**INTRODUCTION**

The objective of this project is to ease the analysis of the log files and to infer as much as information from the log files. A user may input a web server log file to the program and wait for it to run the analysis of the data in the log file. After the analysis it returns visual data representing the data like IP address and the frequency of visit from the IP address. User-Agents used to visit the website and the method of requests made by the clients and frequency of the request methods. The output is not shown as traditional tables, but are visualized using graphs which makes it easier to consume the given information. The interface is a GUI in a windows based application which is a platform-independent because a platform-independent programming language is used to create the application. The application serves to be robust, which means that it could handle huge log files. The final product will have no more features that are specified in this document. This document describes the functional and non-functional requirements of the “Web Server Log Analysis System”. This application “Web Server Log Analysis System” is developed to help Security administrators, developers and Incident responders to remove their burden of looking up the logs manually.

**OVERALL DESCRIPTION**

The user launches the application, then by clicking on the Choose File button he could browse through the file system to find the log file that needs to be analyzed, then he clicks on the Generate Graph button to start the analysis. Then the application starts reading the file line by line and perform analysis on them. After it has done with the analysis, the graphs are generated based on the analysis performed and displayed to the user. From the displayed graphs, the users may easily infer information from the log file. The speed of the program depends on the size of the log file, the speed goes inversely proportional to the size of the log file.

**SPECIFIC REQUIREMENTS**

**3.1 Functional Requirements**

Easy to use – Provides a user-friendly GUI with very minimal input.

Analysis – Collects all data from the log file to provide statistics.

Output – Provide visual outputs in form of graphs, for efficient inference.

Looks – The application is visually appealing.

No Restrictions – One with access to log files can use this application.

**3.2 Non-Functional Requirements**

Performance – It generates statistics faster depending upon the file size.

Usability – People with no prior experience with log files can use the app.

Scalability – Process multiple log files in a single run of the application.

**3.3 External Interface Requirements**

**3.3.1 User Interfaces**

The application is user-friendly menu based interface. It has 2 simple screens as follows.

1. The Initial Screen

This is the screen where the log file that is to be analyzed is selected. There is a button provided to browse the filesystem to select the log file. Another button to generate the output when clicked.

2. The Output Screen

This screen shows the output of the analysis, this is filled with the graphs from the analysis. We may use the drop-down available in this screen to modify the output and acquire what is needed.

**3.3.2 Hardware Interfaces**

Screen resolution of at least 800×600 required for proper and complete rendering of the graphs. Higher resolutions would be better.

Standalone systems, no network connection required.

**3.3.3 Software Interfaces**

Any kind of Operating System(Linux, Unix, Windows, Mac OS)**.**

Python3.8 Installed.

**3.4 Performance Requirements**

The application should be able to handle large log files within seconds.

Graph generation should be processed quickly.

The application must support multiple files in a single run.

The application is platform independent and consumes less space.

**3.5 Design Constraints**

Simple and easy to use with user-friendly experience.

The application is visually appealing.

The graphs generated allow us to infer information with ease.

The data provided as output is accurate.

The application runs on all standalone systems with python.

**3.6 Software System Attributes**

The application can handle multiple files in the single run.

The system is robust to analyze huge log files.

The application is portable and can run on any standalone system.

The application is easy to use and can be used without any knowledge of the system.

**UML ANALYSIS MODEL**

**4.1 USE CASE DIAGRAM**

**<YET TO COME>**

**4.2 CLASS DIAGRAM**

**<YET TO COME>**

**4.3 SEQUENCE DIAGRAM**

**<YET TO COME>**

**4.4 STATE-CHART DIAGRAM**

**<YET TO COME>**

**4.5 ACTIVITY DIAGRAM**

**<YET TO COME>**

**PROJECT DESIGN**

The objective of this project is to develop an application that analyzes the log file from Web servers. This application could analyze logs from the Apache Web server.

**The Web Server Log**

The log file generated by default installation of Apache is as follows,

78.172.44.50 - - [19/Jan/2015:12:08:08 +0000] "GET /run HTTP/1.1" 200 83 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.10; rv:35.0) Gecko/20100101 Firefox/35.0"

Breaking down it into parts,

**“*78.172.44.50*”** – It is the IP address that is used to access the web server.

**“*-*”** - Identity check when authentication is enabled on the server.

**“-”** - The username used to access the authenticated content on the server.

**“*[19/Jan/2015:12:08:08 +0000]*”** – Timestamp, the time which the request made.

**“*GET / HTTP/1.1*”** – The request method used to access the resource.

**“*200*”** – The status cide returned by the server while accessing a resource.

**“*83*”** – The size of the response returned to the user by the server.

**“*-*”** - The Referer, the site which referred the user to our site.

***“Mozilla/5.0 (Macintosh; Intel Mac OS X 10.10; rv:35.0) Gecko/20100101 Firefox/35.0” –*** The User-Agent used by the user to access the server.

