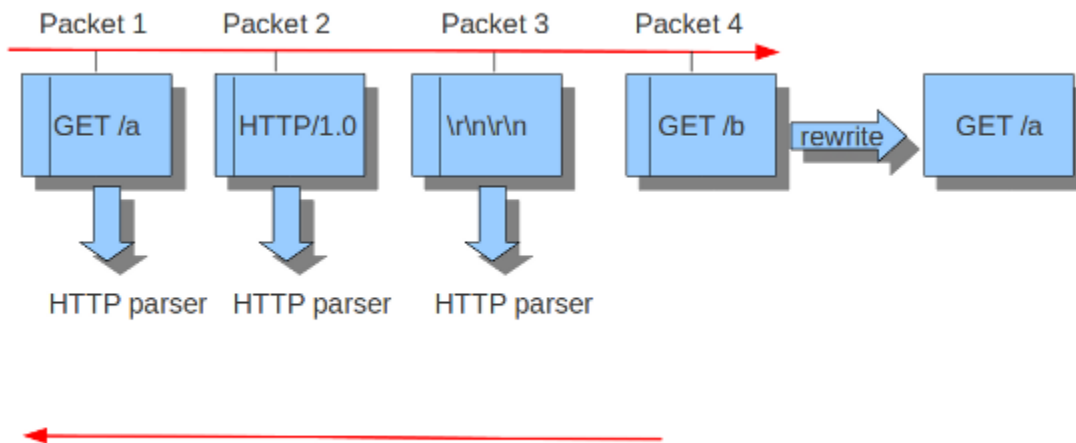
The reassembly-engine has to keep packets in memory to be able to make a reassembled stream. It can make use of the amount of bytes set below. Reassembling a stream is an expensive operation. In the option depth you can set the depth (in a stream) of the reassembling. By default this is 1MB.

```
reassembly:
  memcap: 67108864              #Amount of packets (in bytes) to keep in memory.
  depth: 1048576                #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
```

Example 15 Stream reassembly

toserver_chunk_size: 10
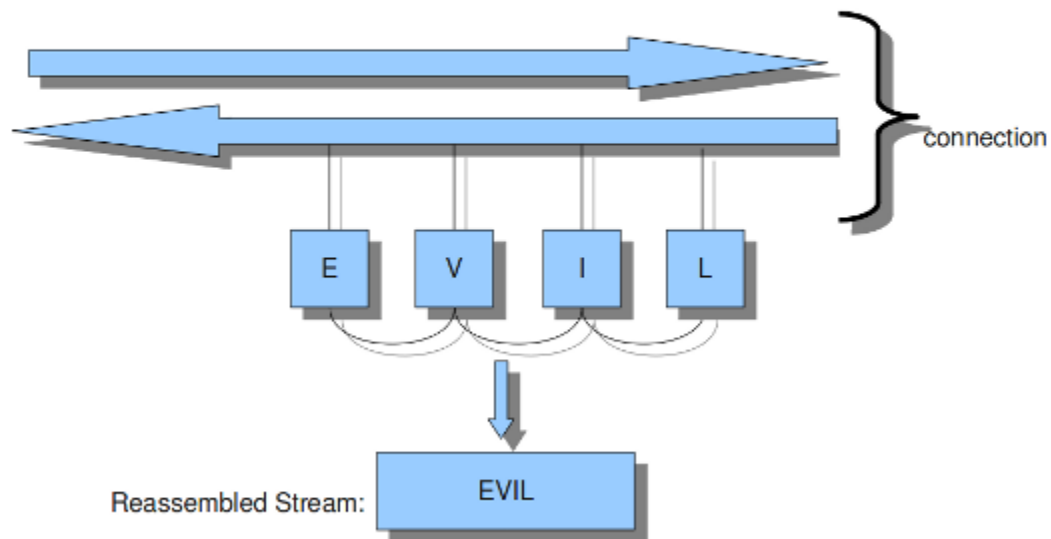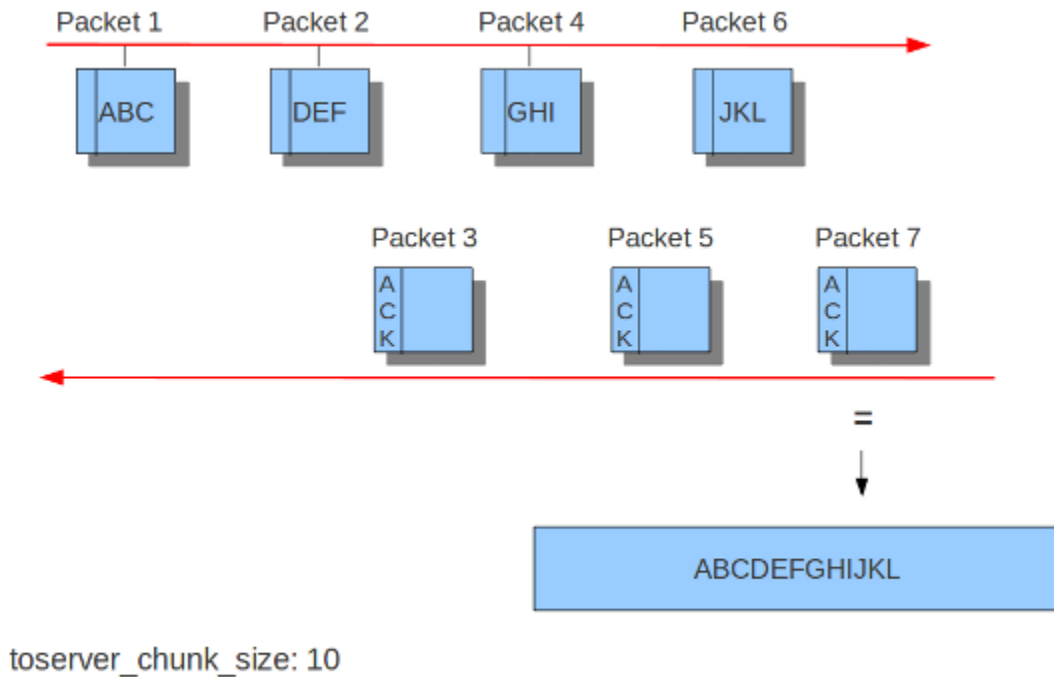
Application Layer Parsers

