Prevention of SQL-Injections input using Regular Expressions and Socket Programming

Syed Onais Ali Shah
National University Of
Computer And Emerging Sciences
Email: k214691@nu.edu.pk

Zohaib Saqib
National University Of
Computer And Emerging Sciences
Email: k213215@nu.edu.pk

Muhammad
National University Of
Computer And Emerging Sciences
Email: k213192@nu.edu.pk

***Abstract*—This report presents a lightweight approach to detect and mitigate SQL injection attacks using Python socket programming. The project involves creating a server-client model in which the server processes client input, evaluates their safety against SQL injection patterns, and enforces disconnection for malicious activity. Inspired by advancements in web application firewalls (WAFs), particularly the SDN-based approach discussed in Alotaibi and Vassilakis's work (2023), this implementation demonstrates how simpler tools can effectively mitigate SQL injection risks in controlled environments.**

I. Introduction

Web applications are integral to modern systems, but they face persistent threats from SQL injection attacks. These attacks exploit vulnerabilities in user input, enabling attackers to manipulate database queries and access sensitive information. As highlighted by Alotaibi and Vassilakis (2023), sophisticated solutions, such as Software-Defined Networking (SDN)-based Web Application Firewalls (WAFs), make use of advanced techniques like deep packet inspection and regular expressions to detect and mitigate such threats effectively.

While these systems offer robust protection, they are often resource-intensive and unsuitable for lightweight or small-scale applications with limited computational capacity. This creates a demand for simpler yet effective methods that can address SQL injection risks without the complexity of traditional WAFs.

This report proposes a socket programming-based system as a proof-of-concept for real-time SQL injection detection and mitigation. The system analyzes user inputs, identifies malicious input patterns, and takes immediate action to prevent potential attacks. By focusing on resource efficiency and simplicity, this project demonstrates how lightweight security solutions can complement existing methods to protect web applications in environments where computational resources are constrained.

II. Literature Review

2.1 SQL Injection

SQL injection is one of the most prevalent vulnerabilities in web applications, as outlined in OWASP's Top 10 Security Risks. It allows attackers to bypass authentication mechanisms, extract sensitive data, or alter database contents by injecting malicious SQL code into user input.

2.2 Web Application Firewalls

Traditional WAFs, such as ModSecurity, provide robust solutions against web-based attacks, but often require significant computational resources. Modern advancements like SDN-based WAFs, discussed in the work of Alotaibi and Vassilakis, demonstrate enhanced detection accuracy through centralized network control and deep packet inspection.

2.3 Lightweight Detection Systems

While advanced systems focus on large-scale deployments, smaller setups can also detect common SQL injection patterns using simple rule-based approaches. This project emphasizes lightweight implementation using Python socket programming.

III. Methodology

3.1 System Design

The system consists of two components:

Server: Listens for incoming client connections. Processes input strings from clients. Matches input against SQL injection patterns using regular expressions. Disconnects malicious clients or returns safe query responses.

Client: Connects to the server. Sends user-provided input strings. Displays server responses or handles disconnection.

3.2 Patterns for Detection

Inspired by the WAF methodologies in the paper by Alotaibi and Vassilakis, this project uses regular expressions to detect:

Malicious WHERE and UPDATE clauses. Multiline SQL statements. Common attack patterns like UNION SELECT, DROP TABLE, and –.

3.3 Code Structure

Server Code: Handles incoming connections, evaluates inputs, and manages disconnections.

Client Code: Sends inputs to the server and receives responses.

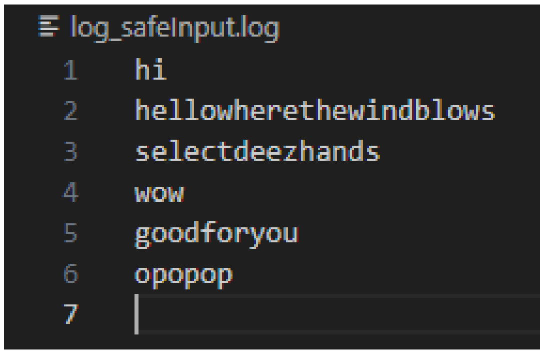
Example Detection Logic:

```python
import re

patterns = [
    r"union\s+select",
    r"or\s+1=1",
    r"order\s+by",
    r"--",
    r"drop\s+table",
    r"\\bwhere\\b.*\\b(or|and)\\b.*="
]

def is_sql_injection(query):
    return (re.search(pattern, query,
            re.IGNORECASE) for pattern
            in patterns)
```

IV. Results

The implementation provided in this project has worked effectively against multiple trialed inputs, showing a clear understanding of how to differentiate between inputs that can pose a threat to the system via SQL injections. Here is an example list comparing allowed inputs with those that pose a risk to the system.

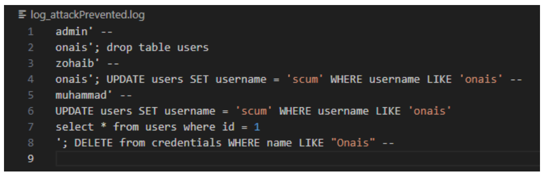
Allowed inputs



```
log_safeInput.log
hi
hellowherethewindblows
selectdeezhands
wow
goodforyou
opopop
```

Fig. 1. Screenshot for log file of allowed inputs

Potentially harmful inputs



```
log_attackPrevented.log
admin' --
onais'; drop table users
zohaib' --
onais'; UPDATE users SET username = 'scum' WHERE username LIKE 'onais' --
muhammad' --
UPDATE users SET username = 'scum' WHERE username LIKE 'onais'
select * from users where id = 1
'; DELETE from credentials WHERE name LIKE "Onais" --
```

Fig. 2. Screenshot for log file of malicious input

V. Discussion

Research has shown that using string signatures (searching for keywords in strings) has been unhelpful in countering SQLi attacks as it is easy to circumvent signatures. In contrast, regular expressions have shown great promise since they are able to scan user input in a faster and more reliable manner by making use of pattern matching in user input rather than depending on signatures. In the future, it is predicted that lightweight Large Language Models (LLMs) could theoretically be trained to assess user input even more accurately than regular expressions. This assumption is driven by the recent boom in the AI industry, showing how adaptable systems integrating it can become. This advancement, however, would increase computational complexity, opposing this project's goal of reducing it. Until it can be shown that the computation required for an LLM can be justified against the simplicity of regular expressions, it is unlikely that LLM's will be used to perform Deep Packet Inspection for SDN-based WAF's.

VI. Conclusion

This project demonstrates the feasibility of a lightweight SQL injection detection system using Python socket programming. Although it lacks the scalability and comprehensive capabilities of traditional or SDN-based WAFs, it provides an effective, resource-efficient solution for small-scale applications. Future iterations could integrate database interactions and improve pattern coverage. The implementation of the project is provided at https://github.com/onaisali4952/SQLi-Detection-SocketProgramming-Project.

References

[1] Alotaibi, F. M. and Vassilakis, V. G., *Toward an SDN-Based Web Application Firewall: Defending Against SQL Injection Attacks*, Future Internet, 15(5), 170, 2023, https://doi.org/10.3390/fi15050170.