Our project reads a log file with lot of lines similar to this. Then save all the data in variables and generate graphs from all the data obtained from the file.

**The File Selection Phase**

First the program is launched, it opens the user-friendly GUI of the application. In the GUI there is a Choose File button to select the log file for analysis. After the button is pressed, the filesystem browser opens allowing us to select the log file to proceed for the analysis. Then the path is stored in a variable and the analysis on the file starts leading us to the next phase.

**The Analysis Phase**

This phase of the application follows up after selecting the file to analyze and proceeds to the analysis. The file is read line by line to avoid memory overhead. First a line is read and is split by the spaces. Then IP address(String), Timestamp (dictionar -y), Request method(dictionary), Status code(dictionary), the Accessed file (dictionar -y) and User-Agent(list) are obtained and stored in seperate variables after the split. First the IP address is appended to a list and a Data Object is created which stores all the above said variables as class variables. Then this data object is appended to another seperate list. While reading the next line, first the IP address list is checked to see if the IP is present, if it is present, the data according to the object is updated(the main reason why the variables are initialized as dictionaries). If it is not present, then the process of initialization occurs. The reason to initialize the User-Agent as a list is that one user may use the same browser, when he uses a large number of User-Agents within a time, it is an alarm of a DoS attack, also may serve as a work for the development team to tailor pages for that specific UA. This process is repeated until the whole file is read. Then the application moves on to the next phase.

**The Visualization Phase**

The list of data objects is iterated through one-by-one and for a given IP address, the total number of requests made is found out by adding all the values in the Requests dictionary. The variables for which the graphs are requested in the first phase is stored in variables and using those variables the graphs are generated. When

the Generate Graph button is pressed, the graph gets displayed to the screen.

The final thing is left to the user, he may infer the necessary information from the graphs. It may either be used for Incident Response, SEO or for further development of the deployed application depending upon the type of user running the application.

**CODE**

**1. The GUI Module**

#!/usr/bin/env python3

"""

FrontEnd Module

"""

import tkinter as tk

class GuiDesign:

def \_\_init\_\_(self, logObj, graphObj):

self.log = logObj

self.op = graphObj

def launchApp(self):

window = tk.Tk() # Starting tkinter window

window.title('Welcome!') # Setting window title

window.configure(bg='#293d3d')

label\_head = tk.Label(window, text='LogaLyzer', width=15, underline=4, padx=8, pady=8, borderwidth=10, highlightthickness=3, anchor='center', relief='ridge', fg='#003d66', bg='#80ccff', font=('Russo One', 20)).grid(row=0,column=0, padx=20, pady=25, columnspan=3) # Displaying heading label

label\_getfile = tk.Label(window, text='Kindly choose the log file:', font=('verdana', 9, 'bold'),fg='#ffffff', bg='#293d3d', anchor='w').grid(row=1,column=0, padx=15)

fileNameSetButton = tk.Button(window, text='Choose File', font=("Open Sans", 9, "bold"), borderwidth=5, bg='#b3e0ff', command=lambda: self.log.open\_file(), cursor='hand2', padx=5, pady=5, anchor='w').grid(row=1, column=1, padx=5, pady=0) # Setting the filename to variable

label\_getfile = tk.Label(window, text='Click for the results:', font=('verdana', 9, 'bold'),fg='#ffffff', bg='#293d3d', anchor='w').grid(row=2,column=0, padx=15)

graphGenButton = tk.Button(window, text='Generate Graph', font=("Open Sans", 9, "bold"), borderwidth=5, bg='#b3e0ff', command=self.op.gen\_graph, cursor='rtl\_logo', padx=5, pady=5).grid(row=2, column=1, padx=20, pady=20)

self.log.objArray.clear()

self.log.ipList.clear()

window.mainloop()

**2. The Analysis Module**

#!/usr/bin/env python3

"""