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 a open source web -server program.

Beside 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 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 the request-body-limit.

The version-specific personalities know exactly how web servers behave, and emulate that. The IDS personality (will be GENERIC in the future) 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 body's 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.

HTTP response body's are often big, so they take a lot of time to process which has a significant impact on the performance. With the option 'response-body-limit' you can set the limit (in bytes) of the server-body that will be inspected. Setting it to 0 will inspect all of the body.

```
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
```

As of 1.4, 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-backslash-separators: yes

# 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-compress-separators: yes

# This parameter is used to predict how a server will react when control
# characters are present in a request path, but does not affect path
# normalization.  Accepted values – none or status_400 */
#path-control-char-handling: none

# Controls the UTF-8 treatment of request paths. One option is to only
# validate path as UTF-8. In this case, the UTF-8 flags will be raised
# as appropriate, and the path will remain in UTF-8 (if it was UTF-8 in
# the first place). The other option is to convert a UTF-8 path into a
# single byte stream using best-fit mapping.  Accepted values – yes, no.
#path-convert-utf8: 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
# backslash_separators option is also enabled, encoded backslash
# characters will be converted too (and subseqently normalized to
# forward slashes).  Accepted values – yes, no.
#path-decode-separators: 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-decode-u-encoding: yes

# Configures how server reacts to invalid encoding in path.  Accepted
# values – preserve_percent, remove_percent, decode_invalid, status_400
#path-invalid-encoding-handling: preserve_percent

# Configures how server reacts to invalid UTF-8 characters in path.
# This setting will not affect path normalization; it only controls what
# response status we expect for a request that contains invalid UTF-8
# characters.  Accepted values – none, status_400.
#path-invalid-utf8-handling: none

# Configures how server reacts to encoded NUL bytes. Some servers will
# terminate path at NUL, while some will respond with 400 or 404. When
# the termination option is not used, the NUL byte will remain in the
# path.  Accepted values – none, terminate, status_400, status_404.
# path-nul-encoded-handling: none

# Configures how server reacts to raw NUL bytes. Some servers will
# terminate path at NUL, while some will respond with 400 or 404. When
```

```
# the termination option is not used, the NUL byte will remain in the
# path.  Accepted values - none, terminate, status_400, status_404.
path-nul-raw-handling: none

# Sets the replacement characater 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.
#set-path-replacement-char: ?

# Controls what the library does when it encounters an Unicode character
# where only a single-byte would do (e.g., the %u-encoded characters).
# Conversion always takes place; this parameter is used to correctly
# predict the status code used in response. In the future there will
# probably be an option to convert such characters to UCS-2 or UTF-8.
# Accepted values - bestfit, status_400 and status_404.
#set-path-unicode-mapping: bestfit
```

Engine output

Logging configuration

The logging subsystem can display all output except alerts and events. It gives information at runtime about what the engine is doing. This information can be displayed during the engine startup, at runtime and while shutting the engine down. For informational messages, errors, debugging, etc.

The log-subsystem has several log levels:

Error, warning, informational and 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.

There are more levels: emergency, alert, critical and notice, but those are not used by Suricata yet. This option can be changed in the configuration, but can also be overridden in the command line by the environment variable: SC_LOG_LEVEL .

