**Detecting SQL Injection using OpenAI**

Submitted by

Andrew DiGeronimo

Computer Science

Gildart Haase School of Computer Sciences and Engineering

A **project report** paper submitted in partial fulfillment of the requirements for the course - **CSCI 6870 Penetration Testing & Vulnerability Analysis**.

Under the supervision

of

Dr. Yaoqing Liu

Professor, Department of Computer Science



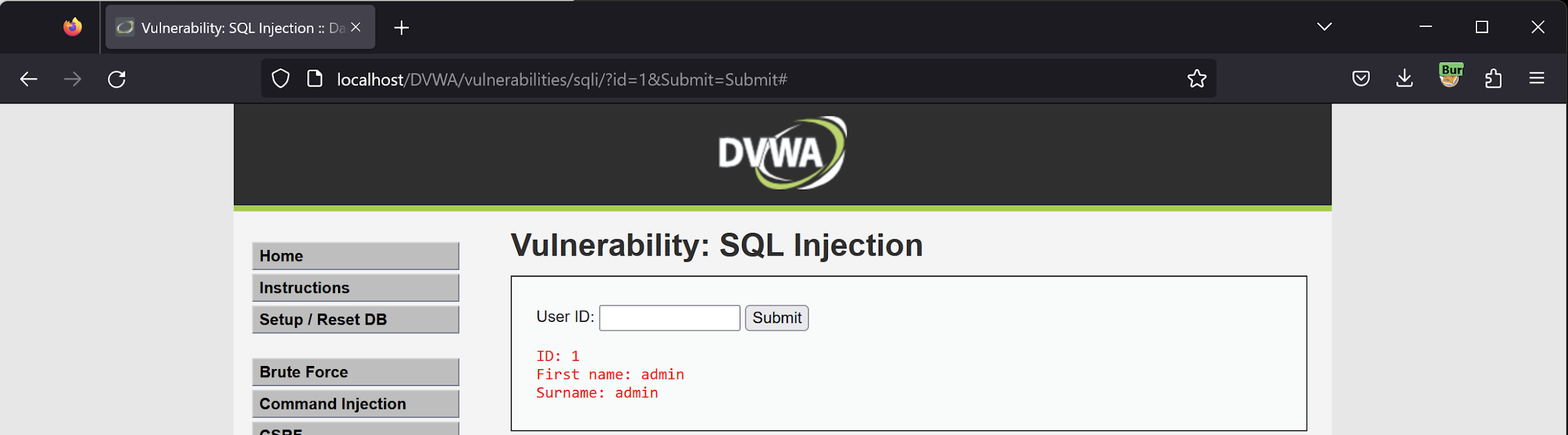
Spring 2023

**Introduction**

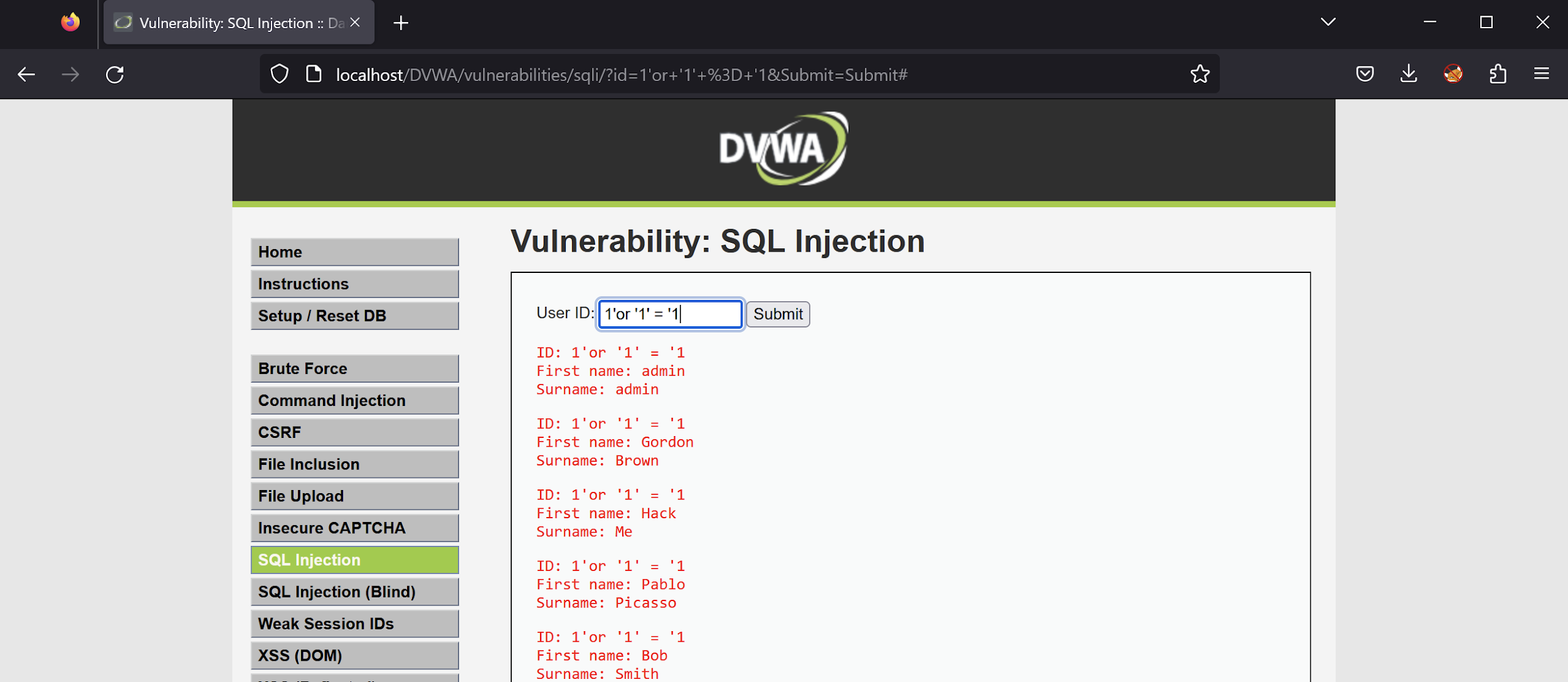
SQL injection is a common attack method used to manipulate and exploit vulnerabilities in web applications. This attack involves the attacker inputting code or queries into a web application which then interprets and potentially exposes sensitive results. Throughout history, this method of attack has been used to gain access to databases where weak security or exploits on web applications exists.  
  
Fortunately, advancements in artificial intelligence technology can allow for the detection of these attacks. Through reinforced training, an AI can learn the patterns of what makes up an SQL injection attack. Through the use of OpenAI’s API, using GPT3, we can train an AI model to detect this kind of attack. In this report, we will be discussing the training methods used to detect SQL injection attempts as well as best practices to avoid false results.

**Project Design**

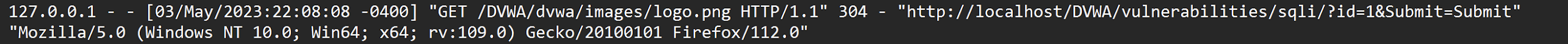
