CounterSnipe – Writing your own rules with Version 8.x.x

CounterSnipe provides vulnerability attack protection by keeping your systems automatically updated. Every hour the countersnipe IPS systems poll countersnipe risk servers to determine if there are new updates to the riskdata. Any new riskdata files are downloaded and 'loaded' into the local countersnipe database. This mechanism ensures that your systems are always protected from any attacks looking to exploit vulnerabilities within your network.

Countersnipe also provides policy signatures. But, in most cases you would wish to write your own policies that are specific to your organization. These could simply be who has what access to what systems or more detailed whereby you might want to monitor, log and protect against any unauthorized activities or unauthorized access to internal resources.

This document should provide you the background information and understanding about the risk data rule set used in countersnipe.

Here are some key points to note:

- It is not possible for you to manually edit any countersnipe rules that are uploaded from our servers. But it is possible for you to copy them into local group and then change them by selecting 'edit'.
- All access to countersnipe rules is via the Management Console.
- Please use the interface provide, by clicking on 'policies' and 'new' to compose your company specific rules.
- The rules language and options are Snort compatible and Suricata compatible. Suricata options are not necessarily compatible with snort IDS engine. CounterSnipe version 7.0.0 upwards uses Suricata and as such any suricata documentation as widely available will be equally useful in wiriting the rules.
- The standard order of rules is
    - Pass
    - drop
    - reject
    - alert

Let us start with an example of a rule;

ACTION tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"MALWARE Traffic Syndicate Add/Remove"; sid:7000009; rev:1; classtype:policy-violation; flow: to_server,established; uricontent:"/Support/AddRemove.aspx?id="; nocase;)

In the rule above all of the options not in yellow are chosen from selection menus. Most of the drop down options are self explanatory and documented in the main admin guide. The purpose of this document is primarily to cover the key options that you will require to complete depending upon what you are trying to achieve.

The following screen shot allows you to edit the rule. There are various selections that can be made;

Name: When an event occurs, it will be logged under this name. Therefore a carefully chosen and detailed name will help you easily identify what happened.

SID: Please do not change. This is system allocated.

Revision: If you have two versions of the same signature, specify the version number.

Classification: Choose a classification level for your signature. If you want to be alerted on it, then you may want to choose 'High' if Alert Handling is configured for alerting on 'High' If you just want to log the events, then you will need to choose a classification that is not configured to send out alerts.

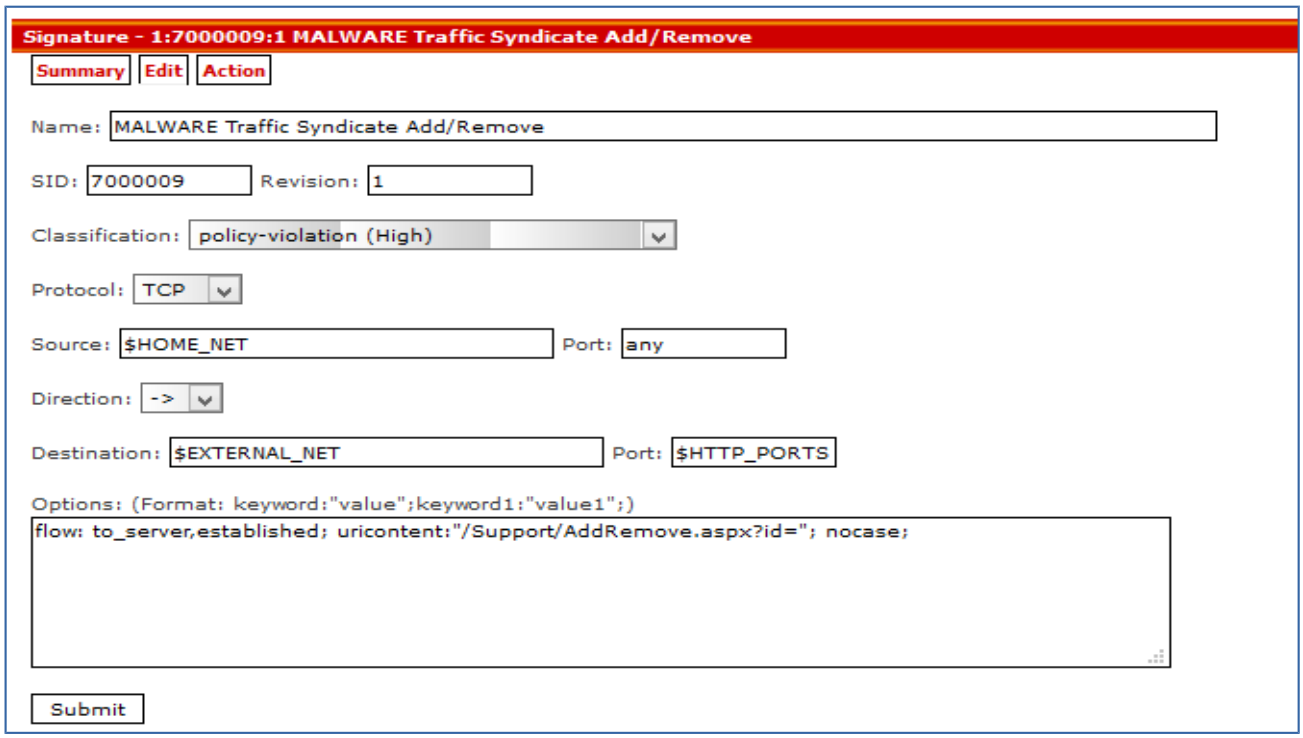
Protocol: Please choose the appropriate protocol.