Analysis Module

"""

from tkinter import filedialog

class Data:

def \_\_init\_\_(self, ip):

self.ip = ip

self.req = {}

self.res = {}

self.UA = []

self.file = {}

self.tms = {}

def \_\_repr\_\_(self):

return self.ip

class LogLyzer:

objArray = []

ipList = []

f = None

ipReq = {}

def open\_file(self):

f = filedialog.askopenfile(mode='r')

self.objArray.clear()

self.ipList.clear()

self.readFile(f)

for obj in self.objArray:

print(obj.ip)

print(obj.req)

self.ipReq[obj.ip] = sum(list(obj.req.values()))

print(obj.res)

print(obj.UA)

print(obj.file)

print(obj.tms)

print("\n")

def readFile(self, f):

while True:

line = f.readline().replace("\"","").strip()

if(not line):

break

else:

temp = line.split()

if( temp[0] not in self.ipList ):

self.ipList.append(temp[0])

obj = Data(temp[0])

obj.req[temp[5]] = obj.req.get(temp[5],0) + 1

obj.res[temp[8]] = obj.res.get(temp[8],0) + 1

obj.file[temp[6]] = obj.file.get(temp[6],0) + 1

obj.UA.append(' '.join(temp[11:]))

obj.tms[temp[3].replace("[","")] = obj.tms.get(temp[3].replace("[",""),0) + 1

self.objArray.append(obj)

else:

index = self.ipList.index(temp[0])

self.objArray[index].req[temp[5]] = self.objArray[index].req.get(temp[5], 0) + 1

self.objArray[index].res[temp[8]] = self.objArray[index].res.get(temp[8], 0) + 1

self.objArray[index].file[temp[6]] = self.objArray[index].file.get(temp[6], 0) + 1

if( ' '.join(temp[11:]) not in self.objArray[index].UA ):

self.objArray[index].UA.append(' '.join(temp[11:]))

self.objArray[index].tms[temp[3].replace("[","")] = self.objArray[index].tms.get(temp[3].replace("[",""), 0) + 1

**3. The Graph Generator Module**

#!/usr/bin/env python3

"""

Output Module

"""

import matplotlib.pyplot as plt

class GraphGenerator:

def \_\_init\_\_(self, logObj):

self.log = logObj

def gen\_graph(self):

fig,ax = plt.subplots()

ax.bar(self.log.ipReq.keys(), self.log.ipReq.values())

fig.show()

**4. The Driver Module**

#!/usr/bin/python3

"""

Main Function #Driving Code

"""

import logaLyzer as ll

import graphGenerator as gg

import frontEndGUI as fg

log = ll.LogLyzer()

graph = gg.GraphGenerator(log)

gui = fg.GuiDesign(log, graph)

gui.launchApp()

**OUTPUT**

**<YET TO COME>**

**CONCLUSION**

Log analysis is an important task for System administrators, Security administrators and even for developers at times. Our project “Web Server Log Analysis System” makes the tasks of these administrators simpler by just getting in the log file as input and visualizing things present in the log file. This application would reduce the burden of those administrators task of manually going through the log files. All these presented using a simple to use and user-friendly GUI.

**REFERENCES**

1. Satya Prakash Singh, Meenu (2017), Analysis of Web Site Using Web Log Expert Tool Based on Web Data Mining, International Conference of Innovations in Information Embedded and Communication Systems(ICIIECS).

2. Dilip Singh Sisodia, Shrish Verma (2012), Web Usage Pattern Analysis Through Weblogs: A Review, Ninth International Joint Conference on Computer Science and Software Engineering(JCSSE)

3. AscToHTM(2005), A Web Server Log File Explained, <https://www.jafsoft.com/sea> [r](https://www.jafsoft.com/sear)[chengine](https://www.jafsoft.com/searchengine)[s/l](https://www.jafsoft.com/searchengines/l)[og\_sample.html](https://www.jafsoft.com/searchengines/log_sample.html), 24 Oct 2020.

4. Nihuo Software Inc, Log File Sample Explained, <https://www.loganalyzer.net/log-analysis-tutorial/log-file-sample-explain.html>, 24 Oct 2020.

Introduction

Overall Description

Specific requirements

Functional Requirements

non-functional requirement

external interface requirements

performance requirements

design constraints

software system attributes

UML Analysis model

Use case diagram

class diagram

sequence diagram

state chart diagram

activity diagram

Project design

Code

output

Conclusion

References