To create our test environment, we will be installing XAMPP to allow us to create a web server in our local environment. Once installed, web applications can be loaded and run for testing. We used Damn Vulnerable Web Application (DVWA) as it contains a host of web vulnerabilities at varying security levels to test. We will be focusing on the SQL Injection webpage listed on the web application (not blind).

The webpage greets us with an input field where a user ID can be typed and submitted. Normal responses (1,2,...) will yield the user with a username and surname tied to the ID. Where the exploit resides is in the input field, where none of the text inputted is filtered for special characters. Single quotations in SQL indicate the beginning or end of a string. By placing a quote in the text field, the user has the ability to run additional commands.  
  


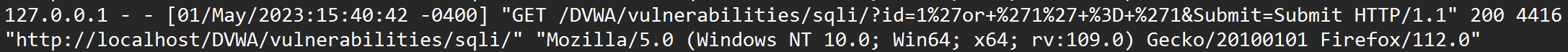
A commonly used SQL injection string that is used is 1’ OR ‘1’ = ‘1. To the database running the query, the OR command allows the user to type a value that is always true, resulting in the contents of the database being outputted.



With these methods of SQL injection documented, we then look to identify ways of training our AI to detect these injection methods. A hint for detecting SQLI can be seen in the address bar of the image above, as it contains the query string that was sent to the database. When communicating and submitting forms with the web server, there is a log that tracks the activity and traffic of the web server. To find these logs, they can be found XAMPP under the Apache tab. The logs. can be found in the file access.log.



