8.1. Rules Format

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

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

There are a number of free rulesets that can be used via suricata-update. To aid in learning about writing rules, the Emerging Threats Open ruleset is free and a good reference that has a wide range of signature examples.

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

A rule/signature consists of the following:

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

An example of a rule is as follows:

```
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"HTTP GET
Request Containing Rule in URI"; flow:established,to_server;
http.method; content:"GET"; http.uri; content:"rule"; fast_pattern;
classtype:bad-unknown; sid:123; rev:1;)
```

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

We will be using the above signature as an example throughout this section, highlighting the different parts of the signature.

8.1.1. Action

```
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"HTTP GET
Request Containing Rule in URI"; flow:established,to_server;
http.method; content:"GET"; http.uri; content:"rule"; fast_pattern;
classtype:bad-unknown; sid:123; rev:1;)
```

Valid actions are:

- alert - generate an alert.
- pass - stop further inspection of the packet.
- drop - drop packet and generate alert.
- reject - send RST/ICMP unreach error to the sender of the matching packet.
- rejectsrc - same as just *reject*.
- rejectdst - send RST/ICMP error packet to receiver of the matching packet.
- rejectboth - send RST/ICMP error packets to both sides of the conversation.

Note

In IPS mode, using any of the *reject* actions also enables *drop*.

For more information see Action-order.

8.1.2. Protocol

```
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"HTTP GET
Request Containing Rule in URI"; flow:established,to_server;
http.method; content:"GET"; http.uri; content:"rule"; fast_pattern;
classtype:bad-unknown; sid:123; rev:1;)
```

The protocol value will limit what protocol(s) the signature will be applied to:

- ip (ip stands for 'all IP packets' or 'any IP packet')
- tcp (for TCP traffic)
- udp
- icmp (both ICMPv4 and ICMPv6)
- icmpv4

- icmpv6
- ipv4/ip4 - just IPv4
- ipv6/ip6 - just IPv6
- pkthdr (for matching on packets with decoder events)
- ether - Ethernet packets
- arp - ARP packets specifically

There are a couple of additional TCP related protocol options:

- tcp-pkt (for matching content in individual tcp packets)
- tcp-stream (for matching content only in a reassembled tcp stream)

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

- http (either HTTP1 or HTTP2)
- http1
- http2
- tls (this includes ssl)
- quic
- ftp (ftp command channel)
- ftp-data (ftp data channel)
- smb
- dns
- doh2 (dns over http/2)
- mdns
- dcerpc
- ldap
- dhcp
- ssh
- smtp
- imap
- pop3
- modbus (disabled by default)
- dnp3 (disabled by default)
- enip (disabled by default)
- nfs
- ike
- krb5
- bittorrent-dht
- mqtt
- ntp

- dhcp
- rfb
- rdp
- snmp
- tftp
- sip
- telnet
- websocket
- pgsql

This list of protocols can be obtained via `suricata --list-rule-protos`.

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

If you have a signature with the protocol declared as 'http', Suricata makes sure the signature will only match if the TCP stream contains http traffic.

8.1.2.1. Matching on non-IP packets

Traditionally the rule language was only about matching on IP packets. For packets that caused decoder events in the layers before IP a special protocol *pkthdr* was added.

```
alert pkthdr any any -> any any (msg:"SURICATA IPv4 packet too
small"; decode-event:ipv4.pkt_too_small; classtype:protocol-
command-decode; sid:2200000; rev:2;)
```

Up until Suricata 8 this protocol was an alias for *alert ip*, but in Suricata 9 it is only to be used in decoder event rules.

8.1.2.2. Explicit rule hooks

In Suricata 8 the protocol field can be used to force evaluation of a rule at a specific explicit protocol state. This takes the format of:

```
<proto>:<hook>
```

Where each application protocol comes with a default set of hooks, as well as per protocol specific hooks.

More details can be found in Explicit rule hook (states).

Note

While developed for the firewall usecase, these hooks can be used in IDS/IPS rules as well.

8.1.3. Source and destination

```
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"HTTP GET
Request Containing Rule in URI"; flow:established,to_server;
http.method; content:"GET"; http.uri; content:"rule"; fast_pattern;
classtype:bad-unknown; sid:123; rev:1;)
```

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

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

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

Normally, you would also make use of variables, such as `$HOME_NET` and `$EXTERNAL_NET`. The suricata.yaml configuration file specifies the IP addresses these concern. The respective `$HOME_NET` and `$EXTERNAL_NET` settings will be used in place of the variables in your rules.

