**Documentation of logParser.py**

Waleed Nasr

4/11/2022

TABLE OF CONTENTS

[TABLE OF CONTENTS 1](#_Toc100157623)

[EXECUTIVE SUMMARY 2](#_Toc100157624)

[BUSINESS SCENARIO 3](#_Toc100157625)

[PROCEDURES 4](#_Toc100157626)

[Function parseLog(list) 5](#_Toc100157627)

[Function jsonFormater(string) 5](#_Toc100157628)

[Function dataAnalysis(\*arguments) 6](#_Toc100157629)

[Functions output(), formattingHelper() and handleUserInput() 6](#_Toc100157630)

[RESULTS 7](#_Toc100157631)

[RECOMMENDATIONS 8](#_Toc100157632)

[REFRENCES 9](#_Toc100157633)

[APPENDIX A: CODE OUTPUT 10](#_Toc100157634)

[Sample output of cleanLog.log: 10](#_Toc100157635)

[Sample output of analysedlog.json 10](#_Toc100157636)

[Beatified output of analysedlog.json 10](#_Toc100157637)

[Sample output of results.txt 10](#_Toc100157638)

EXECUTIVE SUMMARY

An intern developer testing specialist has been asked to write a program that can both parse and combine information in the log file. The solution is explored to be able to manage this data as well as any other discrepancies that may exist in this log file, or future log files that we use your program to parse. The main sections of this report include the procedures, which explore the process, and recommendations sections.

# BUSINESS SCENARIO

Company XYZ wants to develop a script that can parse through logs to find any irregularities such as DOS attacks. The script should be capable/adhere of doing the following:

* Handle messy, raw log files
* Ability to easily reused for other types of logs
* Outputs data for future use
* Analyze DOS attacks
* Must be written in any modern language (possibilities: Python, C, C#)

The required tools for the project are Visual Studio Code, GitHub, Regex. Please check appendix A for out put file samples. The following noted where taken during the meeting:

The log file consists of individual messages that are either being sent to the target or being received from the target. It is a requirement that the target responds within a certain timeframe to pass our tests. In other words, if an instrumentation attempt isn’t immediately followed by a response, a Denial of Service (DoS) failure is present. The way the DoS time is calculated is by taking the difference between the time that the original request was sent and when the target responded. Some examples from the log are shown below for verification.

Example:

Test case #5 – DoS time is 0:00:20.414

Test case #415 – DoS time is 0:00:20.405

Test case #1395 – DoS time is 0:00:06:070

# PROCEDURES

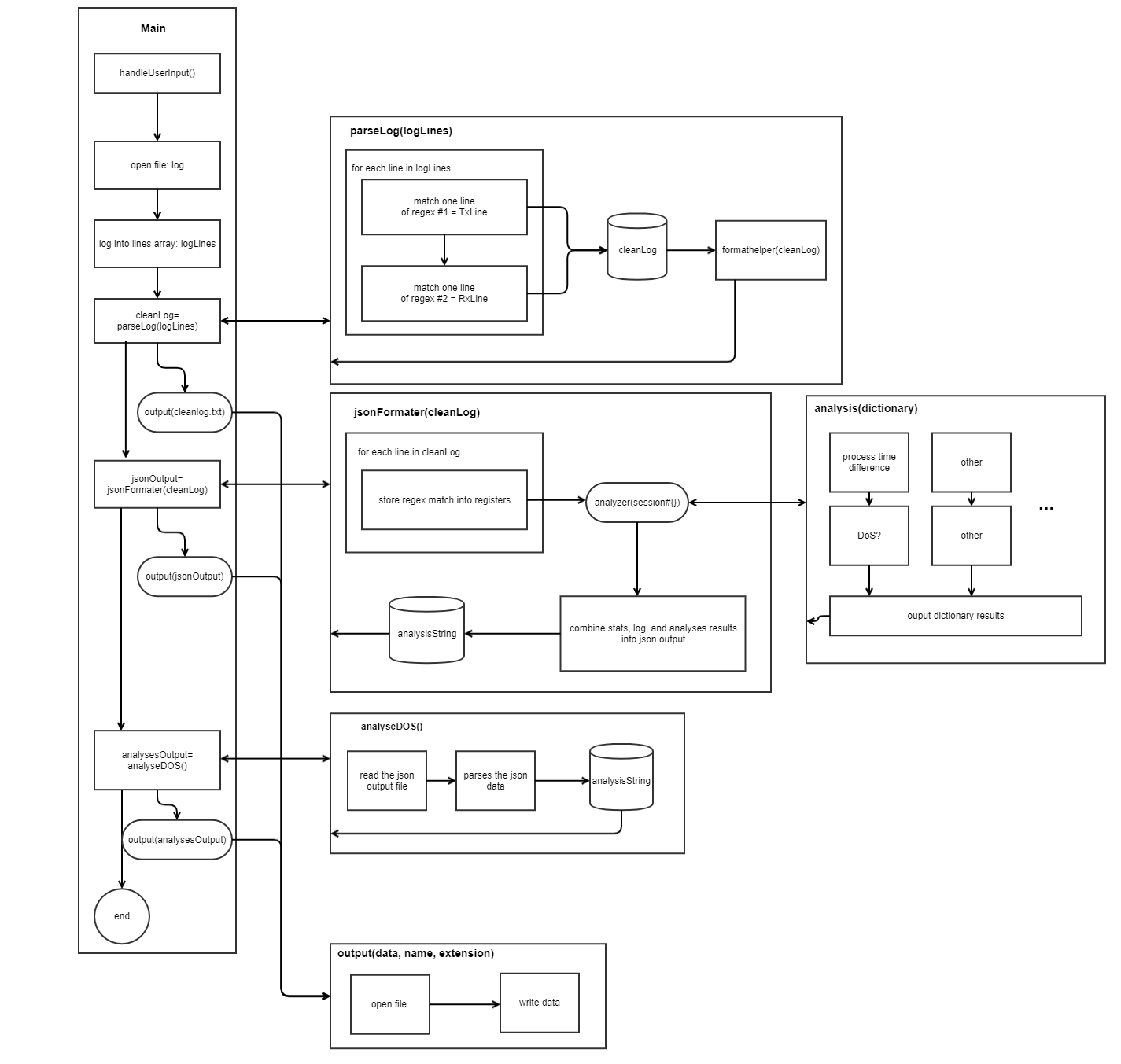
 This section will cover in detail how logParser.py operates. Each section will discover a function in the script. Look at figure 1 for the general layout of logParser.py.

