

AlienVault Unified Security Management™ Solution

Complete. Simple. Affordable

Building A Custom Plugin

**Table of Contents**

Building A Custom Plugin 1

1. Introduction 4

2. Reviewing the log data 4

3. Building the plugin 6

4. Creating plugin entries in the AlienVault database 9

5. Enabling plugin on USM 11

5.1. To enable the plugin on a sensor 11

APPENDIX A - Troubleshooting 14

A.1. Troubleshooting Plugin 14

# Introduction

This document will cover the rudimentary basics on creating a custom plugin. A plugin can be very basic or very complex dependant on the log data it will parse and normalize. Much of the work building a plugin is to use regular expressions, so it is important to get familiar with building regular expressions.

# Reviewing the log data

When building a plugin it is important to identify the types of data in the logs and unique number of events in the log. To do this you could use perl to parse the log to see the number of unique events. I am performing the examples on a Mac which already has perl . If you run Windows you can install Cygwin to be able to run perl scripts.

In my example I am piping the log to parse the log with perl using regular expressions, so I can see the number of unique events in the log.

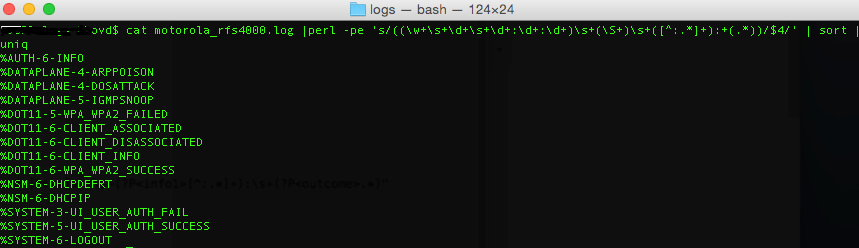
The first s/ is for the substitution so we can pull out specific information out of the log, by encapsulating sections of the regular expression in paranthesis. The first parathesis “()“ which is highlighted in yellow in the image below is encapsulating the whole regular expression which will return the full log message. The first parenthesis (yellow) is represented by $1 inbetween the two forward slash /$1/. The second set of parathesis which is in this case identifying the date and time, is highlighted in green, and the and fourth parenthesis is represented is highlighted in pink and is referred to as $4.



Using a site like <https://regex101.com/#python> is very helpful when building the regular expressions.

cat yourlog.log |perl –pe ‘s/(your regularexpression 1)(regex 2)(regex 3)/$1,$2,$3/ |sort|uniq

In this case using the regular expression looking at $4 we would only return the different types of events in the log, By adding | sort and | uniq we can sort the events and only return unique event types. So now we know how many different unique types of events there are in the log.



Now that we know the different types of events in the log we need to build regular expressions which extracts the data we need out of each of these event types. It could be that the logging format is very uniform and that one regular expression will cover multiple event types, but it could also be that each event type have different sub categories which may contain different values which would be valuable to normalize.

|  |  |
| --- | --- |
|  | When using regex101 to build your regular expressions make sure to select to use python as the flavor!  <https://regex101.com/#python> |

# Building the plugin

Custom plugins need to be assigned a unique plugin ID, the custom plugin ID needs to be in the 9000 to 10,000 range.

The plugins should contain information about the plugin version, supported product for the plugin, all should be commented by adding the # at the beginning of the line. It is important to populate the comment fields with the name and comments of the device. These will be picked up by the system each night so that the plugin can be applied under the asset in the USM web interface. If you want to make the plugin available right away you can run a command from USM which will parse the cfg headers and make it available from the asset plugin.

# Alienvault plugin

# Author: Alienvault Team at [devel@alienvault.com](mailto:devel@alienvault.com" \t "_blank)

# Plugin {{ plugin\_name }} id:{{ plugin\_id }} version: -

# Last modification: {{ LAST\_MODIFICATION\_DATE }}

#

# Plugin Selection Info:

# {{vendor}}:{{model}}:{{version}}:{{per\_asset}}

#

# END-HEADER

The command to run to parse the headers is:

|  |
| --- |
| /var/lib/dpkg/info/alienvault-cpe.postinst configure |

The comments required in the cfg file headers which the command will parse:

|  |  |
| --- | --- |
|  | # Alienvault plugin # Author: Alienvault Team at [devel@alienvault.com](mailto:devel@alienvault.com" \t "_blank) # Plugin {{ plugin\_name }} id:{{ plugin\_id }} version: - # Last modification: {{ LAST\_MODIFICATION\_DATE }} # # Plugin Selection Info: # {{vendor}}:{{model}}:{{version}}:{{per\_asset}} # # END-HEADER |

There are a few sections in a plugin [DEFAULT] will contain the unique plugin ID that you have decided to use for the custom plugin. [config] will contain all the information listed, but you only need to specify where you want the plugin to read the log files from (location=/var/log/yourlog.log). The [translation] is important and is linked to how many unique events you found in the log when you parsed it. Each unique event identified will need to have a plugin SID assigned to it, you can assign any value to the translation. Each event which requires its own regular expression will need its own heading such as [001 -AA Cisco FS300 Event] The regular expressions are processed based on the alpha numeric order of the header name. The more specific regular expressions should be ordered so that they are processed before more general regular expressions, if it is done in reverse the more general regular expression will be used instead, and it would not parse out all the information which would be valuable to normalize.

