**REPORT FOR ASSIGNMENT 1**

**Log File Analysis and Threat Detection**

By:

Anastasia Atole (c00312185) **GITHUB**: https://github.com/stasiax0x0

Peace Ehirim (c00308865) **GITHUB**: <https://github.com/chickenfilletroll>

This program uses Python to analyse a log file and detect brute force attacks.

According to[*https://owasp.org/*](https://owasp.org/), brute force attacks are when an attacker attempts to discover a password by systematically trying every single combination of letters, numbers and symbols until they discover the right password.

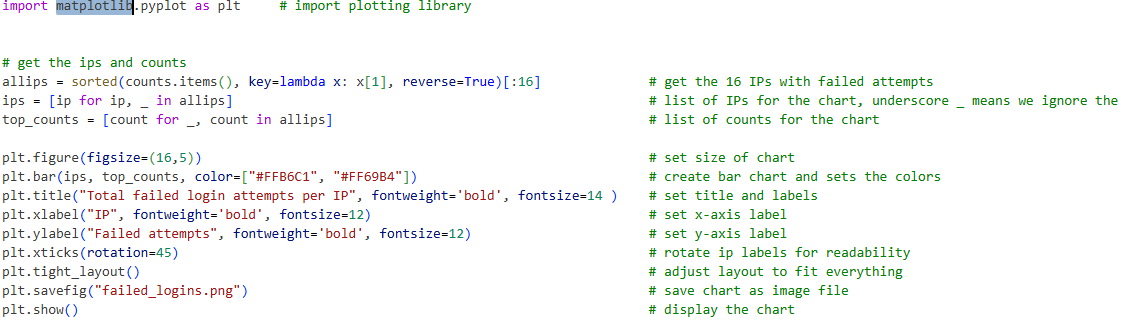
This type of attack is a common threat when a system requires user authentication, so tools like this are very useful when it comes to cybersecurity.

The tool that detects brute force attacks is divided into multiple parts.

* The first part is a function called “***parse\_auth\_line(line)***” that processes the lines in the log file, extracts and returns the IP addresses, timestamps, and event type.
* The second part is the main script, and it detects failed login attempts and groups them by IP address. It opens the log file, and then for every line we call the ***parse\_auth\_line(line)*** function to get the IP address, event type and timestamp. If the event type shows there’s been a failed login and the IP address isn’t null, the count of failed login attempts from that IP address increases. Then if the timestamp isn’t null, the program adds it to the list for the IP address.
* The third part detects brute force attacks, which is when there are 5 or more failed login attempts from one IP address within 10 minutes. If it detects an incident, then it appends the details of the attack to the lists of incidents. Then it proceeds to print out the information on the terminal.

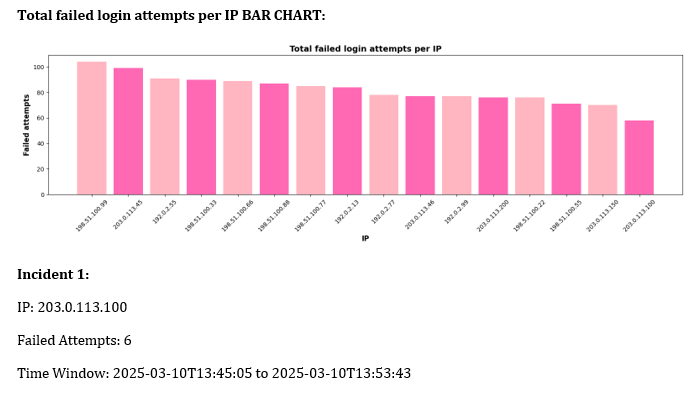
The next part of the program turns the results into a **bar chart**.

We used the plotting library called ***matplotlib***, which, according to *https://www.datacamp.com/*, is a data visualisation library in Python and allows you to generate plots, histograms, bar charts...

With the library we were able to easily add a title, labels to the x and y axis, and colours to the bar chart. The code for this is here:

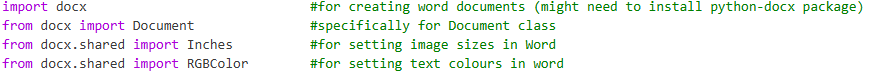
When you run the program, the image is automatically generated and saved as *failed\_logins.png*.

The next part of the code generates the report that shows the output of the tool we created. The report contains the total number of failed login attempts per IP address, the corresponding bar chart and all the incidents detected.



We decided to generate a .docx file so we could easily add the picture of the bar chart to the report, as we were not able to add the image to a text file.

Firstly, we imported the following libraries and functions that allowed us to create the document, set text colours and change the sizes of images in the file.

However, you might need to install the python-docx library for this to work. So in the terminal you need to type in *pip install python-docx*.