Source/Destination: You can specify a pre defined Variable, a Range, a single IP or a all of those with a !(not) sign preceeding them. For example !192.168.0.24 will function for everything but 192.168.0.24.

Port: Specify a Variable, a single port or a range of ports. To specify a range, 1:444 all ports including the two values, 443: means all ports starting 443 and :443 means all ports upto 443. You can use ! For negating the same way as for Source/Dest.

Direction: The direction field is used to tell Suricata which IP address and port is the source and which pair is the destination. Specifying <> as the direction tells Suricata that you want the rule to apply bidirectionally. This is especially useful when using log rules, since you can log both sides of the TCP stream rather than just one direction.

Options: There are wide range of options available. We will cover the most common ones in this document. I training course is recommended for complete a-z of options.



Looking at our rule above, flow: to_server,established; uricontent:"/Support/AddRemove.aspx?id="; nocase; then remains our options part.

When a packet comes in, its source and destination IP addresses and ports are then compared to the rules in the ruleset. If any of them are applicable to the packet, then the options are compared to the packet. If all of these comparisons return a match, then the specified action is taken. The action can be individually set for each rule by selecting 'Action' from the menu and changing the action either globally or for an individual sensor.

Back to our rule then. This rule monitors for all network traffic originationg from within the Home_Net and going to External_Net, over tcp protocol and Http ports if that traffic is flowing to the server and the flow is established it will look in the URI field and look for the Add/Remove options. If it gets a match it will take an action in line with the Action setting.

Now that you know all of the fields, you will be able to write your own rules. At least some simple rules to perform regular tasks. All you will have to do is create new policy, then click to access it, click edit and select appropriate fields. Once selected use the keywords and values to fine tune the rule.

Here are some of the more useful keywords and their possible values;

content: allows you to search a packet for a sequence of characters or hexadecimal values. If you are searching for a string, you can just put it in quotes. In addition, if you want it to do a case-insensitive search, you can add nocase; to the end of all your options. However, if you are looking for a sequence of hexadecimal digits, you must enclose them in | characters.

Remeber you only add the options as all else is taken care of in the menu drop downs.

Example1: content: “Youtube”; nocase;

Example2: content:”|90|”;

This rule will trigger when it sees the digit 0x90. This digit is the hexadecimal equivalent of the NOP instruction on the x86 architecture and is often seen in exploit code since it can be used to make buffer overflow exploits easier to write.

The offset and depth options can be used in conjunction with the content option to limit the searched portion of the data payload to a specific range of bytes.

If you wanted to limit content matches for NOP instructions to between bytes 40 and 75 of the data portion of a packet, you could modify the previously shown rule to look like this:

Example: content:"|90|"; offset:40; depth:75;

You can also match against packets that do not contain the specified sequence by prefixing it with a !. In addition, many shell code payloads can be very large compared to the normal amount of data carried in a packet sent to a particular service. You can check the size of a packet's data payload by using the dsize option. This option takes a number as an argument. In addition, you can specify an upper bound by using the < operator, or you can choose a lower bound by using the > operator. Upper and lower bounds can be expressed with <>.

Example: content:"|90|"; offset:40; depth:75; dsize: >6000;

This modifies the previous rule to match only if the data payload's size is greater than 6000 bytes, in addition to the other options criteria.

flow:

The flow rule option is used in conjunction with TCP stream reassembly. It allows rules to only apply to certain directions of the traffic flow.

This allows rules to only apply to clients or servers. This allows packets related to $HOME_NET clients viewing web pages to be distinguished from servers running the $HOME_NET.

The established keyword will replace the flags: A+ used in many places to show established TCP connections.