The log file contains time and date, device identifiers, HTTP status code, and the query string of the traffic incoming to the webserver. We will be focusing on the query string as it gives us a clue as to what the user typed in the text field.

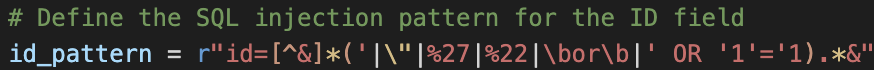


When SQLI is attempted, additional characters can be seen in the query string. Special characters ( =, &), quotes (%22, %27), and spaces should not be user-submittable and can indicate an attack.

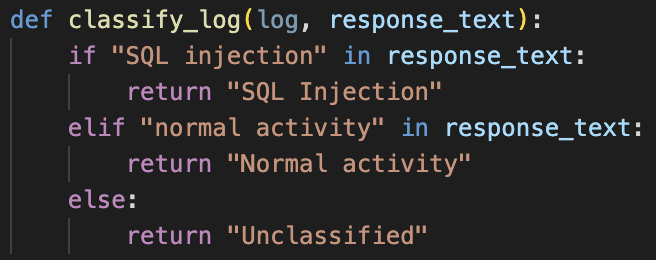
**Code Implementation**

To program the AI to learn what we are looking for, we will be writing in Python using the OpenAI API. We must first import the necessary libraries to run these commands, such as OpenAI for API support and RE for regular expression support. The API can be initialized by providing an API key.

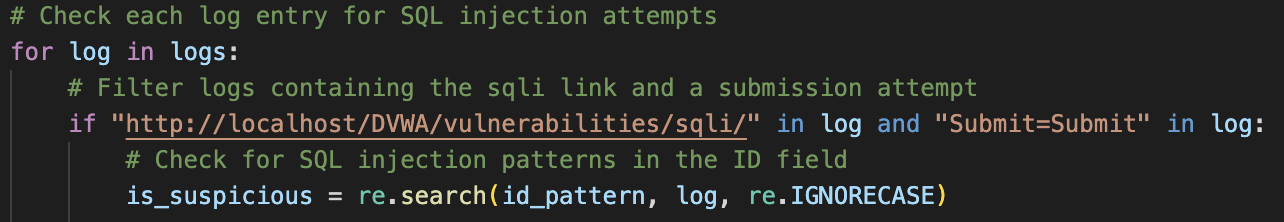
To analyze the logs, the program must have a saved version of access.log on the local machine. This can be transfered onto the local machine, and its pathname can then be copied into the program for analysis.  
  
To filter and teach the AI to detect SQL injection, we will be using a combination of regular expression filtering and training prompts. We first classify what regular expressions may indicate SQ injection by pattern. We also indicate what pattern a normal user would input for an ID to find normal activity.



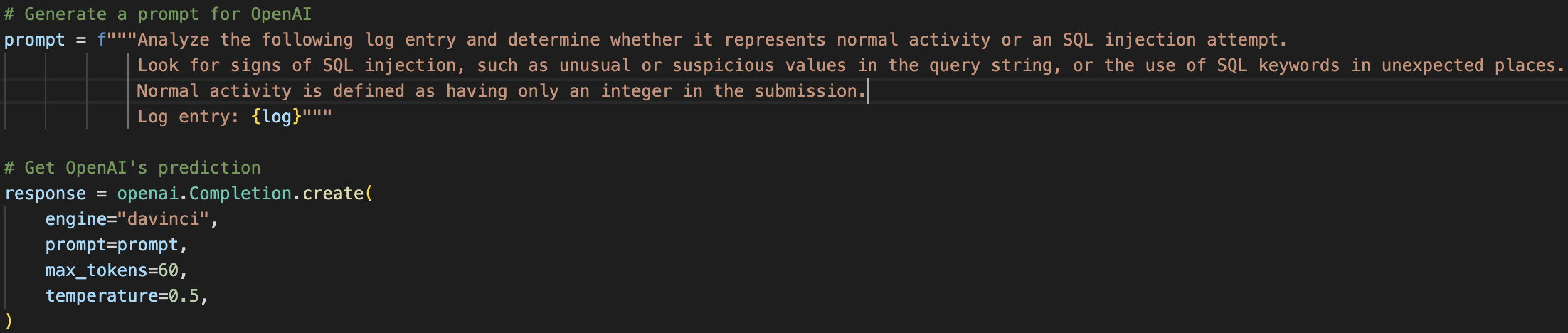
Once filtered, we will be directing the AI to only give us one of three answers to avoid overly complex outputs. The AI is directed after scanning the logs to either classify each log as SQL injection, normal activity, or unclassified if it cannot fully determine an answer.