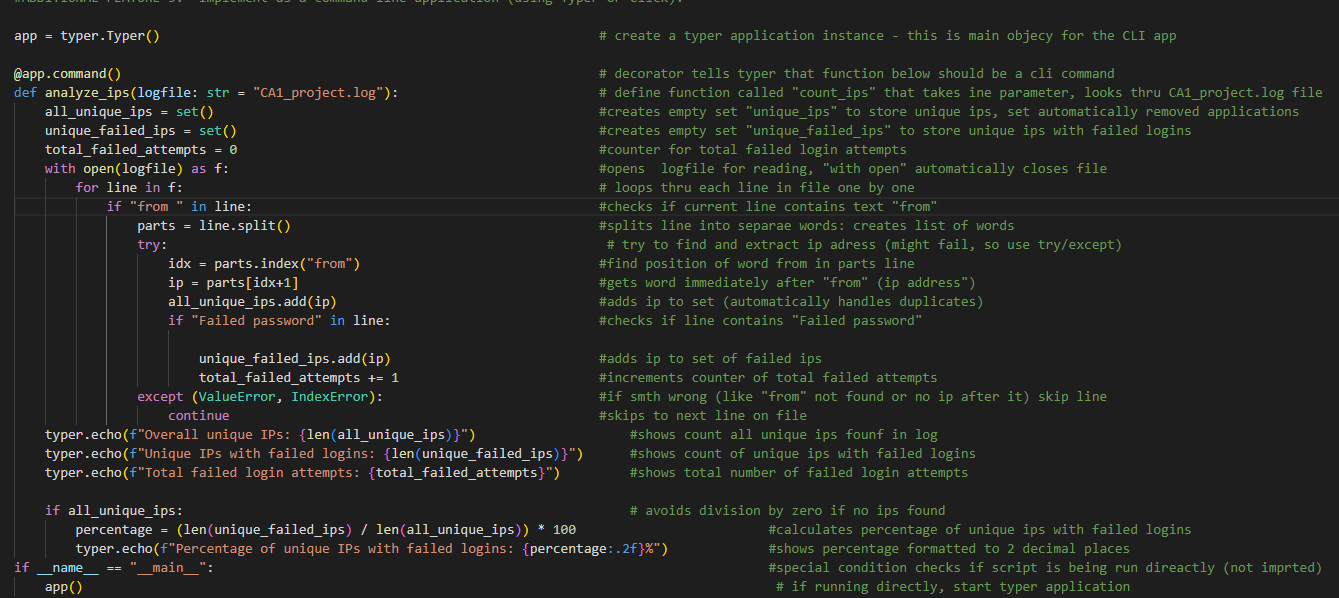
Another advantage of using the *docx* format is that it gives the report a more polished and professional appearance compared to the *txt* format.

There are two extra libraries we implemented: **typer** and **rich**.

We used the **Typer library** to turn the script into a CLI application. This allows the user to run the program directly from the terminal and receive the output from there. Typer made it easy to define and manage commands, handle arguments (in this case, the log file name) and display results in a simple and neat format.

For it to work, typer might need to be installed, so just type “*pip install typer”* in the terminal.

Typer was implemented at the end of the program.



* The statement *app = typer.Typer()* initialises a typer application.
* This object manages all the commands and options in the CLI.
* The decorator *@app.command()* creates a command called *analyze\_ips*. This function can run directly from the command line.
* This block makes sure that when the script is being executed directly, Typer runs and makes the command available:  *if \_\_name\_\_ == "\_\_main\_\_":*

*app().*

* Once implemented, the program can be executed in the terminal using this command: *python3 assignment1.py analyse-ips.*
* The user can also specify a different log file if needed, e.g., python3 assignment1.py *analyze-ips* sample\_log.log.
* Inside the *analyze\_ips* function, the program reads through the log file and calculates the number of unique IPs, the number of unique IPs with failed login attempts and the percentage of IPs that failed at least once. We used the *typer.echo()* function because it works better with CLI tools and supports a more formatted output.

**Advantages of using Typer in this program:**

* Users can run the analysis easily from a terminal with a single command.
* Commands and arguments are defined clearly, making the code easier to understand and modify.

We used the **rich library** to make the output on the terminal prettier and more organised by adding a table.

First, we imported the necessary functions using the following code.

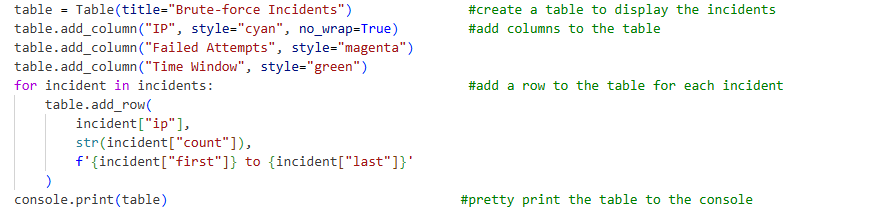


With the first line, *“from rich import print”* alone, the output on the terminal was already more organised without us having to do anything else.

With the second line, *“from rich.console import Console”,* we were able to personalise the output ourselves. For example, in this part below, we made the sentence *“Failed login attempts:”* in bold red.



With the third line, “*from rich.table import Table*”, we were able to create a table of all the brute incidents. We added a title and 3 columns for the IP, number of failed attempts and the time window. Each row is for an incident.



In conclusion, this project demonstrates how Python can be used to create a tool to automate a cybersecurity task, which is log analysis. With this tool people don’t need to waste time manually analysing log files, and it also accelerates incident response times.