Figure 1. logParser.py

The program is divided into six main functions. The function main() calls each function in the order that is appropriate. Refer to table 1 for a brief description of each function.

Table 1. Function descriptions

|  |  |
| --- | --- |
| Function | Description |
| parseLog | Will parse the raw log file, clean, and return a string with the help from *formatHelper()* |
| jsonFormater | Takes in a string, fetches data using *dataAnalysis(),* and then formatit to .json |
| dataAnalysis | Is responsible for dealing any kind of calculations, returns a dictionary |
| output | Handles all file writes. |
| formattingHelper | Reformates a list into a string |
| handleUserInput | Handles users input, including file selection. |

## Function parseLog(list)

This function takes in the output of the log file as a list. To filter out all of the unnecessary data, Regex will be used. For each line in the file, the function will alternate between the two regex matches (care for spaces):

regexMatchOne = r'^\d+-\d+-\d+ \d+:\d+:\d+\.\d+\s+\D+-\D+.+[0-9].+Tx.+1{6}'

regexMatchTwo = r'^\d+-\d+-\d+ \d+:\d+:\d+\.\d+\s+\D+-\D+.+[0-9].+Rx.+9{6}'

The function will only alternate if there is a successful match. If there is a successful match, the value will be stored in *cleanList.* To help with other Regex matches, the list is turned each stream into one line by using *formattingHelper*. Visit <https://regex101.com/> to have a visual representation of the match.

## Function jsonFormater(string)

This function takes the string input, feeds the filtered data into *dataAnalysis()* and then parses the data into a .json format output. To achieve this, Regex’s register functionality is used. Three capture groups are created using this Regex string:

(\d+-\d+-\d+ \d+:\d+:\d+\.\d+).+ ([0-9]+).+Tx.+1{6} (\d+-\d+-\d+ \d+:\d+:\d+\.\d+).+ [0-9]+.+Rx.+9{6}

The function then loops through each of the results, sends the required data to the function dataAnalysis() and then processes the results into a json formatted string. If future, changes is required, this is the place to change.

## Function dataAnalysis(\*arguments)

This function handles all the calculations, comparisons, or any type of data processing. The use of \*arguments allow for a flexible structure. Since the output of the function can be dynamic, the output will be a list.

## Functions output(), formattingHelper() and handleUserInput()

These functions are basic string and file handling. The main function of *formattingHelper()* is to collapse a single stream into one. The function handleUserInput() is mainly for grabbing the project name, the input file path and then append the project name to each file output type.

# RESULTS

Running the script will produce three files: cleanLog.log, analysedLog.json, results.txt. The DOS attacks that were observed in the sample file provided with a threshold of 100ms are provided in appendix A. The resultant code is modular enough for other types of uses. For example, if it is required to filter out more data the function *parseLog()* can be modified to do so. However, if the data needs more analyses the function *dataAnalysis()* can be adjusted to do so. However, there are further enchantments that can be made.

