Log Parser

By James Fulford, Software Release Engineer, March-April 2018

# Purpose

When posting environments, sometimes the sheer quantity of errors in syslog and apache2/error.log can make cataloging different types of errors time consuming. For instance, due to the extensive tests on NOC/C2 performed during a live post, it can take several hours just to parse out these logs and figure out what errors are the same, what errors are different, and mapping these errors to bugs in our tracking software.

The goal of this script is:

* To reduce the information overload that comes with scraping logs
  + Groups errors by error type
  + Has options to reduce output detail
* To minimize the time it takes to assess the situation in the logs during a live post
  + Passes through the logs once
  + Has options to present only relevant errors (after timestamp, given environment, didn’t previously raise)
  + Displays server-related unparsed errors in separate section
* To accelerate the bug documentation process
  + Identifies unique error types
  + Associates errors that often occur together
  + Has options to filter out bugs irrelevant to code changes
* To improve the quality of bug documentation by picking up on information and differences manual log checking often misses.
  + Finds less-evident associations between error types
  + Has options to investigate rotated-out logs for full back-history

This script is not intended to find bug numbers in any tracking software.

# Usage

## System Requirements

This script requires python, and currently works on 2.7.x. No additional packages are required. (Note: a compatible version of python ships with Ubuntu installs, including 14 and 16.)

## Common Usage

Most of the time, the following usage is what one is interested in using:

sudo python parse\_log.py /var/log/syslog /var/log/apache2/error.log --environments westec

parse\_log.py: This script, after being checked out. Note that you are not required to be in the same directory of the script in order to run.

/var/log/syslog /var/log/apache2/error.log: Logs to check. Currently, logs with the name “syslog” or “error.log” have parsing support. Errors from all files are combined into one output.

westec: The name of the environment you would like to ‘grep’ for. Case insensitive, and optional.

sudo: syslog and apache2/error.log require root privileges in order to read the file on most servers.

## Useful Options

|  |  |
| --- | --- |
| -h, --help | Shows a help message. |
| -m, --must\_match | Requires errors to match given regex. Optional. May give many patterns. |
| -e, --environments | Requires errors to be from one of the environments given. Optional. May give many environments arguments. |
| -o, --output | Dumps json version of parsed error output. Path is relative to the user’s present working directory. Filtering options do not apply to this file. |
| -x, --exclude-unparsed | Does not print unparsed errors. |
| -q, -qq, -qqq | Reduces/quiets amount of parsed error output.  -q: hides raw error text  -qq: hides parsed details about each error  -qqq: hides associations information  -qqqq: hides all parsed error output |
| --min-date | Forces output to only include errors which raised on or after this timestamp. Use the format "%Y-%m-%d %H:%M:%S" ([strftime.org](http://strftime.org)) (be sure to quote this value). For example, --min-date "2018-05-01 17:01:23" restricts output to only include errors logged after May 1st, 2018, 5:01PM and 23 seconds. |
| -d, --debug | Outputs more information useful during development, such as stack traces. All parsing errors from within this script are always reported to the user, so no errors from the logs disappear silently during this process. |
| -b, --backlog | Sets maximum file rotation to go back to for each file. For example, if parsing both syslog and apache2/error.log, the usage of --backlog 3 will cause syslog, syslog.1, syslog.2.gz, syslog.3.gz, error.log, error.log.1, error.log.2.gz, and error.log.3.gz. Note that this may slow down the script, and may include errors logged during other releases. |
| -i, --ignore-persisting | Only works if --min-date option is set. All error types |

# Handling Output

The standard out output format is divided into 3 main sections:

## Internal Errors

### Parsing Errors

It is possible that new log entry formats will be introduced to these logs over time. If the entry is an error, yet cannot be parsed by this script, the internal python exception is printed along with the raw error text so you can investigate the error manually.

If debug mode is enabled, the stack trace for these errors will be sent in this section via standard error.

### Filesystem Errors

If a log does not exist, or the script does not have read permissions for a given file, a message is sent in this section via standard error.

