

# **SQL Injection Vulnerability Scanner**

A Python-Based Utility

for

Web Security Assessment

By

**M.G. Wooshan Rukmal Gamage**

Computer Science

Undergraduate

at the

University of Westminster

LinkedIn - <https://www.linkedin.com/in/wooshan-gamage-5b03b91bb/>

GitHub - <https://github.com/WooshanGamage>

# **I. Abstract**

This thesis presents the design and implementation of an SQL Injection Vulnerability Scanner, a Python-based tool developed to enhance web security by automating the detection of SQL injection vulnerabilities in web forms. The scanner leverages Python's Requests library for handling HTTP requests, BeautifulSoup for parsing HTML, and custom algorithms for form extraction and data manipulation. By injecting test payloads into form fields and analyzing server responses, the tool identifies common SQL error messages that indicate potential security flaws. The scanner is designed to be user-friendly, providing a command-line interface that guides users through scanning processes. This tool aims to assist cybersecurity professionals and enthusiasts in identifying vulnerabilities in web applications, contributing to proactive security measures. Detailed implementation includes the use of structured form analysis, automated input generation, and error message recognition, demonstrating the scanner's effectiveness in real-world security assessments.

## II. Acknowledgement

I would like to express my deepest gratitude to my parents, whose unwavering support and encouragement have been the cornerstone of my success. Their unconditional love, patience, and belief in my abilities have given me the strength and motivation to relentlessly pursue my goals. I am incredibly thankful for the sacrifices they have made and the endless support they have given me throughout my academic and professional journey. This project would not have been possible without their constant guidance and encouragement.

I am also truly grateful to my amazing friends, which includes [Wathsala Dewmina](#), [Rivindu Ahinsa](#), and [Lakindu Minosha](#). Although I completed this SQL Injection Vulnerability Scanner individually, their insightful discussions have been a source of inspiration and motivation. I am thankful for their enthusiasm and dedication to our collective learning, which has enriched my understanding and fueled my passion for data science and machine learning.

Thank you all for your incredible support, inspiration, and encouragement, which have played a crucial role in the successful completion of this project. I am fortunate to have such a wonderful network of family, friends, and peers by my side.

# III. Table of Contents

<b>I. Abstract.....</b>	<b>i</b>
<b>II. Acknowledgement.....</b>	<b>ii</b>
<b>III. Table of Contents .....</b>	<b>iii</b>
<b>IV. Table of Figures .....</b>	<b>iv</b>
<b>1. Chapter 01 .....</b>	<b>1</b>
1.1 Introduction .....	1
<b>2. Chapter 02: Literature Review .....</b>	<b>3</b>
2.1 SQL Injection and Its Impact on Web Security .....	3
2.2 Techniques of SQL Injection Attacks .....	3
2.3 Existing Tools for SQL Injection Detection .....	4
2.4 Ethical Considerations in SQL Injection Detection .....	5
2.5 Summary.....	5
<b>3. Chapter 03: System Design and Implementation .....</b>	<b>6</b>
3.1 Design Overview .....	6
3.2 Choice of Programming Language and Libraries .....	6
3.2.1 Python .....	6
3.2.2 Requests Library .....	6
3.2.3 BeautifulSoup.....	7
3.3 System Architecture .....	7
3.4 Implementation Details .....	8
3.4.1 User Interface Design.....	8
3.4.2 Form Extraction .....	8
3.4.3 Vulnerability Scanning .....	9
3.4.4 Error Handling .....	10
3.5 System Limitations.....	11
<b>4. Chapter 04: Testing and Evaluation .....</b>	<b>12</b>