# RECOMMENDATIONS

The script is written in mind that the .json output will be processed somewhere else. The function *analyseDOS()* should be removed to create a very specific tasked script. Moreover, the function *jsonFormater()* should have a more dynamic method of writing data into the .json string. For example, rather than having a static string with the values pre-defined:

analysisString +="""{"stream": "%s", "details":[{"start time": "%s" ,"end time" : "%s" ,"dos time" : "%ss", "dos attack" : "%s"}]},"""

Something more dynamic like creating a list, looping through all values, and then constructing a string out of it might be more expandable. However, this is a decision for the manager to make. Finally, the script could be drastically improved by writing it in C, since python runs on C.

# REFRENCES

Dib, F. (n.d.). *Build, test, and debug regex*. regex101. Retrieved April 6, 2022, from https://regex101.com/

Microsoft. (2021, November 3). *JSON editing in Visual studio code*. RSS. Retrieved April 6, 2022, from https://code.visualstudio.com/docs/languages/json

Microsoft. (n.d.). *Visual Studio Code*. computer software. Retrieved from https://code.visualstudio.com/.

*Python regex capturing groups - pynative*. (n.d.). Retrieved April 6, 2022, from https://pynative.com/python-regex-capturing-groups/

*Regular expressions (regex)*. Regular Expression (Regex) Tutorial. (n.d.). Retrieved April 6, 2022, from https://www3.ntu.edu.sg/home/ehchua/programming/howto/Regexe.html

## APPENDIX A: CODE OUTPUT

## Sample output of cleanLog.log:

2021-02-09 13:10:55.876 CAN-FD 0 Tx 11111111 2021-02-09 13:10:55.878 CAN-FD 0 Rx 99999999

2021-02-09 13:10:56.012 CAN-FD 0 Tx 11111111 2021-02-09 13:10:56.018 CAN-FD 1 Rx 99999999

2021-02-09 13:10:56.121 CAN-FD 2 Tx 11111111 2021-02-09 13:10:56.127 CAN-FD 2 Rx 99999999

2021-02-09 13:10:56.233 CAN-FD 3 Tx 11111111 2021-02-09 13:10:56.237 CAN-FD 3 Rx 99999999

2021-02-09 13:10:56.341 CAN-FD 4 Tx 11111111 2021-02-09 13:10:56.348 CAN-FD 4 Rx 99999999

2021-02-09 13:10:56.453 CAN-FD 5 Tx 11111111 2021-02-09 13:11:16.637 CAN-FD 5 Rx 99999999

2021-02-09 13:11:16.851 CAN-FD 6 Tx 11111111 2021-02-09 13:11:16.857 CAN-FD 6 Rx 99999999

2021-02-09 13:11:16.861 CAN-FD 7 Tx 11111111 2021-02-09 13:11:16.867 CAN-FD 7 Rx 99999999

2021-02-09 13:11:16.912 CAN-FD 8 Tx 11111111 2021-02-09 13:11:16.917 CAN-FD 8 Rx 99999999

## Sample output of analysedlog.json

{"DOS\_THRESHOLD" : 100, "logs":[{"stream": "0", "details":[{"start time": "2021-02-09 13:10:55.876" ,"end time" : "2021-02-09 13:10:55.878" ,"dos time" : "0.00s", "dos attack" : "False"}]},{"stream": "0", "details":[{"start time": "2021-02-09 13:10:56.012" ,"end time" : "2021-02-09 13:10:56.018" ,"dos time" : "0.01s", "dos attack" : "False"}]}

## Beautify output of analysedlog.json

{

"DOS\_THRESHOLD": 100,

"logs": [

{

"stream": "0",

"details": [

{

"start time": "2021-02-09 13:10:55.876",

"end time": "2021-02-09 13:10:55.878",

"dos time": "0.00s",

"dos attack": "False"

}

]

},…

## Sample output of results.txt

DOS\_THRESHOLD: 100

DOS alert triggered -> Stream 5: 20.18s

DOS alert triggered -> Stream 19: 3.24s

DOS alert triggered -> Stream 29: 3.65s

DOS alert triggered -> Stream 204: 4.71s

DOS alert triggered -> Stream 415: 20.17s

DOS alert triggered -> Stream 465: 3.25s

DOS alert triggered -> Stream 784: 3.14s

DOS alert triggered -> Stream 1283: 3.25s

DOS alert triggered -> Stream 1395: 6.08s

DOS alert triggered -> Stream 1414: 3.86s