## Unparsed Errors

If any log entries which are not associated with an environment are found (kernel errors in syslog, for example), they are printed out in the Unparsed Errors section. To mute this output, add the -x or the --exclude-unparsed options.

### Example output

================================================================================

Example unparsed error, perhaps from the kernel or from something non-environment specific

================================================================================

Since this line includes a keyword (“error” in this case), yet is not obviously connected to any cswapi, swf, or gdf environment, it is included in the unparsed errors section.

### Keywords and Severities

Supported keywords and associated severities are, in order of most severe to least:

|  |  |
| --- | --- |
| Keyword | Severity |
| Fatal | Fatal |
| Error | Minor |

If an error presents multiple keywords, the highest severity of those keywords is assigned.

## Parsed Errors

### Example output

FATAL

BardHelper.php - Call to undefined function getresponse() (554)

Apr 30 23:18:40 HSNOTX08-SP-SRV-01 php: SERVER: HSNOTX08-SP-SRV-01 Device ID: 152689 Node ID: 122648 FATAL PHP AT LINE 554 IN /var/www/cswapi\_pacnw/networking/helpers/BardHelper.php: Call to undefined function getresponse()

2018-04-30 23:18:40 2018-04-30 23:18:40

1 instance(s)

MINOR

rmx.php - Trying to get property of non-object (231)

Mar 1 03:26:00 LWCTOH02MO1-E-CQ-SIPO-01 php: MainHandler status: [03/01/18 03:26:00] PHP error 'E\_NOTICE': on line 231 Array#012(#012 [type] => 8#012 [message] => Trying to get property of non-object#012 [file] => /var/www/cswapi\_ohpa/networking/traps/rmx.php#012 [line] => 231#012)

Mar 1 03:26:00 LWCTOH02MO1-E-CQ-SIPO-01 php: MainHandler status: [03/01/18 03:26:00] PHP error 'E\_NOTICE': on line 233 Array#012(#012 [type] => 8#012 [message] => Trying to get property of non-object#012 [file] => /var/www/cswapi\_ohpa/networking/traps/rmx.php#012 [line] => 231#012)

2018-03-01 03:26:00 2018-03-01 03:25:00

2 instance(s)

Trap.php - Undefined offset: 1 (124)

Mar 2 02:29:08 HCHLILMTMO1-E-CQ-SIPO-01 php: MainHandler status: [03/02/18 02:29:08] PHP error 'E\_NOTICE': on line 124 Array#012(#012 [type] => 8#012 [message] => Undefined offset: 1#012 [file] => /srv/generic-device-framework/ilwic/Trap/Trap.php#012 [line] => 124#012)

2018-03-02 02:29:08 2018-03-02 02:29:08

1 instance(s)

Associated with:

GDF Trap Handler: Emerson, NCU - Couldn't find source and destination IP addresses (125)

### Explanations

Severity: Output is grouped in order of severity, with the most severe error group being at the top of the output. (Mute: -qqqq)

Error Type: A name for a given set of directly-related errors. This allows for errors which raise at different times to be grouped together. Note that numbers in parenthesis tend to be line numbers. (Mute: -qqqq)

Raw Error: The most recent instance of this error type in the logs. (Mute: -q)

Line Number: If the same error type occurs on multiple lines, then individual raw errors are printed, most recent of each line number displayed. (Mute: -qqqq)

Date Range: The timestamp of the most recent and the oldest instance of this error type observed in the logs parsed. (Mute: -qqq)

Instances: Number of lines in the logs corresponding to this error type. (Mute: -qqq)

Associations: Based on the information available, this error type appears to be correlated with the other error type(s) given here. Remember that *correlation does not imply causation*. Factors used for associating errors include the PID of the process (when available) and the exact timestamp. (Mute: -qq)

# Sample Usage

Check logs for mobilerf\_clone errors raised after 11 o’clock PM May 3rd, 2018 on mobilerf\_clone:

sudo python ~/parse\_log.py /var/log/syslog /var/log/apache2/error.log -e mobilerf\_clone --min-date "2018-05-03 23:00:00" -xqq