# Alienvault plugin

# Author: Customer at customer@email.com

# Plugin ciscofs300 id:9222 version: 0.0.1

# Last modification: 2015-05-05 11:45

#

# Accepted products:

# Cisco – FS300 X.X

# Description:

[DEFAULT]

plugin\_id=9222

[config]

type=detector

enable=yes

source=log

location=/var/log/ciscofs300.log

create\_file=false

process=

start=no

stop=no

startup=

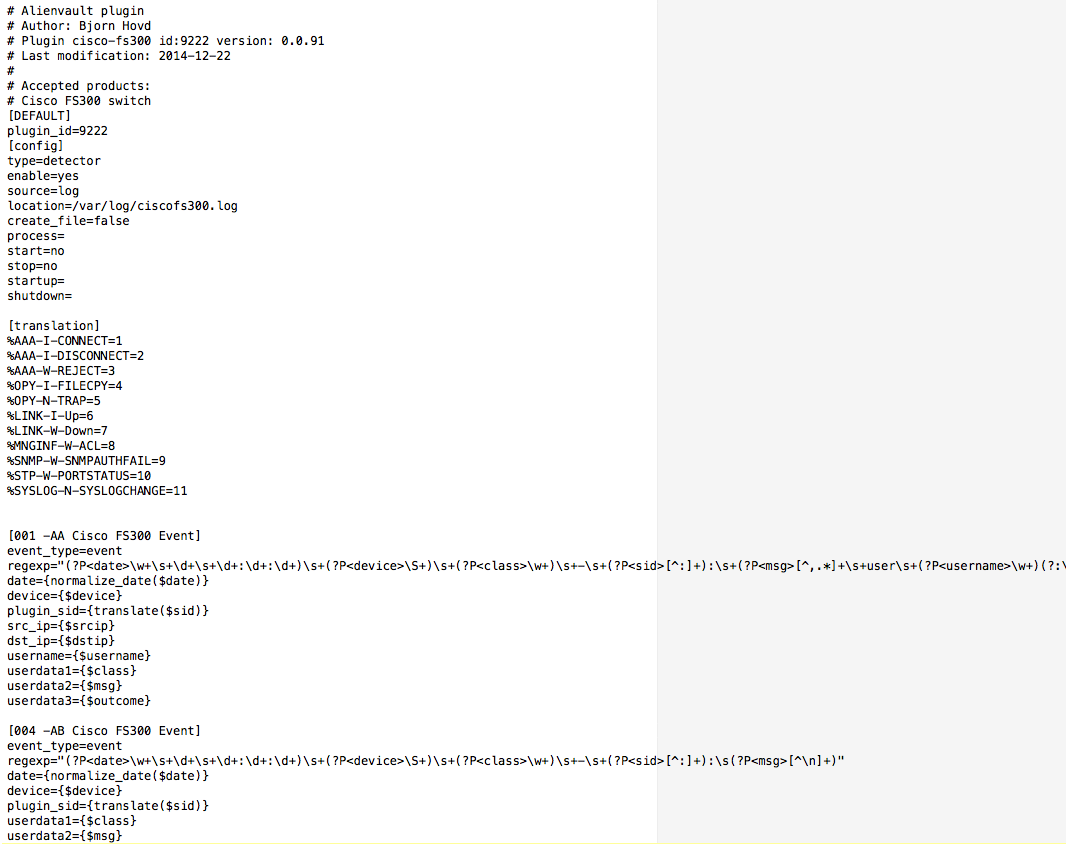
shutdown=

[translation]

%AAA-I-CONNECT=1

%AAA-I-DISCONNECT=2

%AAA-W-REJECT=3



Each event which is parsed should be tokenized. Each section which is to be normalized needs to be mapped to a field in the event table in the AlienVault database. For example +(?P<sid>[^:]+) is tokenized as sid. In the section below you can see that the sid is translated to the numeric value and is mapped to plugin\_sid in the event table in the AlienVault database.

regexp="(?P<date>\w+\s+\d+\s+\d+:\d+:\d+)\s+(?P<device>\S+)\s+(?P<class>\w+)\s+-\s+(?P<sid>[^:]+):\s(?P<msg>[^\n]+)"

date={normalize\_date($date)}

device={$device}

plugin\_sid={translate($sid)}

userdata1={$class}

userdata2={$msg}

|  |  |
| --- | --- |
|  | The following are some fields that can be mapped to in in the alienvault event table:  Interface  plugin\_sid  protocol  src\_ip  dst\_ip  src\_port  dst\_port  asset\_src  asset\_dst  filename  username  password  userdata1  userdata2 userdata3  userdata4  userdata5  userdata6  userdata7  userdata8  userdata9 |

# Creating plugin entries in the AlienVault database

The custom plugin you create also needs to be inserted into the database.