Options

to_client

trigger on server responses from A to B

to_server

trigger on client requests from A to B

from_client

trigger on client requests from A to B

from_server

trigger on server responses from A to B

established

trigger only on established TCP connections

stateless

trigger regardless of the state of the stream processor (useful for packets that are designed to cause machines to crash)

no_stream

do not trigger on rebuilt stream packets (useful for dsize and stream4)

only_stream

only trigger on rebuilt stream packets

Format

```
flow:[to_client|to_server|from_client| \
  from_server|established|stateless|no_stream|only_stream]}
```

Examples:

```
alert tcp !$HOME_NET any -> $HOME_NET 21 (flow: from_client; \
      content: "CWD incoming"; nocase; \
      msg: "cd incoming detected"; )
```

```
alert tcp !$HOME_NET 0 -> $HOME_NET 0 \
    (msg: "Port 0 TCP traffic"; flow: stateless;)
```

flags: To check the TCP flags of a packet, Suricata provides the flags option. This option is especially useful for detecting port scans that employ various invalid flag combinations.

For example, this rule will detect when the SYN and FIN flags are set at the same time:

flags: SF,12;

Valid flags are S for SYN, F for FIN, R for RST, P for PSH, A for ACK, and U for URG. In addition, Snort lets you check the values of the two reserved flag bits. You can specify these by using either 1 or 2. You can also match packets that have no flags set by using 0. There are also several operators that the flags option will accept. You can prepend either a + , *, or ! to the flags, to match on all the flags plus any others, any of the flags, or only if none of the flags are set, respectively.

Tag:

The tag keyword allow rules to log more than just the single packet that triggered the rule. Once a rule is triggered, additional traffic involving the source host is ``tagged''. Tagged traffic is logged to allow analysis of response codes and post-attack traffic. See Figure 2.26 for usage examples.

Format

tag: <type>, <count>, <metric>, [direction]

type

session

log packets in the session that set off the rule

host

log packets from the host that caused the tag to activate (uses [direction] modifier)

count

Count is specified as a number of units. Units are specified in the <metric> field.

metric

packets

tag the host/session for <count> packets

seconds

tag the host/session for <count> seconds

```
alert tcp !$HOME_NET any -> $HOME_NET 143 (flags: A+; \
    content: "|e8 c0ff ffff|/bin/sh";  tag: host, 300, packets, src; \
    msg: "IMAP Buffer overflow, tagging!";)
```

alert tcp !$HOME_NET any -> $HOME_NET 23 (flags: S; \

tag: session, 10, seconds; msg: "incoming telnet session";

File Detection Related Keywords:

The filestore comes enabled as default. However no files will be logged and stored unless a rule is written and deployed to do so.

The simplest and very throughput expensive rules will be as below (only showing the options part)

flow:established,to_client;filestore;

or

filestore;

for file recording in any direction.

These rules will store all files transferred via http in a directory called filestore on the Countersnipe system. The directory can be compressed and backed up frequently by using the included script called backup-filestore. The task can also be automated for regular backups. CounterSnipe support@countersnipe.com will be able to help in the automation process.

Other file related words for narrowing the search and store;

filename:<string>;

Example:

filename:"secret";

fileext:<string>;

Example:

fileext:"jpg";

filemagic:<string>;

Example:

filemagic:"executable for MS Windows";

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:[!]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.

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

Additionally here is a list of other Keywords and their description. These are less commonly used in bespoke situations.

ttl

test the IP header's TTL field value

tos

test the IP header's TOS field value

id

test the IP header's fragment ID field for a specific value

ipoption

watch the IP option fields for specific codes

fragbits

test the fragmentation bits of the IP header

seq

test the TCP sequence number field for a specific value

ack

test the TCP acknowledgement field for a specific value

itype

test the ICMP type field against a specific value

icode

test the ICMP code field against a specific value

icmp_id

test the ICMP ECHO ID field against a specific value

icmp_seq

test the ICMP ECHO sequence number against a specific value

content-list

search for a set of patterns in the packet's payload

session

dumps the application layer information for a given session

rpc

watch RPC services for specific application/procedure calls

resp

active response (knock down connections, etc)

react

active response (block web sites)

tag

advanced logging actions for rules

ip_proto

IP header's protocol value

sameip

determines if source ip equals the destination ip

stateless

valid regardless of stream state

regex

wildcard pattern matching

byte_test

numerical evaluation

within

forcing relative pattern matching to be within a count

byte_test

numerical pattern testing

byte_jump

numerical pattern testing and offset adjustment

We hope that the information above will enable you to write some quick rules for daily tasks. Please contact support@countersnipe.com if you need help with any specific rules. We also recommend the comprehensive training course that we offer where you will be able to learn in details all of the options and practice your rule writing skills in the supervision of our instructors.