Continuously check logs every 60 seconds, same conditions as above example:

watch -n 60 -t "sudo python ~/parse\_log.py /var/log/syslog /var/log/apache2/error.log -e mobilerf\_clone --min-date \"2018-05-03 00:00:00\" -xq"

Check logs for *new* errors raised after 12:32 AM May 24th, 2018 on c2:

sudo python ~/james.py /var/log/syslog -e c2 --min-date "2018-05-24 00:32:00" -i --backlog 2 -x -qq

Only look at unparsed errors:

sudo python ~/parse\_log.py /var/log/syslog /var/log/apache2/error.log -qqq

Search for errors in 7 most recent syslogs:

sudo python ~/parse\_log.py /var/log/syslog -e mobilerf\_clone --backlog 7 -xq

Search for errors in 7 most recent apache logs that include the phrase ‘treeName’:

sudo python ~/parse\_log.py /var/log/apache2/error.log -e mobilerf\_clone --backlog 7 -m treeName -xq

# Process

1. Filter
   1. Iterate through all lines of every log
      1. Filter out lines which match an exclusion pattern (case sensitive):
         1. (/usr/bin/php( \*?) /var/www/cswapi\_(.\*?)/cron/build\_queued\_device\_error\_check.php 2&>1)
         2. snmptrapd[
         3. CRON[
         4. ...
      2. If a keyword is found in the line, it is a candidate.
         1. If the line is not associated with an environment, it is an unparsed error.
         2. If the line is associated with an environment, it is an error.
2. Parse
   1. Depending on the source file and format, parse each error for rich fields such as the file, line number, error message, etc.
   2. Assign each error an error type depending on the source file and format of each error.
      1. This will be used to group errors of the same error type together at the Group stage.
   3. Assign each error an invocation value depending on the source file and format of each error.
      1. This value may consider the PID and/or the timestamp of the error, depending on availability.
      2. This will be used to associate errors together at the Associate stage.
3. Group
   1. Collect all errors of the same error type together.
4. Prototype
   1. Transform each group of errors into an error-like object to represent the error type.
      1. This error-like object is referred to as a prototype.
      2. Note that no information about the individual errors is lost during this process.
5. Associate
   1. If two or more errors from different error types have the same invocation value, link the two error types.
6. Display
   1. Print the unparsed errors to standard out
   2. Print the parsed errors output to standard out
   3. Dump all error type prototypes into a json file.

# Parsers

When attempting to match an error format to a parser, the source file is consulted, and then each is checked by a regex in descending order as shown here. The first pattern to match the error is the parser that will be used.

Formatting is used to match extracted values to the source of the data, whenever the relationship may not be obvious.

The term “basename” is the name of the file given in the “file” value, including the extension. For example, the basename of “/var/www/cswapi\_westnyack/networking/**NetworkTreeManager.php**” is “**NetworkTreeManager.php**”.

## Apache2/error.log

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pattern | Example | Parser Name | Extracted Values | Error Type Strategy | Invocation Strategy |
| .\* | [Tue Mar 13 00:03:19.826893 2018] [:error] [pid *6584*] [client 10.213.115.104:52814] PHP Notice: Undefined variable: treeMode in /var/www/cswapi\_**westnyack**/networking/NetworkTreeManager.php on line 2405, referer: https://10.212.74.108/westnyack/main.swf/[[DYNAMIC]]/4 | parse\_apache\_error | raw, timestamp, **environment**, severity, file,  *pid*, message, line | {basename} - {message} ({line})  NetworkTreeManager.php - PHP Notice: Undefined variable: treeMode (2405) | {pid}: {timestamp}  6584: 2018-03-13 00:03:19 |