```
logging:
  default-log-level: info
```

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

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.
  - file:                                   #Output stored in a file.
      enabled: no                           #This option is not enabled.
      filename: /var/log/suricata.log       #Filename and location on disc.
  - syslog:                                 #This is a program to direct log-output to several d:
      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.
```

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 be:
                                 #processed by Suricata.
   repeat_mask: 1
   route_queue: 2                #Here you can assign the queue-number of the tool that Suricata has t
                                 #send the packets to after processing them.
```

Example 1 NFQ1

mode: accept
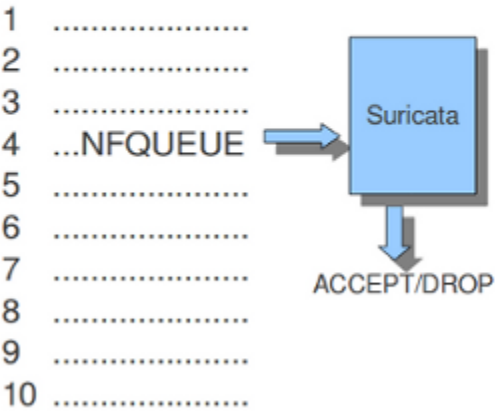
Example 2 NFQ

mode: repeat

Example 3 NFQ

mode: route

iptables and NFQ
Mode: route
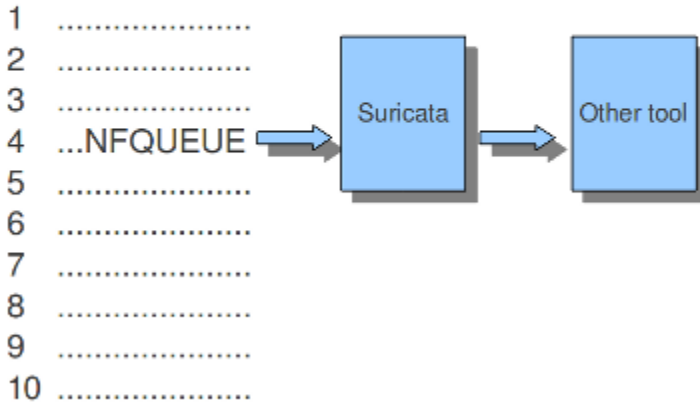
Ipfw

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.
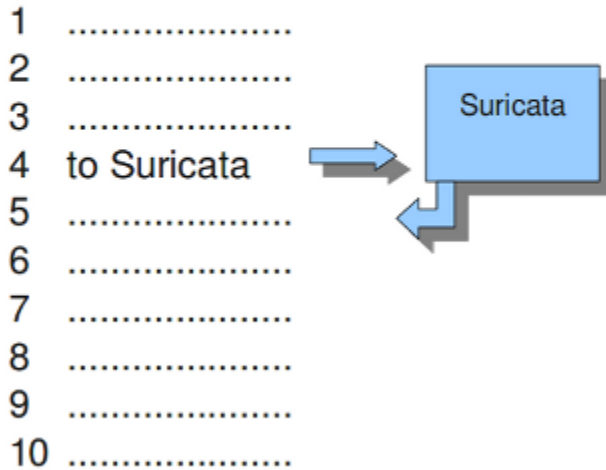
Rules

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;

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
```

```
                                                               #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
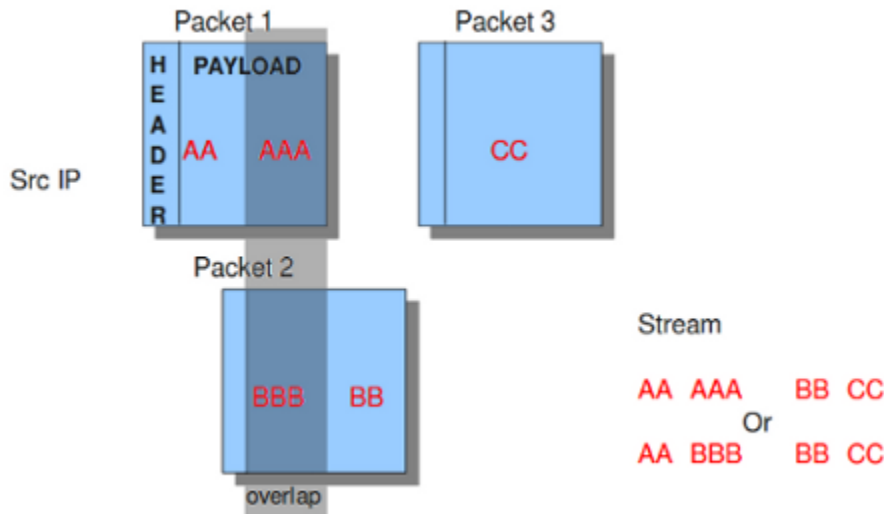
In the configuration-file, the operating-systems are listed. You can add your IP-address behind the name of the operating system you make use of.