We then add variables that hold counts of total logs analyzed as well as the results for each log. To help the administrator with detecting intrusion times, each log that is marked as SQL injection also has its date and time saved and outputted at the end of the program.

We also must clean the dataset, as the logs contain activity unrelated to the SQL injection webpage. To focus the AI to analyze activity specifically on SQL injection, we create another filter specifically looking for logs containing the web handle.

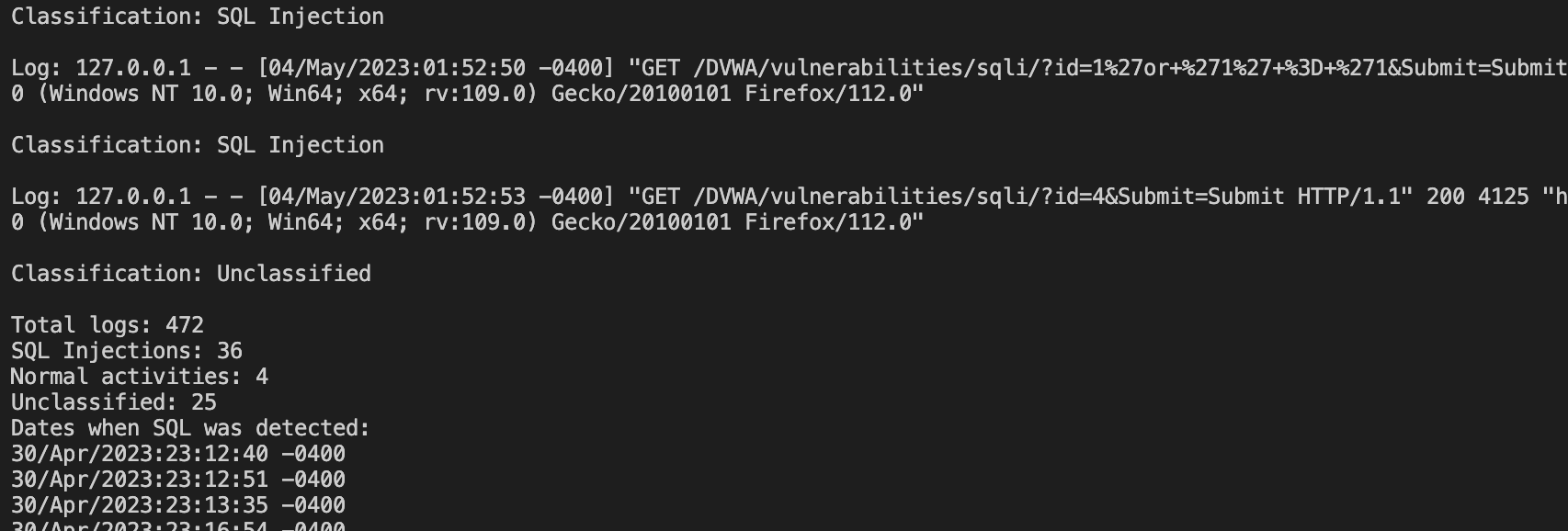
We then create a prompt for the AI to follow, being as specific as possible to help with accuracy. Once the prompt is created, we can generate a model that will analyze the logs in conjunction with the filters and prompts we gave the AI. The engine we will be using to analyze the logs is Davinci.



Once the results of the AI return back to the program, we then process the the classified results and add it to our count that is shown at the end of the program.

**Results**

When the program is ran, it will analyzie each log one by one showing in the console the result of what was classified. Once the program is complete, a full breakdown of how many logs were analyzed and their results as well as corresponding times of SQL injection attempts are outputted.