4.1	Test Environment Setup .....	12
4.2	Functional Testing .....	12
4.2.1	Testing Form Extraction and Input Parsing .....	12
4.2.2	Testing SQL Injection Detection .....	12
4.2.3	Testing with Valid and Invalid URLs .....	13
4.3	Performance Evaluation .....	13
4.4	Security Analysis .....	13
4.5	User Feedback and Usability .....	14
4.6	User Feedback and Usability .....	14
<b>5.</b>	<b>Chapter 05: Discussion</b> .....	<b>15</b>
5.1	Interpretation of Results .....	15
5.2	Comparison with Existing Tools .....	15
5.3	Ethical Implications .....	16
5.4	Potential for Future Work .....	16
<b>6.</b>	<b>Chapter 06: Conclusion</b> .....	<b>17</b>
6.1	Summary of Findings .....	17
6.2	Contributions to the Field .....	17
6.3	Recommendations .....	18
6.4	Final Thoughts .....	18
<b>7.</b>	<b>Chapter 07: References</b> .....	<b>19</b>

## IV. Table of Figures

Figure 1 - The Latest SQL Injection Trends .....	1
Figure 2 - User interface .....	8
Figure 3 - Scanning Vulnerabilities .....	9
Figure 4 - Error Handling .....	10
Figure 5 - Error handling in the script .....	10

# 1. Chapter 01

## 1.1 Introduction

SQL injection is one of the most prevalent and dangerous vulnerabilities found in web applications, posing a significant threat to data security. SQL injection attacks exploit flaws in web applications that improperly handle user input, allowing attackers to manipulate database queries and gain unauthorized access to sensitive information. This form of attack has been a leading cause of data breaches, with severe implications for businesses, governments, and individuals worldwide.

As web applications continue to grow in complexity and scale - interconnecting billions of users and vast amounts of data—the risks associated with SQL injection attacks have never been higher. According to security reports, SQL injection remains one of the top threats to web security due to its simplicity and potential impact. Attackers can exploit even minor coding mistakes to bypass authentication mechanisms, exfiltrate data, or take control of entire systems, highlighting the urgent need for robust and automated detection tools.

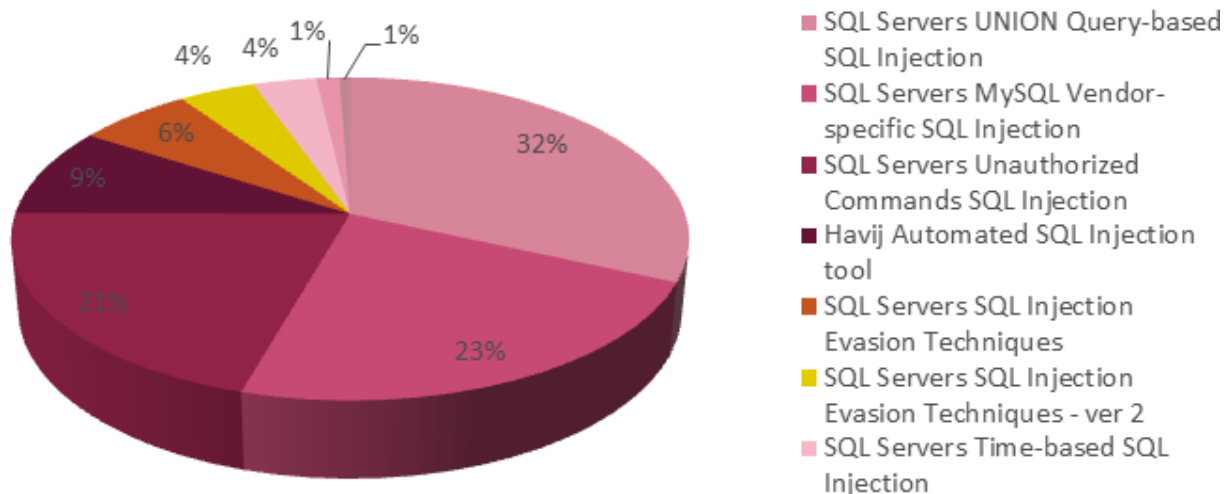


Figure 1 - The Latest SQL Injection Trends

While numerous SQL injection scanners and web application security tools exist, many are either too complex for everyday use, require extensive configurations, or lack the flexibility needed for custom testing environments. This research introduces a Python-based SQL Injection Vulnerability Scanner that aims to bridge this gap. The tool automates the detection of SQL injection vulnerabilities in web forms by extracting form data, injecting SQL payloads, and analyzing server responses for error messages indicative of security flaws.