```
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: []
```

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 organises 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-Analyis 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
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

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 informat
                                                   #stored.
  append: yes                                      #If set to yes, new packet profiling information wil
                                                   #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 defa
    filename: packet_stats.csv               #name of the file in which csv packet profiling inform
                                             #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.

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

  # Completely stop processing TLS/SSL session after the handshake
  # completed. If bypass is enabled this will also trigger flow
  # bypass. If disabled (the default), TLS/SSL session is still
  # tracked for Heartbleed and other anomalies.
  #no-reassemble: yes
```

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 `no-reassemble` option controls the behaviour after the handshake.

If `no-reassemble` is set to `true`, all processing of this session is stopped. No further parsing and inspection happens. If `bypass` is enabled this will lead to the flow being bypassed, either inside Suricata or by the capture method if it supports it.

If `no-reassemble` is set to `false`, which is the default, Suricata will continue to track the SSL/TLS session. Inspection will be limited, as `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.

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
```

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.

Global-Thresholds

Thresholds can be configured in the rules themselves, see Rule Thresholding. They are often set by rule writers based on their intel for creating a rule combined with a judgement on how often a rule will alert.

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.

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>, ip <ip|subnet>
```

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
```

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 evironment.

```
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 #425.

Rate_filter

TODO

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.

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
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.

Decoder alerts

snort.conf

```
# Stop generic decode events:
config disable_decode_alerts

# Stop Alerts on experimental TCP options
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.

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.

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
#
# 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 REAL
#
# 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 se
#
# 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
```

Log directory

snort.conf

```
# Configure default log directory for snort to log to.  For more information see snort -h command lin
#
# config logdir:
```

suricata.yaml

```
default-log-dir: /var/log/suricata/
```

This value is overridden by the -l commandline option.

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.

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
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
```

Log Rotation

Suricata can generate lot of output, so it's important to manage the files to avoid issues with disks filling up.

A HUP signal sent to Suricata will force it to reopen the logfiles.

Example logrotate file:

```
/var/log/suricata/*.log /var/log/suricata/*.json
{
    rotate 3
    missingok
    nocompress
    create
    sharedscripts
    postrotate
            /bin/kill -HUP $(cat /var/run/suricata.pid)
    endscript
}
```

Multi Tenancy

Introduction

Multi tenancy support allows for different rule sets with different rule vars.

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) or vlan
- 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

> – tenant id: tenant to associate with the vlan id

```
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"

    ...
```

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.

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
```

REPUTATION

IP Reputation

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

If the "rule-reloads" option is enabled, 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.

IP Reputation Rules

IP Reputation can be used in rules through a new rule directive "iprep".

iprep

The iprep directive matches on the IP reputation information for a host.

```
iprep:<side to check>,<cat>,<operator>,<value>
```

side to check: <any|src|dst|both>

cat: the category short name

operator: <, >, =

value: 1-127

Example:

```
alert ip $HOME_NET any -> any any (msg:"IREP internal host talking to CnC server"; flow:to_server; 
```

IP-only

The "iprep" keyword is compatible to "IP-only" rules. This means that a rule like:

```
alert ip any any -> any any (msg:"IREP High Value CnC"; iprep:src,CnC,>,100; sid:1; rev:1;)
```

will only be checked once per flow-direction.

For more information about IP Reputation see IP Reputation Config and IP Reputation Format.