To speed up the process of creating the SQL file you can use this script with will populate the SQL file for you. You can get the script from:

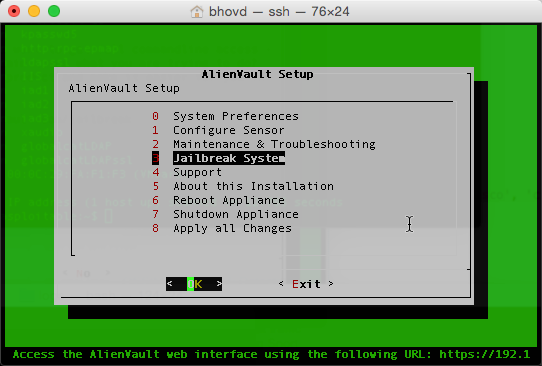
<https://github.com/hovdb/Alien-Plugins/blob/master/scripts/SQL_FILE_FOR_PLUGIN_CREATOR.vbs>

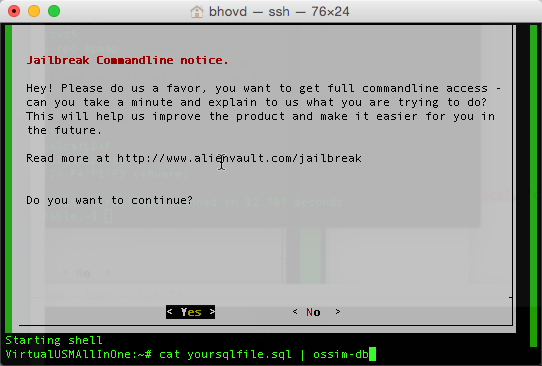
It is a vbscript which will prompt you for information and populate the SQL file.

Save the sql file and copy it over to the USM server. To insert the updates from the SQL file you created. Select option 3 Jailbreak System, and then type:

|  |
| --- |
| cat YOURSQLFILENAME.sql | ossim-db |

This will insert plugin information required in SQL.





|  |
| --- |
| -- cisco fs300 switch  -- plugin\_id: 9222  --dropping all references of plugin from plugin, plugin\_sid & software\_cpe  DELETE FROM plugin WHERE id = "9222";  DELETE FROM plugin\_sid where plugin\_id = "9222";  DELETE FROM software\_cpe where plugin = 'cisco-fs300:9222';  -- creating plugin in plugin table  INSERT IGNORE INTO plugin (id, type, name, description, product\_type, vendor) VALUES (9222, 1, 'cisco-fs300', 'Cisco Switch', '24', 'Cisco');  -- each event in plugin under translate needs to have its own insert  -- referenced by the ID assigned in translation of the plugin.  INSERT IGNORE INTO plugin\_sid (plugin\_id, sid, category\_id, class\_id, name, priority, reliability) VALUES (9222, 1, NULL, NULL, 'AAA-I-CONNECT', 2, 2);  INSERT IGNORE INTO plugin\_sid (plugin\_id, sid, category\_id, class\_id, name, priority, reliability) VALUES (9222, 2, NULL, NULL, 'AAA-I-DISCONNECT', 2, 2);  INSERT IGNORE INTO plugin\_sid (plugin\_id, sid, category\_id, class\_id, name, priority, reliability) VALUES (9222, 3, NULL, NULL, 'AAA-W-REJECT', 2, 2);  INSERT IGNORE INTO plugin\_sid (plugin\_id, sid, category\_id, class\_id, name, priority, reliability) VALUES (9222, 4, NULL, NULL, 'COPY-I-FILECPY', 2, 2);  -- insert to make plugin available to be assigned directly to asset  INSERT IGNORE INTO software\_cpe (cpe, name, version, line, vendor, plugin) VALUES ('cpe:/o:cisco:FS\_300\_switch:fs\_300', 'FS 300 Switch','fs\_300','Cisco FS 300 Switch','Cisco', 'cisco-fs300:9222'); |

# Enabling plugin on USM

Once the plugin is created copy the plugin over to /etc/ossim/agent/plugins on the USM server. Make sure to name the plugin the same as you referenced it in the SQL statement. For example if the name of the plugin is cisco-fs300.cfg use the same name in the sql statement(cisco-fs300). Once the plugin file is copied over. You can enable the plugin either at the asset level or at a sensor level.