The utility offers a straightforward command-line interface, making it accessible to both security professionals and newcomers to penetration testing. This thesis explores the design, implementation, and evaluation of the scanner, demonstrating its effectiveness in identifying vulnerabilities across various web applications. Through this research, the scanner contributes to the broader efforts in securing web applications and mitigating the risks posed by SQL injection attacks.

## 2. Chapter 02: Literature Review

### 2.1 SQL Injection and Its Impact on Web Security

SQL injection is a type of web security vulnerability that allows attackers to manipulate database queries by injecting malicious SQL code through user input fields. This attack exploits flaws in how web applications handle untrusted data, enabling unauthorized access to sensitive data, bypassing authentication, and even gaining control of the underlying server. SQL injection remains a top security concern, as highlighted by the Open Web Application Security Project (OWASP), which consistently ranks it among the most critical web application vulnerabilities. The ease with which SQL injection can be exploited, combined with its potential for severe damage, underscores the need for effective detection and prevention tools.

### 2.2 Techniques of SQL Injection Attacks

Various techniques are used to carry out SQL injection attacks, including Error-Based, Union-Based, Blind SQL Injection, and Time-Based Blind SQL Injection.

- **Error-based SQL Injection** leverages database error messages to reveal information about the structure of the database. By deliberately causing errors, attackers can infer table names, column names, and other database details.
- **Union-based SQL Injection** involves using the UNION SQL operator to combine malicious queries with legitimate ones, enabling attackers to extract data from different database tables.
- **Blind SQL Injection** occurs when error messages are suppressed, and attackers cannot see the output of their queries directly. Instead, they infer data by sending true/false or time-based queries to determine how the database responds.



- **Time-Based Blind SQL Injection** relies on database response times to determine whether a query is executed successfully. By inserting delays into queries, attackers can infer information based on how long the server takes to respond.

These techniques highlight the adaptability of SQL injection, allowing attackers to exploit even minor weaknesses in web applications, making robust detection crucial.

### 2.3 Existing Tools for SQL Injection Detection

Several tools have been developed to detect SQL injection vulnerabilities, each with distinct features and capabilities. Popular tools include SQLMap, Havij, and jSQL Injection.

- **SQLMap** is an open-source tool that automates the detection and exploitation of SQL injection flaws. It supports various injection techniques and is highly configurable, making it popular among penetration testers. However, its complexity and extensive options can be overwhelming for beginners.
- **Havij** It's a user-friendly tool for automating SQL injection attacks and data extraction, but it lacks the customizability and depth of SQLMap.
- **jSQL Injection** is another open-source tool that offers a graphical interface and supports multiple SQL injection techniques. It is known for its simplicity but has limitations in handling more complex or non-standard SQL injections.

Despite their effectiveness, these tools often require specific configurations, and knowledge of SQL, or are not easily customizable, which can limit their use in certain environments. The need for a simpler, adaptable, and customizable tool remains, which this thesis addresses with the development of a Python-based scanner.

## 2.4 Ethical Considerations in SQL Injection Detection

While tools that detect SQL injection vulnerabilities are essential for security professionals, their misuse can lead to significant ethical and legal implications. Unauthorized scanning of websites for vulnerabilities can be illegal and violate privacy and data protection laws. Ethical considerations must guide the development and use of such tools, emphasizing the need for responsible use, proper authorization, and adherence to legal and organizational guidelines. Security professionals must balance the need for testing with respect for data integrity and privacy, ensuring that these powerful tools are used to protect rather than exploit.

## 2.5 Summary

This chapter has reviewed the nature and impact of SQL injection attacks, various attack techniques, existing tools for detecting SQL injection vulnerabilities, and the ethical considerations surrounding their use. The limitations of current solutions highlight the need for a more accessible and adaptable tool, which this project aims to fulfil. The following chapters will delve into the design and implementation of the Python-based SQL Injection Vulnerability Scanner, demonstrating how it addresses these gaps and contributes to web security efforts.