## Syslog

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pattern | Example | Parser Name | Extracted Values | Error Type Strategy | Invocation Strategy |
| Array#012\(#012(.\*?#012)\) | Mar 1 03:25:00 LWCTOH02MO1-E-CQ-SIPO-01 php: MainHandler status: [03/01/18 03:25:00] PHP error 'E\_NOTICE': on line 231 Array#012(#012 [type] => 8#012 [message] => Trying to get property of non-object#012 [file] => /var/www/cswapi\_**ohpa**/networking/traps/rmx.php#012 [line] => *231*#012) | syslog\_array\_parser | raw, timestamp, **environment**, severity, file, message, *line*, type | {basename} - {message} ({line})  rmx.php - Trying to get property of non-object (231) | {timestamp}  2018-03-01 03:25:00 |
| Uncaught exception: Generic Device Trap Handler \(Vendor: (.\*?), Model: (.\*?)\) Critical Error#012Message: (.\*?)#012#012Stack trace:#012#0 (.\*?)\((.\*?)\)" | Mar 2 02:29:08 HCHLILMTMO1-E-CQ-SIPO-01 php: MainHandler status: [03/02/18 02:29:08] Uncaught exception: Generic Device Trap Handler (Vendor: Emerson, Model: NCU) Critical Error#012Message: Couldn't find source and destination IP addresses#012#012Stack trace:#012#0 /srv/generic-device-framework/**ilwic**/Trap/TrapHandler.php(125): GenericDevices\Trap\TrapHandler->abort('Couldn't find s...')#012#1 /var/www/cswapi\_ilwic/networking/traps/MainHandler.php(1073): GenericDevices\Trap\TrapHandler->processTrap()#012#2 /var/www/cswapi\_ilwic/networking/traps/MainHandler.php(464): processUsingNewGenericTrapHandler()#012#3 /var/www/cswapi\_ilwic/networking/traps/MainHandler.php(293): process\_mib()#012#4 {main} | syslog\_gdf\_trap\_handler\_parser | raw, timestamp, **environment**, severity, file, message, *line*, vendor, model | GDF Trap Handler: {vendor}, {model} - {message} ({line})  GDF Trap Handler: Emerson, NCU - Couldn't find source and destination IP addresses (125) | {timestamp}  2018-03-02 02:29:08 |
| FATAL | Apr 30 23:18:40 HSNOTX08-SP-SRV-01 php: SERVER: HSNOTX08-SP-SRV-01 Device ID: 152689 Node ID: 122648 FATAL PHP AT LINE *554* IN /var/www/cswapi\_**pacnw**/networking/helpers/BardHelper.php: Call to undefined function getresponse() | syslog\_fatal\_parser | raw, timestamp, **environment**, severity, file, message, *line* | {basename} - {message} ({line})  BardHelper.php - Call to undefined function getresponse() (554) | {timestamp}  2018-04-30 23:18:40 |
| .\* | Apr 30 23:18:40 HSNOTX08-SP-SRV-01 php: <...> (a filename with **environment**)<...>(a keyword) | syslog\_default\_parser | raw, timestamp, **environment**, severity, file | {basename} – UNCATEGORIZED () | {timestamp}  2018-04-30 23:18:40 |

\*syslog\_array\_parser: Each key-value pair is included in extracted values. These values tend to show up, but may not always appear.

### Syslog: Special Case

Whenever the same message is logged to syslog twice in the same second, it shows up in a different format. Consider this example:

Apr 10 02:35:20 HCHLILMTMO1-E-CQ-SIPO-01 php: message repeated **2** times: [ MainHandler status: [04/10/18 02:35:20] PHP error 'E\_NOTICE': on line 491 Array#012(#012 [type] => 8#012 [message] => Undefined offset: 1#012 [file] => /var/www/cswapi\_shofar/networking/traps/adc\_flexwave.php#012 [line] => 491#012)]

This error is unpacked into **two** instances of the repeated message, each of which are parsed using syslog parsers (wasteful, but it works with the existing framework). The space is left on the front of the raw text for easter egg purposes, so a developer can trace this back to a repeated error based on the raw error text.