See Rule-vars for more information.

Rule usage examples:

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

Warning

If you set your configuration to something like this:

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

You cannot write a signature using `$EXTERNAL_NET` because it evaluates to 'not any', which is an invalid value.

Note

Please note that the source and destination address can also be matched via the `ip.src` and `ip.dst` keywords (See IP Addresses Match). These keywords are mostly used in conjunction with the dataset feature (Datasets).

8.1.4. Ports (source and destination)

```
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"HTTP GET
Request Containing Rule in URI"; flow:established,to_server;
http.method; content:"GET"; http.uri; content:"rule"; fast_pattern;
classtype:bad-unknown; sid:123; rev:1;)
```

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

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

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

In setting ports you can make use of special operators as well. Operators such as:

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

Rule usage examples:

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

8.1.5. Direction

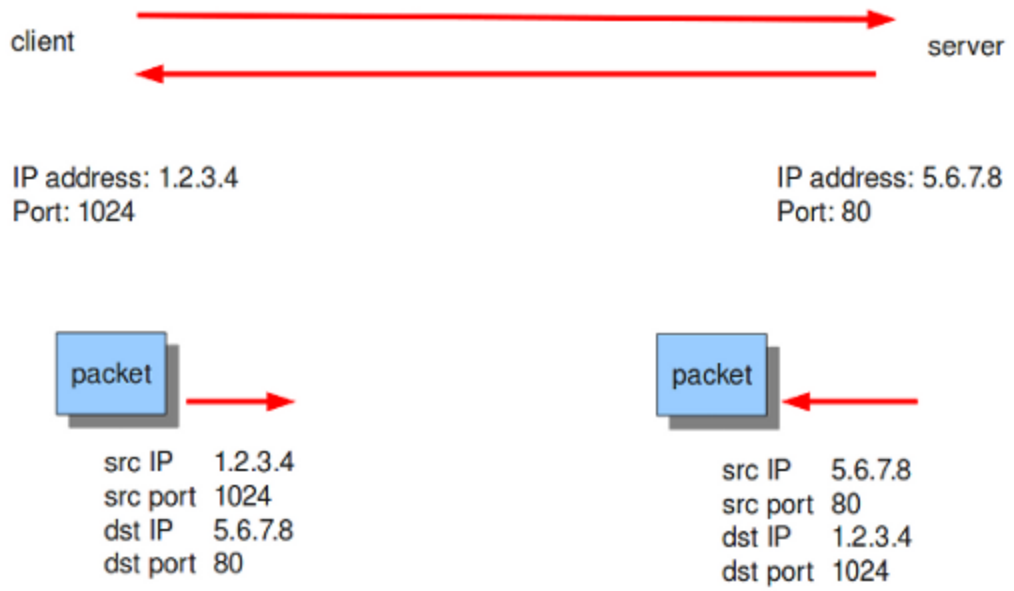
```
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"HTTP GET
Request Containing Rule in URI"; flow:established,to_server;
http.method; content:"GET"; http.uri; content:"rule"; fa
```

```
classtype:bad-unknown; sid:123; rev:1;)
```

The directional arrow indicates which way the signature will be evaluated. In most signatures an arrow to the right (`->`) is used. This means that only packets with the same direction can match. There is also the double arrow (`=>`), which respects the directionality as `->`, but allows matching on bidirectional transactions, used with keywords matching each direction. Finally, it is also possible to have a rule match either directions (`<>`):

```
source -> destination
source => destination
source <> destination  (either directions)
```

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



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

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

Only the traffic from the client to the server will be matched by this rule, as the direction specifies that we do not want to evaluate the response packet.

Now, if we have a rule with the following header:

```
alert tcp 1.2.3.4 any <> 5.6.7.8 80
```

Suricata will duplicate it and use the same rule with headers in both directions :

alert tcp 1.2.3.4 any -> 5.6.7.8 80 alert tcp 5.6.7.8 80 -> 1.2.3.4 any

Warning

There is no 'reverse' style direction, i.e. there is no `<-`.

8.1.5.1. Transactional rules

Here is an example of a transactional rule:

```
alert http any any => 5.6.7.8 80 (msg:"matching both uri and
status"; sid: 1; http.uri; content: "/download"; http.stat_code;
content: "200";)
```

It will match on flows to 5.6.7.8 and port 80. And it will match on a full transaction, using both the uri from the request, and the stat_code from the response. As such, it will match only when Suricata got both request and response.