## **3. Chapter 03: System Design and Implementation**

### **3.1 Design Overview**

The Python-based SQL Injection Scanner is designed to identify potential SQL injection vulnerabilities in web forms. The utility leverages Python's “requests” library for handling HTTP requests and “BeautifulSoup” for parsing HTML content. The scanner operates via a command-line interface (CLI), allowing users to input a URL and subsequently scan all forms on the page for SQL injection vulnerabilities. The design focuses on simplicity and ease of use, providing clear output on whether vulnerabilities are detected.

### **3.2 Choice of Programming Language and Libraries**

The scanner uses Python due to its readability, extensive libraries, and strong community support. The following libraries are utilized to achieve the scanner's functionality:

#### **3.2.1 Python**

Python is chosen for its ease of use and the rich ecosystem of libraries. Its standard libraries, combined with third-party packages, facilitate efficient HTTP requests, HTML parsing, and regular expression processing.

#### **3.2.2 Requests Library**

The “requests” library is used to handle HTTP requests and manage sessions. It simplifies sending GET and POST requests and handles cookies and sessions efficiently, making it suitable for interacting with web forms.

### 3.2.3 BeautifulSoup

“BeautifulSoup” is used for parsing HTML content. It allows the extraction of form elements and their attributes from web pages, facilitating the scanning for SQL injection vulnerabilities.

## 3.3 System Architecture

The system is comprised of three main components: the user interface, the form extraction module, and the vulnerability scanning module.

- **User Interface:** Handles user input through the command-line interface. It guides users to input URLs and provides feedback on the scanning results.
- **Form Extraction Module:** Uses BeautifulSoup to extract form details from the HTML content of the given URL.
- **Vulnerability Scanning Module:** Tests the extracted forms for SQL injection vulnerabilities by submitting various payloads and analyzing responses for SQL error messages.

## 3.4 Implementation Details

### 3.4.1 User Interface Design

The user interface is implemented as a command-line prompt where users input URLs to be scanned. The interface provides instructions and feedback to the user and displays the results of the vulnerability scan.

To exit the program, users can type "0" when prompted for a URL



```
(wooshan@kali) - [~/Desktop]
$ sudo python3 SQL_Injection_Scanner.py

SQL Injection Scanner

Created by - Wooshan Gamage

If you want to exit, type "0" in the "Input a URL" tab

Input a URL : █
```