IP Reputation Format

Description of IP Reputation file formats. For the configuration see IP Reputation Config and IP Reputation Rules 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>,<discription>
```

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>,<cat>,<score>

The IP is an IPv4 address in the quad-dotted notation. The category is the number as defined in the categories file and the value is a number between 1 and 127. 0 means no data.

Example:

```
1.2.3.4,1,101
1.1.1.1,6,88
```

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 Rules and IP Reputation Format.

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
```

SETTING UP IPS/INLINE FOR LINUX

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
```

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.
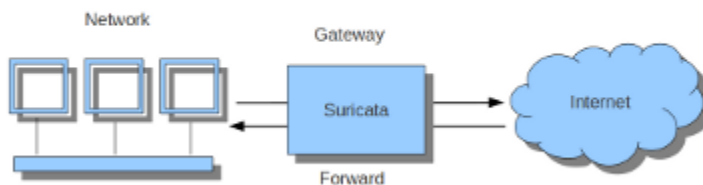
If Suricata is running on a gateway and is meant to protect the computers behind that gateway you are dealing with the first scenario: *forward_ing* . If Suricata has to protect the computer it is running on, you are dealing with the second scenario: *host* (see drawing 2). These two ways of using Suricata 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 iptable 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.
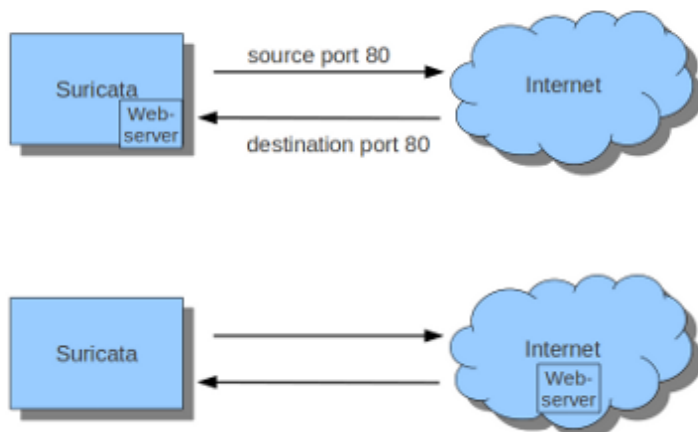
To see if you have set your iptables rules correct make sure Suricata is running and enter:

```
sudo iptables -vnL
```

In the example you can see if packets are being logged.

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:
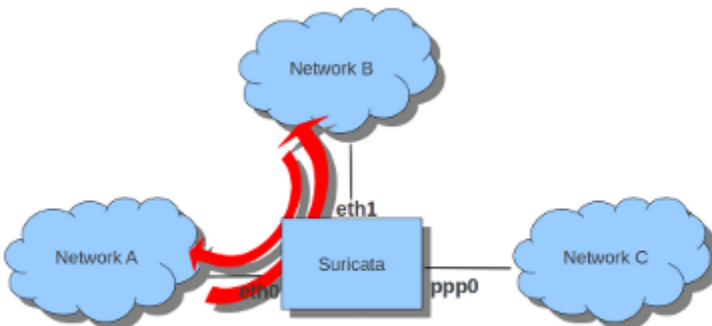

```
sudo iptables -I FORWARD -i eth0 -o eth1 -j NFQUEUE
sudo iptables -I FORWARD -i eth1 -o eth0 -j NFQUEUE
```

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 iptable rules.

To erase all iptable rules, enter:

```
sudo iptables -F
```

OUTPUT

EVE

Eve JSON Output

Suricata can output alerts, http events, dns events, tls events and file info 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.

```
# 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
    #   mode: list ## possible values: list (default), channel
    #   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
    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: yes              # enable dumping of http fields
          tls: yes               # enable dumping of tls fields
          ssh: yes               # enable dumping of ssh fields
          smtp: yes              # enable dumping of smtp fields
```

```
          # Enable the logging of tagged packets for rules using the
          # "tag" keyword.
          tagged-packets: yes

          # 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
            # 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:
        # 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
    - 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]
```

```
        - ssh
        - stats:
            totals: yes       # stats for all threads merged together
            threads: no       # per thread stats
            deltas: no        # include delta values
        # bi-directional flows
        - flow
        # uni-directional flows
        #- netflow
```

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 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
#  mode: list ## possible values: list (default), channel
#  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 ammended with metadata, such as the HTTP record an alert was generated for.

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: yes                  # enable dumping of http fields
    tls: yes                   # enable dumping of tls fields
    ssh: yes                   # enable dumping of ssh fields
    smtp: yes                  # enable dumping of smtp fields
```

DNS

DNS records are logged one log record per query/answer record.

YAML:

```
- 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]
```

To reduce verbosity the output can be filtered by supplying the record types to be logged under `custom`.

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'.