## To enable the plugin on a sensor

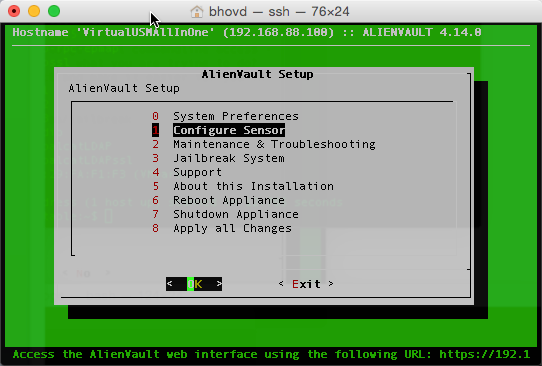
First you need to configure rsyslog to take all syslog coming from that IP and write it to the file path specified in the plugin. Jail Break USM and navigate to /etc/rsyslog.d/ and create a new file with a file name of the IPaddress of the server which is forwarding the logs to USM, such as 192.168.150.14.conf. The file content of the file should look like the following:

|  |
| --- |
| if $fromhost-ip == '192.168.150.14' then /var/log/ciscofs300.log  & ~ |

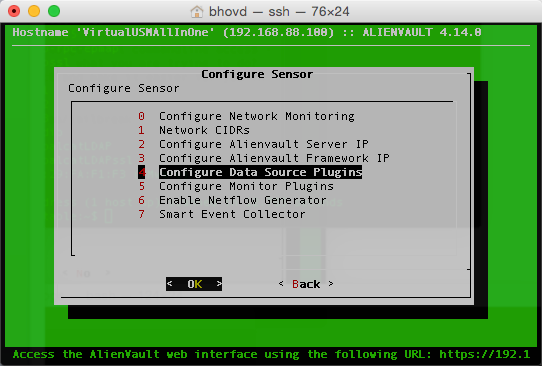
Once the file is saved under /etc/rsyslog.d/ restart the rsyslog service by typing:

|  |
| --- |
| service rsyslog restart |

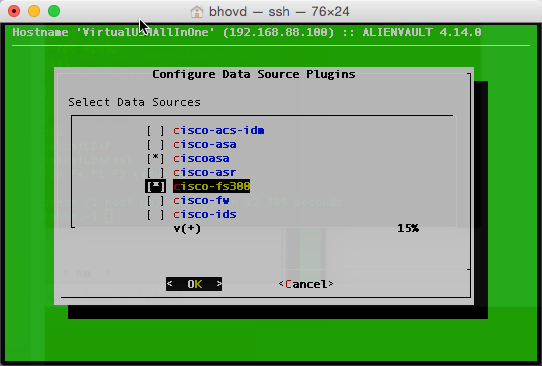
Exit back to the AlienVault Setup and select option 2 Configure Sensor.



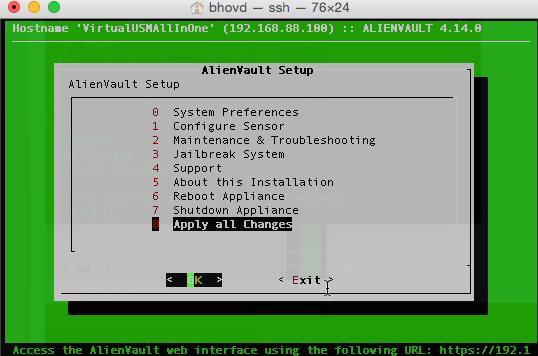
Then select option 4 Configure Data Source Plugins



Locate your plugin, enable it and click OK.



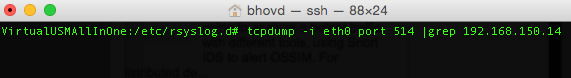
Navigate back to the initial AlienVault Setup window and select option 8 Apply all Changes.



Once the changes are applied the plugin should start processing and normalizing events.

1. Troubleshooting
   1. Troubleshooting Plugin

If have set up forwarding of logs, but you do not see the files being created under /var/log based on the rsyslog.d configuration file you specified you may want to see that the logs are actually coming from the IP specified in the configuration file. You can do that by performing a tcpdump



If you do not see any traffic then you can take off the grep and IP and see if you can identify the IP address that the logs may be coming from. You would then need to modify the conf file under /etc/rsyslog.d/ to match that IP, and after making the change the rsyslog service needs to be restarted.

If the logs are coming in, but are not processed, or normalized you can look at the logs and then look at the plugin for problems with the translation, or regular expression.

For troubleshooting check the log files at:

* /var/log/ossim/agent.log
* /var/log/ossim/server.log

|  |  |
| --- | --- |
|  | Each time you make a change to the plugin you need to jailbreak AlienVault USM and run alienvault-reconfig. |