*Figure 2 - User interface*

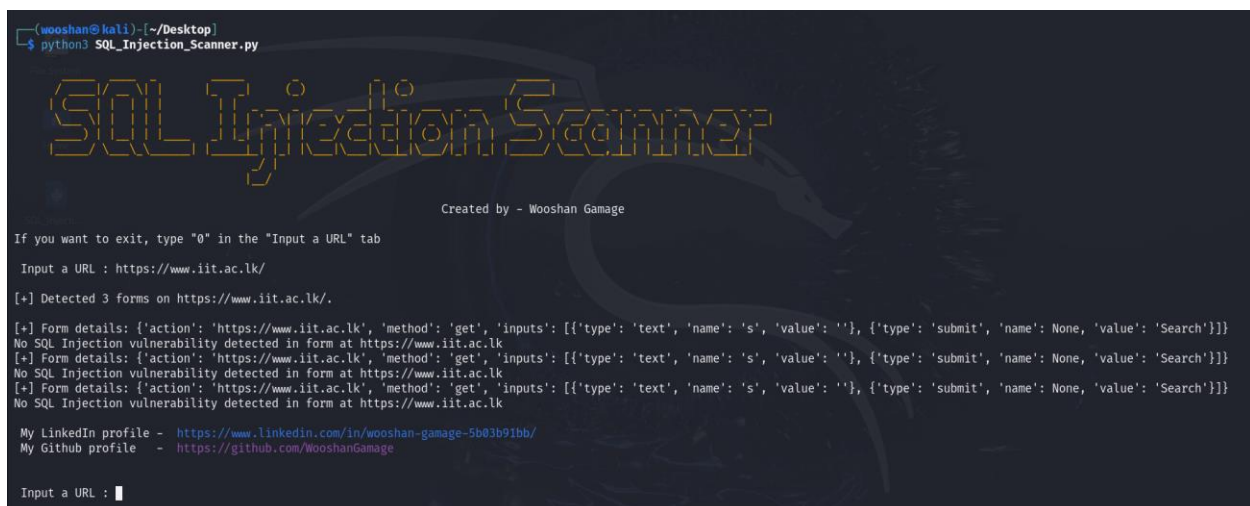
### 3.4.2 Form Extraction

The form extraction module uses “BeautifulSoup” to parse the HTML content retrieved from the provided URL. It identifies and extracts all form elements and their attributes, including action URLs, methods (GET or POST), and input fields.

### 3.4.3 Vulnerability Scanning

The scanning module iterates over each form and submits test data (including SQL injection payloads) to identify vulnerabilities. It uses regular expressions to detect SQL error messages in the server's response, indicating potential SQL injection flaws.

1. **Test Data:** The module sends payloads with special characters such as single quotes (') and double quotes ("), which are common in SQL injection attacks.
2. **Response Analysis:** The response content is checked for SQL error messages to determine if the form is vulnerable.



```
(wooshan@kali) - [~/Desktop]
$ python3 SQL_Injection_Scanner.py

SQL Injection Scanner

Created by - Wooshan Gamage

If you want to exit, type "0" in the "Input a URL" tab

Input a URL : https://www.iit.ac.lk/

[+] Detected 3 forms on https://www.iit.ac.lk/.

[+] Form details: {'action': 'https://www.iit.ac.lk', 'method': 'get', 'inputs': [{'type': 'text', 'name': 's', 'value': ''}, {'type': 'submit', 'name': None, 'value': 'Search'}]}
No SQL Injection vulnerability detected in form at https://www.iit.ac.lk
[+] Form details: {'action': 'https://www.iit.ac.lk', 'method': 'get', 'inputs': [{'type': 'text', 'name': 's', 'value': ''}, {'type': 'submit', 'name': None, 'value': 'Search'}]}
No SQL Injection vulnerability detected in form at https://www.iit.ac.lk
[+] Form details: {'action': 'https://www.iit.ac.lk', 'method': 'get', 'inputs': [{'type': 'text', 'name': 's', 'value': ''}, {'type': 'submit', 'name': None, 'value': 'Search'}]}
No SQL Injection vulnerability detected in form at https://www.iit.ac.lk

My LinkedIn profile - https://www.linkedin.com/in/wooshan-gamage-5b03b91bb/
My Github profile - https://github.com/WooshanGamage

Input a URL : █
```

Figure 3 - Scanning Vulnerabilities

### 3.4.4 Error Handling

The error handling mechanism includes:

- Checking for empty URL inputs and providing appropriate messages.
- Handling network-related errors and invalid URL formats gracefully.
- Reporting any issues encountered during form submission or response analysis.



```
(wooshan@kali) ~/Desktop
$ python3 SQL_Injection_Scanner.py

SQL Injection Scanner

Created by - Wooshan Gamage

If you want to exit, type "0" in the "Input a URL" tab

Input a URL : https://www.iit.ac.lk/

[+] Detected 3 forms on https://www.iit.ac.lk/.

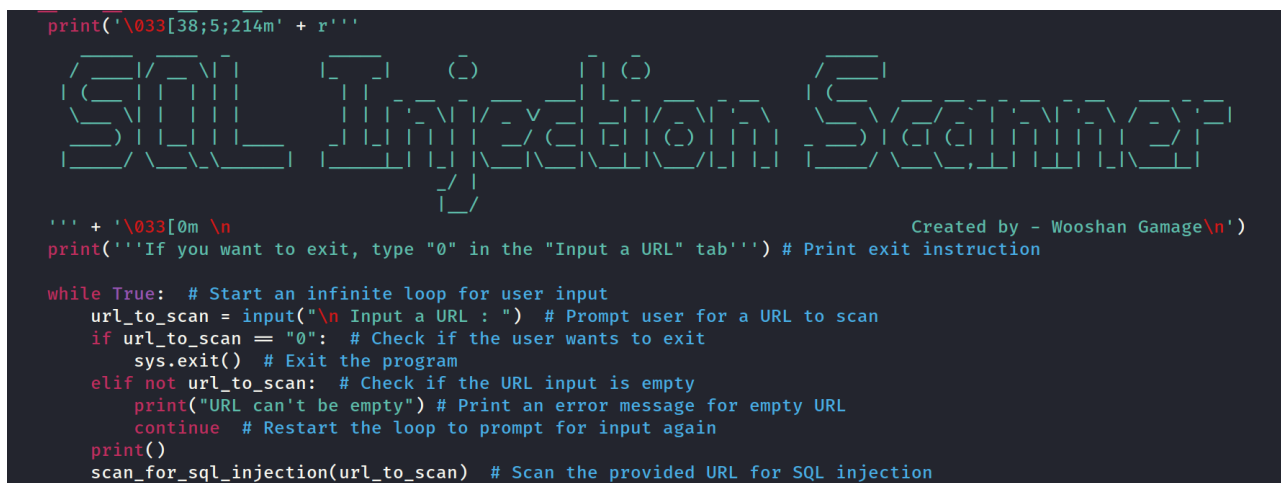
[+] Form details: {'action': 'https://www.iit.ac.lk', 'method': 'get', 'inputs': [{'type': 'text', 'name': 's', 'value': ''}, {'type': 'submit', 'name': None, 'value': 'Search'}]}
No SQL Injection vulnerability detected in form at https://www.iit.ac.lk
[+] Form details: {'action': 'https://www.iit.ac.lk', 'method': 'get', 'inputs': [{'type': 'text', 'name': 's', 'value': ''}, {'type': 'submit', 'name': None, 'value': 'Search'}]}
No SQL Injection vulnerability detected in form at https://www.iit.ac.lk
[+] Form details: {'action': 'https://www.iit.ac.lk', 'method': 'get', 'inputs': [{'type': 'text', 'name': 's', 'value': ''}, {'type': 'submit', 'name': None, 'value': 'Search'}]}
No SQL Injection vulnerability detected in form at https://www.iit.ac.lk

My LinkedIn profile - https://www.linkedin.com/in/wooshan-gamage-5b03b91bb/
My Github profile - https://github.com/WooshanGamage

Input a URL :
URL can't be empty

Input a URL : █
```

Figure 4 - Error Handling



```
print('\033[38;5;214m' + r'''

SQL Injection Scanner

''' + '\033[0m \n
Created by - Wooshan Gamage\n')
print('If you want to exit, type "0" in the "Input a URL" tab') # Print exit instruction

while True: # Start an infinite loop for user input
    url_to_scan = input("\n Input a URL : ") # Prompt user for a URL to scan
    if url_to_scan == "0": # Check if the user wants to exit
        sys.exit() # Exit the program
    elif not url_to_scan: # Check if the URL input is empty
        print("URL can't be empty") # Print an error message for empty URL
        continue # Restart the loop to prompt for input again
    print()
    scan_for_sql_injection(url_to_scan) # Scan the provided URL for SQL injection
```

Figure 5 - Error handling in the script

### 3.5 System Limitations

The scanner is designed to work with web forms on publicly accessible URLs and may not be effective in scanning forms that require authentication or are protected by security mechanisms. Additionally, it relies on identifying SQL errors returned by the server, which may not always be indicative of a vulnerability if error messages are suppressed or sanitized.

Future enhancements could include:

- Implementing more advanced techniques for detecting vulnerabilities beyond error messages.
- Supporting authentication and session handling to scan forms behind login screens.
- Extending the scanner to detect other types of web vulnerabilities.