In addition to this, each log can be handled completely separately:

```
outputs:
  - alert-json-log:
      enabled: yes
      filename: alert-json.log
  - dns-json-log:
      enabled: yes
      filename: dns-json.log
  - drop-json-log:
      enabled: yes
      filename: drop-json.log
  - http-json-log:
      enabled: yes
      filename: http-json.log
  - ssh-json-log:
      enabled: yes
      filename: ssh-json.log
```

```
- tls-json-log:
    enabled: yes
    filename: tls-json.log
```

For most output types, you can add multiple:

```
outputs:
  - alert-json-log:
      enabled: yes
      filename: alert-json1.log
  - alert-json-log:
      enabled: yes
      filename: alert-json2.log
```

Except for `drop` for which only a single logger instance is supported.

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",
        "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 specif:
```

Event types

The common part has a field "event_type" to indicate the log type.

```
"event_type":"TYPE"
```

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.

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 statuscode

- "protocol": Protocol / Version of HTTP (ex: HTTP/1.1)

- "http_method": The HTTP method (ex: GET, POST, HEAD)

- "http_refer": The referer for this action

In addition to the extended logging fields one can also choose to enable/add from 47 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]
          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, 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.

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"
}
```

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 type: DNS

Fields

Outline of fields seen in the different kinds of DNS events:

- "type": Indicating DNS message type, can be "answer" or "query".

- "id": <needs explanation>

- "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

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 an IPv4 (resource record type 'A') return:

```
"dns": {
    "type": "answer",
    "id":16000,
    "rrname": "twitter.com",
    "rrtype":"A",
    "ttl":8,
    "rdata": "199.16.156.6"
}
```

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_e
```

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").netflo
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
```

Lua Output

Note: this page new Lua scripting available for outputs. It will be available in 2.1.

Script structure

A script defines 4 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", ".")

    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", ".")

    ts = SCPacketTimeString()
    ipver, srcip, dstip, proto, sp, dp = SCFlowTuple()

    file:write (ts .. " " .. http_host .. " [**] " .. http_uri .. " [**] " ..
            http_ua .. " [**] " .. srcip .. ":" .. sp .. " -> " ..
            dstip .. ":" .. dp .. "\n")
    file:flush()

    http = http + 1
end

function deinit (args)
    SCLogInfo ("HTTP transactions logged: " .. http);
    file:close(file)
end
```

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.

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()
```

flow

```
function init (args)
    local needs = {}
    needs["type"] = "flow"
    return needs
end
```

SCFlowTimeString

```
startts = SCFlowTimeString()
```

SCFlowTuple

```
ipver, srcip, dstip, proto, sp, dp = SCFlowTuple()
```

SCFlowAppLayerProto

Get alproto as string from the flow. If alproto is not (yet) known, it returns "unknown".

Example:

```
function log(args)
    alproto = SCFlowAppLayerProto()
    if alproto ~= nil then
        print (alproto)
    end
end
```

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()
```

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
```

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

TLS

Initialize with:

```
function init (args)
    local needs = {}
    needs["protocol"] = "tls"
    return needs
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
```

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
```

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)

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()
```

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
```

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
```

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.

Popular syslog daemons

- **syslogd** - logs system messages

- **syslog-ng** - logs system messages but also suports 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.

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.

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 (ususally 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.

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 ->
      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
```

```
08/28/12-22:14:30.693856 - Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:11.0) Gecko/20100101 Firefox/11.0
```

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)

FILE EXTRACTION

Architecture

The file extraction code works on top of the HTTP and SMTP parsers. The HTTP parser takes care of dechunking and unzipping the request and/or response data if necessary. The HTTP/SMTP parsers runs on top of the stream reassembly engine.

This means that settings in the stream engine, reassembly engine and the HTTP parser all affect the workings of the file extraction.

What files are actually extracted and stored to disk is controlled by the rule language.

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.

Output

For file extraction two separate output modules were created: "file-log" and "file-store". They need to be enabled in the Suricata.yaml. For "file-store", the "files" drop dir must be configured.

```
- 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-md5: no      # force logging of md5 checksums
```

```
    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
```

Each file that is stored with have a name "file.<id>". The id will be reset and files will be overwritten unless the waldo option is used.

```
- file-log:
    enabled: yes
    filename: files-json.log
    append: yes
    #filetype: regular # 'regular', 'unix_stream' or 'unix_dgram'
    force-magic: no   # force logging magic on all logged files
    force-md5: no     # force logging of md5 checksums
```

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; r
```

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.

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.