Transactional rules can use direction-ambiguous keywords, by specifying the direction.

```
alert http any any => 5.6.7.8 80 (msg:"matching json to server and
xml to client"; sid: 1; http.content_type: to_server; content:
"json"; http.content_type: to_client; content: "xml";)
```

Transactional rules have some limitations :

- They are only meant to work on transactions with first a request to the server, and then a response to the client, and not the other way around (not tested).
- They cannot have `fast_pattern` or `prefilter` the direction to client if they also have a streaming buffer on the direction to server, see example below.
- They will refuse to load if a single directional rule is enough.

This rule cannot have the `fast_pattern` to client, as `file.data` is a streaming buffer and will refuse to load.

```
alert http any any => any any (file.data: to_server; content: "123";  http.stat_code;
content: "500"; fast_pattern; sid: 1;)
```

If not explicit, a transactional rule will choose a fast_pattern to server by default.

Transactional rules behavior depends on how transactions are implemented per protocol, see Protocols for more details.

8.1.6. Rule options

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

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

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

Note

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

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

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

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

Some generic details about keywords follow.

8.1.6.1. Disabling Alerts

There is a way to disable alert generation for a rule using the keyword `noalert`. When this keyword is part of a rule, no alert is generated if the other portions of the rule match. That is, the other rule actions will *still be applied*. Using `noalert` can be helpful when a rule is collecting or setting state using *flowbits*, *datasets* or other state maintenance constructs of the rule language. See Thresholding Keywords for other ways to control alert frequency.

The following rules demonstrate `noalert` with a familiar pattern:

- The first rule marks state without generating an alert.
- The second rule generates an alert if the state is set and additional qualifications are met.

```
alert http any any -> $HOME_NET any (msg:"noalert example: set
state"; flow:established,to_server; xbits:set,SC.EXAMPLE,track
ip_dst, expire 10; noalert; http.method; content:"GET"; sid:1; )
```

```
alert http any any -> $HOME_NET any (msg:"noalert example: state
use"; flow:established,to_server; xbits:isset,SC.EXAMPLE,track
ip_dst; http.method; content:"POST"; sid: 2; )
```

In IPS mode, `noalert` is commonly used in when Suricata should *drop* network packets without generating alerts (example below). The following rule is a simplified example showing how `noalert` could be used with IPS deployments to drop inbound SSH requests.

```
drop tcp any any -> any 22 (msg:"Drop inbound SSH traffic";
noalert; sid: 3;)
```

8.1.6.2. Modifier Keywords

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

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

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

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

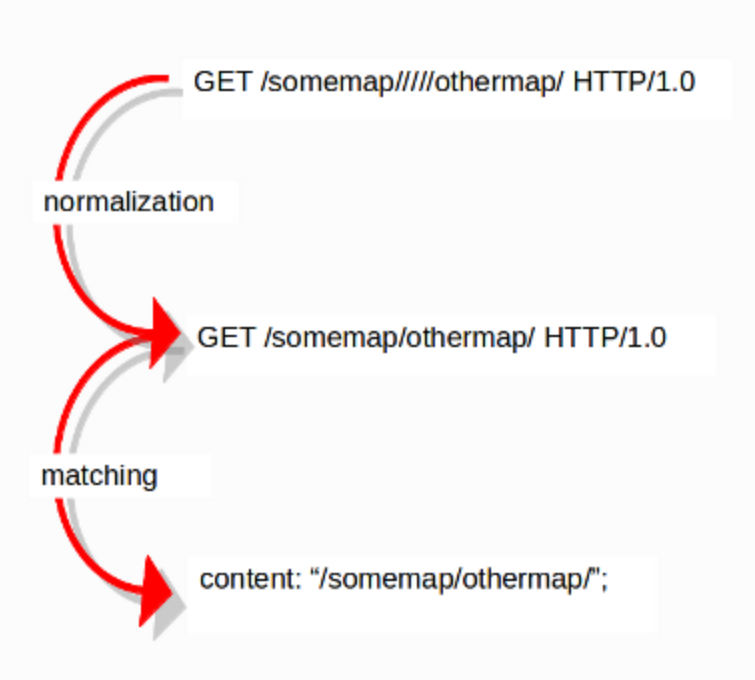
- The more recent type is called the **'sticky buffer'**. It places the buffer name first and all keywords following it apply to that buffer, for instance:

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

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

8.1.6.3. Normalized Buffers

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



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

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