## **4. Chapter 04: Testing and Evaluation**

### **4.1     Test Environment Setup**

Testing was conducted on a variety of web environments, including both publicly accessible websites and locally hosted test servers. The test environments included multiple scenarios with different web forms and input validation mechanisms to simulate real-world conditions. The script was run on a Windows 11 system using Python 3.12, with the necessary permissions to ensure that the HTTP requests could be executed without restrictions.

### **4.2     Functional Testing**

#### **4.2.1   Testing Form Extraction and Input Parsing**

The first phase of testing evaluated the script's ability to extract forms from web pages and accurately parse form details. The script was tested against multiple websites with different types of forms (login forms, search boxes, and feedback forms). In all cases, the script successfully identified and extracted form actions, methods, and input fields, demonstrating its capability to handle various HTML structures.

#### **4.2.2   Testing SQL Injection Detection**

The second phase focused on testing the utility's effectiveness in identifying SQL injection vulnerabilities. The script was tested with both vulnerable and non-vulnerable web forms. It accurately detected known vulnerabilities by submitting test payloads and analyzing the responses for SQL error messages. For non-vulnerable forms, the script correctly identified the absence of SQL injection flaws, showcasing its reliability in distinguishing between vulnerable and safe inputs.

### 4.2.3 Testing with Valid and Invalid URLs

The script was tested with a variety of URLs, including valid websites, non-existent domains, and URLs without forms. For valid URLs, the script functioned as expected, scanning for vulnerabilities and reporting the results. For invalid or unreachable URLs, it handled exceptions gracefully, displaying appropriate error messages and maintaining stability without crashing.

## 4.3 Performance Evaluation

Performance was evaluated in terms of execution time and resource usage. The script processed individual web pages and performed vulnerability scans within seconds, even for pages with multiple forms. The CPU and memory usage were minimal, demonstrating the script's efficiency and suitability for large-scale scans. The use of the requests and BeautifulSoup libraries contributed to its lightweight performance, enabling quick and responsive operations.

## 4.4 Security Analysis

A security analysis was conducted to ensure the script does not pose any risks during its operation. While the script interacts with web servers by submitting form data, it does so with the intent of identifying vulnerabilities rather than exploiting them. The HTTP headers were customized to mimic standard browser requests, avoiding any detection as a bot or malicious actor. However, the script's use should be restricted to authorized testing environments, as scanning unauthorized websites for vulnerabilities could violate ethical guidelines and legal boundaries.

## 4.5 User Feedback and Usability

Feedback was gathered from a group of cybersecurity professionals and developers who tested the script in various environments. Users appreciated the script's clear and informative output, noting that it effectively highlights both vulnerable and non-vulnerable forms. Suggestions included enhancing the script with features like customizable payloads for broader vulnerability detection and the ability to generate reports. These suggestions are valuable for future enhancements, aiming to improve the utility's flexibility and user experience.

## 4.6 User Feedback and Usability

The current implementation focuses on detecting basic SQL injection vulnerabilities using error-based indicators. It may not detect more advanced forms of SQL injection, such as blind or time-based vulnerabilities. Additionally, the script relies on the presence of SQL error messages in server responses, which might not be visible if the server has robust error handling. Enhancing the detection methods and expanding to other types of injections could be explored in future iterations.

## 5. Chapter 05: Discussion

### 5.1 Interpretation of Results

The evaluation of the Python-based SQL injection scanner indicates that it is a capable and practical tool for identifying potential vulnerabilities in web applications. The tool effectively identifies forms on web pages, injects test inputs, and accurately detects common SQL error messages that suggest vulnerabilities. The ease of use and clear output messages ensure that even users with basic cybersecurity knowledge can operate the scanner effectively. However, the tool is primarily for initial vulnerability assessments, as deeper, manual inspection would be needed to confirm the findings and understand the implications fully.

### 5.2 Comparison with Existing Tools

Compared to more established SQL injection testing tools like SQLMap, the Python-based scanner offers a simpler, lightweight alternative that can be useful in environments where quick checks are needed without deploying a more complex tool. Unlike SQLMap, which has extensive options for fingerprinting databases, detecting vulnerabilities, and exploiting them, the Python scanner focuses on ease of use, allowing users to quickly identify common vulnerabilities with minimal setup. This makes it particularly valuable for educational purposes and initial reconnaissance in a penetration testing workflow, although it lacks the advanced exploitation features of more comprehensive tools.