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-md5: yes     # force logging of md5 checksums
    #waldo: file.waldo # waldo file to store the file_id across runs
```

Optionally, for JSON output:

```
- file-log:
    enabled: yes
    filename: files-json.log
    append: no
```

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-rout
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"
{ "id": 12, "timestamp": "05\/01\/2012-11:12:57.421420", "ipver": 4, "srcip": "2.23.144.170", "dstip"
```

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-md5: yes    # 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-md5: yes    # force logging of md5 checksums
```

Public SHA1 MD5 data sets

National Software Reference Library - http://www.nsrl.nist.gov/Downloads.htm

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

USING CAPTURE HARDWARE

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 an
[5570] 10/7/2012 -- 13:52:30 - (source-erf-dag.c:288) <Info> (ReceiveErfDagThreadInit) -- Starting pr
```

Napatech Suricata Installation Guide

Contents

- Introduction
- Package Installation
- Basic Configuration
- Advanced Multithreaded Configuration

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 Napatech Software Suite (driver package) comes in two varieties, NAC and OEM. The NAC package distributes deb and rpm packages to ease the installation. The OEM 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.

Package Installation

Note that make, gcc, and the kernel headers are required for installation

Root privileges are also required

Napatech NAC Package

Red Hat Based Distros:

```
$ yum install kernel-devel-$(uname -r) gcc make ncurses-libs
$ yum install nac-pcap-<release>.x86_64.rpm
```

Some distributions will require you to use the –nogpgcheck option with yum for the NAC Software Suite package file:

```
$ yum --nogpgcheck install nac-pcap-<release>.x86_64.rpm
```

Debian Based Distros:

```
$ apt-get install linux-headers-$(uname .r) gcc make libncurses5
$ dpkg .i nac-pcap_<release>_amd64.deb
```

To complete installation for all distros stop ntservice:

```
$ /opt/napatech3/bin/ntstop.sh -m
```

Remove these existing setup files:

```
$ cd /opt/napatech3/config
$ rm ntservice.ini setup.ini
```

Restart ntservice (a new ntservice.ini configuration file will be generated automatically):

```
$ /opt/napatech3/bin/ntstart.sh -m
```

Napatech OEM Package

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
```

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-libra
$ make
$ make install-full
```

Now edit the suricata.yaml file to configure the maximum number of streams to use. If you plan on using the load distribution (RSS - like) feature in the Napatech accelerator, then the list should contain the same number of streams as host buffers defined in ntservice.ini:

```
Napatech:
        # The Host Buffer Allowance for all streams
        # (-1 = OFF, 1 - 100 = percentage of the host buffer that can be held back)
        hba: -1

        # use_all_streams set to "yes" 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.
        use-all-streams: yes

        # The streams to listen on
        streams: [0, 1, 2, 3, 4, 5, 6, 7]
```

Basic Configuration

For the basic installation we will setup the Napatech capture accelerator to merge all physical ports into single stream that Suricata can read from. for this configuration, Suricata will handle the packet distribution to multiple threads.

Here are the lines that need changing in /opt/napatech3/bin/ntservice.ini for best single buffer performance:

```
TimeSyncReferencePriority = OSTime          # Timestamp clock synchronized to the OS
HostBuffersRx = [1,16,0]                     # [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 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
Setup[numaNode=0] = streamid==0         # Set stream ID 0 to NUMA 0
Assign[priority=0; streamid=0]= 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
```

Advanced Multithreaded Configuration

Now let's do a more advanced configuration where we will use the load distribution (RSS - like) capability in the accelerator. We will create 8 streams and setup the accelerator to distribute the load based on a 5 tuple hash. Increasing buffer size will minimize packet loss only if your CPU cores are fully saturated. Setting the minimum buffer size (16MB) will gave the best performance (minimize L3 cache hits) if your CPU cores are keeping up.

Note that it is extremely important that the NUMA node the host buffers are define in is the same physical CPU socket that the Napatech accelerator is plugged into

First let's modify the ntservice.ini file to increase the number and size of the host buffers:

```
HostBuffersRx = [8,256,0]                       # [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 let's assign the streams to host buffers and configure the load distribution. The load distribution will be setup to support both tunneled and non-tunneled traffic. Create a file that contains the ntpl commands below:

```
Delete=All                                      # Delete any existing filters
Setup[numaNode=0] = streamid==0
Setup[numaNode=0] = streamid==1
Setup[numaNode=0] = streamid==2
Setup[numaNode=0] = streamid==3
Setup[numaNode=0] = streamid==4
Setup[numaNode=0] = streamid==5
Setup[numaNode=0] = streamid==6
Setup[numaNode=0] = streamid==7
HashMode[priority=0; InnerLayer4Type=UDP, TCP, SCTP] = HashInner5TupleSorted
HashMode[priority=1; InnerLayer3Type=IPV4] = HashInner2TupleSorted
HashMode[priority=2; Layer4Type=UDP, TCP, SCTP] = Hash5TupleSorted
HashMode[priority=3; Layer3Type=IPV4] = Hash2TupleSorted
HashMode[priority=4]=HashRoundRobin
Assign[priority=0; streamid=(0..7)]= all
```

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
```