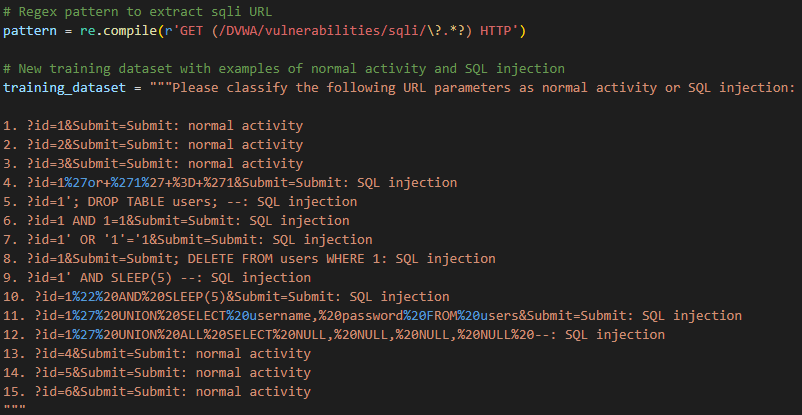
When manually inspecting logs and comparing what the AI classified the activity as, it can be seen that the AI is not as accurate as hoped. Rerunning the program may also yield different results as well. There are many potential factors that may be causing these results. The prompt given to the AI may not be specific enough, the filtering and regular expression patterns could be refined further, the model temperature may be too low (uncertain) or too high (overly certain), or other factors related to the AI models limitation of analysis. OpenAI’s GPT3 was designed to generate and tailor responses not only based on user prompts but also a number generator (seed) to add more variability. The seed value may also contribute to the varying responses and potential innacuracies.

**Further Revisions**

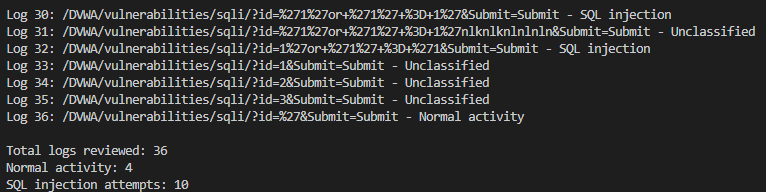
To further refine the results, we created two additional programs that focus on a different method of classification.The original program heavily uses regular expression to identify SQL injection attempts, however not all SQL injection attempts are written the same and may not fit the mold the expression is looking for. The original program may have also been checking the same logs twice to identify attempts. When a submission is made, an HTTP request containing the query string is sent along with the referrer URL. We want to specifically search for SQL injection through HTTP requests, however the referrer URL may also contain the previous attempt of SQL injection, further complicating detection. This can be seen in the provided log.

"GET /DVWA/vulnerabilities/sqli/?id=1%27or+%271%27+%3D+%271&Submit=Submit HTTP/1.1" 200 4416 "http://localhost/DVWA/vulnerabilities/sqli/?id=1%27or+%271%27+%3D+%271&Submit=Submit" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/112.0"

To avoid this in the newer revisions, we used regular expression to only grab GET requests followed by the proper URL for the DVWA SQLI webpage. This way, we only will be analyzing the GET requests as proposed. To further help with debugging, the program will also output of what log it analyzed to help us verify its results. We also refined the training prompt by providing it a dataset of SQL injection commands and activity that would be submitted through the request. This can potentially help the AI identify incoming requests that contain malicious code.



In the end, we receive more accurate results compared to the original program. The logs outputted by the program are also visually easier to read, allowing us to verify if the log matches the proposed classification by the AI.



**Conclusion**

In this study, we explored the potential of using OpenAI's GPT-3 to detect SQL injection attacks in web applications, specifically using the Damn Vulnerable Web Application (DVWA) for testing purposes. By analyzing web server access logs and implementing regular expression filtering and AI training prompts, we aimed to create a reliable method for detecting malicious SQL injection attempts.

Our initial results showed some inconsistencies in the AI's classification of SQL injection attempts, which prompted us to revise our approach by refining regular expressions, focusing on GET requests, and providing a more comprehensive dataset for training the AI. The revised approach yielded more accurate results and improved the readability of the output, making it easier for administrators to verify the AI's classifications.

While our study demonstrated the potential of AI technology in detecting SQL injection attacks, further refinement and testing are necessary to optimize the accuracy and efficiency of the system. Future work could include exploring alternative methods of classification, refining the AI training process, and testing the system on a wider range of web applications and attack patterns. By continuing to advance our understanding of AI-based detection methods, we can contribute to more secure and resilient web applications, reducing the risk of data breaches and other cyber threats.