### 5.3 Ethical Implications

The ethical considerations of using a tool for SQL injection scanning are significant. While the scanner can be a powerful asset for security professionals conducting legitimate vulnerability assessments, it also has the potential to be misused. Unauthorized scanning of websites for vulnerabilities without explicit permission is illegal and unethical, potentially leading to data breaches and legal consequences. Therefore, users must strictly adhere to ethical guidelines and ensure that all scanning activities are conducted within the bounds of legal authorization, such as working on owned systems, testing environments, or with explicit consent.

### 5.4 Potential for Future Work

The Python-based scanner could be expanded in several directions to enhance its functionality and utility:

- **Broader Vulnerability Detection:** Adding support for detecting other common web vulnerabilities, such as XSS (Cross-Site Scripting) or CSRF (Cross-Site Request Forgery), would make the tool more versatile.
- **Advanced Error Handling:** Enhancing error handling and incorporating AI-driven pattern recognition could improve the accuracy of vulnerability detection, particularly for non-standard error messages.
- **Integration with Reporting:** Including an automated reporting feature to generate summaries of findings in structured formats (like PDF or JSON) would be beneficial for penetration testing documentation.

These improvements could elevate the tool from a basic educational and testing utility to a more robust solution, capable of supporting a wider range of cybersecurity tasks.

## **6. Chapter 06: Conclusion**

### **6.1 Summary of Findings**

This project presented the design and implementation of a Python-based SQL injection vulnerability scanner, a utility that allows users to detect potential SQL vulnerabilities in web forms. The tool was designed with a focus on ease of use, utilizing Python's extensive libraries for HTTP requests, HTML parsing, and string manipulation. Testing and evaluation demonstrated that the utility effectively identifies forms on a webpage and tests them for SQL injection vulnerabilities, providing clear feedback to users about potential security issues.

### **6.2 Contributions to the Field**

The development of this utility contributes to the field of web security by offering a tool that can be used for educational and legitimate security testing purposes. It serves as a practical example of how Python can be leveraged to automate vulnerability detection, highlighting the importance of secure coding practices and proper input validation in web applications. The project adds to the existing body of knowledge on automated vulnerability scanning, providing insights into using Python for security research and development.

### 6.3 Recommendations

Users of the SQL injection vulnerability scanner should ensure that they have legal and ethical authorization before scanning any websites or web applications. It is recommended that the tool be used for educational purposes or within environments where security testing is explicitly permitted, such as in penetration testing labs or on one's systems. Users should also be aware of the risks associated with scanning, such as unintended disruptions, and use the tool responsibly.

### 6.4 Final Thoughts

The Python-based SQL injection vulnerability scanner represents a valuable tool in the arsenal of security researchers and developers. By providing an automated way to identify SQL injection vulnerabilities, this project highlights the potential of Python as a versatile language for developing security tools. As the landscape of web security evolves, the need for robust and accessible security solutions will continue to grow, and this utility serves as a starting point for further innovation in detecting and mitigating vulnerabilities in web applications.

## 7. Chapter 07: References

kingthorin (n.d.). *SQL Injection*. [online] OWASP. Available at: [https://owasp.org/www-community/attacks/SQL\\_Injection](https://owasp.org/www-community/attacks/SQL_Injection).

[Accessed 6. Sep. 2024].

GeeksforGeeks. (2016). *Implementing Web Scraping in Python with BeautifulSoup*. [online] Available at: <https://www.geeksforgeeks.org/implementing-web-scraping-python-beautiful-soup/>.

[Accessed 7. Sep. 2024].

Bferrite (2015). *The Latest SQL Injection Trends*. [online] Check Point Blog. Available at: <https://blog.checkpoint.com/latest-sql-injection-trends/>.

[Accessed 6. Sep. 2024].