Parsing Script User Guide

The following parsing scripts help you write rules for parsing logs. The scripts identify common errors in rules, such as invalid XML or data types, and provide the number of logs that each rule hits. The scripts require that you install the following:

- Python 3
- Syslog-ng

Manual bash script (Syslog)

This script parses logs using the rules written by the user and returns errors, if any.

1. First, the script validates whether the XML structure of your rules file is valid (the structure is the same as defined in the file 'patterndb.xsd'). If it is not valid, the script returns an error and exits.
2. The script checks the fields in the rules. If any fields do not match the FireEye Helix Taxonomy, the script prints them on the console. Sample error:
   Found field names that did not match with the taxonomy.
   {'detectedtime123', 'version123'}
3. The script checks for duplicate fields in each rule and displays any duplicates on the console. (Values are overwritten when duplicate field names are used.) Sample error:
   Duplicate field names found in rule-id: claroty_ctd-1913782642
4. The script prints on the console the total number of logs that were parsed correctly and the total number of logs that were not. Example:
   Logs matched: 1000
   Logs unmatched: 0
5. The script generates the following files:
   - An output file that contains the parsed information.
   - A matching file that contains all the logs that were parsed correctly.
   - An unmatching file that contains all the logs that were not parsed correctly.
   - A text file file named 'unidentified_fields.txt' that contains any field names that did not match the FireEye Helix Taxonomy,

Instructions for running the bash script:

1. Run the bash script 'new_manual_run.sh' with the following arguments in order:
   i. *1st argument*: The rules XML file. (Mandatory)
   ii. *2nd argument*: The logs text file. (Mandatory)
   iii. *3rd argument*: The text file where the parsed information will be written. (Mandatory)
   iv. *4th argument*: The file containing the logs that matched. (Optional)
   v. *5th argument*: The file containing the logs that did not match. (Optional)

2. If there is a program name in the rules file, edit the line that creates a variable called ‘res’ in the ‘new_manual_run.sh’ script and add ‘*-P program_name*’ after the keyword ‘match’.
3. Keep the files ‘check_field.py’, ‘patterndb.xsd’ and ‘taxonomy.json’ in the same directory as the bash script.

   For example:
   ./new_manual_run.sh rules.xml logs.txt output.txt matching_logs.txt unmatching_logs.txt

Datatype Checker script

This script verifies whether the parser used by a developer while writing a rule is in accordance with the datatype mentioned in the FireEye Helix Taxonomy. (Please check ‘taxonomy.json’ for the datatypes of fields.) The script displays errors and (optional) warnings.

- *Error* means the parser used in the rule is incorrect.
- *Warning* means the taxonomy for this field includes a data format. You should verify whether the incoming values in logs match the data format.

Sample Error: *ERROR: In rule-id: claroty_ctd-3454687452, the incorrect parser used for field: portid, @NUMBER used while the data type of this field in Taxonomy is: string.*

Sample Warning: *Warning: In rule-id: claroty_ctd-2084354322, @ESTRING used for field: detectedtime, but in taxonomy, data-type is: string and data-format is: date-time, please ensure output's value matches data-format.*

By default, warnings are not displayed. To display warnings, remove the ‘#’ character from line number 158 in the python script: #print_warnings().

Instructions for running the datatype checker script:

1. Run the script using the path to the XML file that contains your ruleset as the first argument. For example: *python3 datatype_checker.py rules.xml*
2. Keep the file ‘taxonomy.json’ in the same directory as the datatype checker script.
3. For help, run ‘*python3 script.py --h*’.

Field Count script

This script counts the number of times a field name is used in a particular ruleset. It is useful to give you an idea of what field names have been used when you need to choose a field name for a new incoming logline. The script displays the number of times each field name is used in the specified ruleset. Sample output:

{'uri': 15, 'tmp.min': 33, 'manufacturer': 32, 'srcmac': 25, 'srcipv6': 4, 'tmp.hour': 33, 'srcipv4': 23, 'detectedtime': 32, 'action': 31, 'tmp.day': 33, 'eventid': 33, 'category': 31, 'msg': 33, 'network': 32, 'status': 16, 'site': 32, 'dstzone': 31, 'product': 32, 'interface': 1, 'srchost': 25, 'hostname': 33, 'dstipv4': 19, 'severity': 32, 'rule': 32, 'srczone': 31,

'firstseen': 16, 'tmp.month': 33, 'tmp.year': 33, 'dsthost': 26, 'tmp.sec': 33, 'requestid': 31, 'timezone': 33, 'protocol': 22, 'domain': 15, 'portid': 13, 'dstipv6': 6, 'version': 32, 'signature': 32, 'dstmac': 15}

Instructions for running the field count script:

1. Run the script using the path to the XML file that contains your ruleset as its first argument. Example: *python3 fields_count.py rules.xml*
2. For help, run '*python3 script.py --h*'.

Parsing Summary script

This script displays each rule ID and the total number of logs hit by each rule. This script requires the output file that you generated with the manual bash script. Sample output: *Rule claroty_ctd-2192844616 hit 922 logs.*

Instructions for running the parsing summary script:

1. Run the script using the path to the XML file that contains your ruleset as its first argument and the output text file as your second argument. Example: *python3 parsing_summary.py rules.xml output.txt*
2. For help, run '*python3 script.py --h*'.