Support

Contact a support engineer at: ntsupport@napatech.com

Myricom

From: http://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/ --prefi
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 CPU's. *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 it's 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
```

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
```

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_M
```

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-emulex-booth/

INTERACTING VIA UNIX SOCKET

Introduction

Since 1.4 version, 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 and it is described in this commit message. An example script called suricatasc is provided in the source and installed automatically when updating Suricata.

The unix socket is disabled by default.

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

```
apt-get install libjansson4 libjansson-dev python-simplejson
```

If libjansson is present on the system , unix socket will be compiled in automatically - when you use "–enable-unix-socket" in your configure line.

The creation of the socket is activated by setting enabled to yes 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)

Commands in standard running mode

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

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, du
>>> 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"
```

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:~#
```

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
```

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 don't know from where the script has been run.

To know how much 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"
```

Build your own client

The protocol is documented in the following page https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Unix_Socket#Protoc

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/d
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']}
```

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. This method is used here: source:scripts/suricatasc/suricatasc.in#L43

MAN PAGES

Suricata

SYNOPSIS

suricata [OPTIONS] [BPF FILTER]

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).

OPTIONS

-h
Display a brief usage overview.

-V
Displays the version of Suricata.

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

-T
Test configuration.

-v
The -v option enables more verbosity of Suricata's output. Supply multiple times for more verbosity.

-r `<path>`
Run in pcap offline mode reading files from pcap file.

-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.

--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 deamon (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.

--mpipe
> Enable packet capture using the TileGX mpipe interface.

--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.

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.

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.

FILES AND DIRECTORIES

/usr/local/etc/suricata/suricata.yaml Default location of the Suricata configuration file.

/usr/local/var/log/suricata Default Suricata log directory.

BUGS

Please visit Suricata's support page for information about submitting bugs or feature requests.

NOTES

- Suricata Home Page

     https://suricata-ids.org/

- Suricata Support Page

     https://suricata-ids.org/support/

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
- Rob MacGregor
- Russel Fulton
- Victor Julien
- Vincent Fang
- Zach Rasmor

NINETEEN

LICENSES

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.

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.

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.

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 LIM-ITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PUR-POSE. 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 SERVIC-ING, 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 PRO-GRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPE-CIAL, 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 IN-ACCURATE 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

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.

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.

Section 1 – Definitions.

1. **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.

2. **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.

3. **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.

4. **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.

5. **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.

6. **Licensed Material** means the artistic or literary work, database, or other material to which the Licensor applied this Public License.

7. **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.

8. **Licensor** means the individual(s) or entity(ies) granting rights under this Public License.

9. **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.

10. **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.

11. **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.

12. **You** means the individual or entity exercising the Licensed Rights under this Public License. Your has a corresponding meaning.

Section 2 – Scope.

1. **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:

   1. reproduce and Share the Licensed Material, in whole or in part, for NonCommercial purposes only; and

   2. 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.**

1. **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.

2. **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).

2. **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.

Section 3 – License Conditions.

Your exercise of the Licensed Rights is expressly made subject to the following conditions.

1. **Attribution.**

   1. If You Share the Licensed Material (including in modified form), You must:

      1. retain the following if it is supplied by the Licensor with the Licensed Material:

      1. 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);

      2. a copyright notice;

      3. a notice that refers to this Public License;

      4. a notice that refers to the disclaimer of warranties;

      5. a URI or hyperlink to the Licensed Material to the extent reasonably practicable;

      2. indicate if You modified the Licensed Material and retain an indication of any previous modifications; and

      3. 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.

Section 4 – Sui Generis Database Rights.

Where the Licensed Rights include Sui Generis Database Rights that apply to Your use of the Licensed Material:

1. 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;

2. 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

3. 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.

Section 5 – Disclaimer of Warranties and Limitation of Liability.

1. **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.**

2. **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.**

3. 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.

Section 6 – Term and Termination.

1. 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.

2. 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.

3. 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.

4. Sections 1, 5, 6, 7, and 8 survive termination of this Public License.

Section 7 – Other Terms and Conditions.

1. The Licensor shall not be bound by any additional or different terms or conditions communicated by You unless expressly agreed.

2. 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.

Section 8 – Interpretation.

1. 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.

2. 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.

3. 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.

4. 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.

Suricata Source Code

The Suricata source code is licensed under version 2 of the GNU General Public License.

Suricata Documentation

The Suricata documentation (this documentation) is licensed under the Creative Commons Attribution-NonCommercial 4.0 International Public License.

Symbols

C