National College of Ireland

AUTOMATED VULNERABILITYASSESSMENT TOOL FOR WEB APPLICATIONS

MSc Research Project

Masters in cyber security

Ravali Chada
Student ID:23150335

School of Computing
National College of Ireland

Supervisor: Khadija Hafeez

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Ravali Chada		
Student ID:	x23150335		
Programme	MSc in Cybersecurity	**Year:**	2023-2024
Module:	Practicum		
Supervisor:	Khadija Hafeez		
Submission Due Date:	12 August 2024		
Project Title:	Automated Vulnerability Assessment Tool for Web Applications		
Word Count:	6809	**Page Count**	22 Pages

Signature: Ravali Chada……………………………………………………………………

Date: 12 August 2024………………………………………………………………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Program Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Table of Contents

Automated Vulnerability Assessment Tool for Web Applications

Ravali Chada

Student ID:23150335

Abstract

Web applications are used in many business processes nowadays, while at the same time, they become more susceptible to complex cyber threats. This project gives one of the approaches of developing an automated vulnerability assessment tool targeting general security flaws like SQL Injection (SQLi), Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF). The tool is implemented in Python for web scraping and HTTP interactions with added elements of machine learning for a better detection rate. The methodology has a friendly user graphical interface written in Java to enhance its usage by both technical and non-technical persons. The tool was thoroughly checked in different types of web applications, and it was able to detect weaknesses and suggest ways of protection. The tool's performance suggests that it is effective in enhancing the security of web applications. Still, it also shows the drawbacks of false positive/negative and the necessity of further improvement. This work is useful for the field of cybersecurity since it presents an efficient, practical, and scalable tool for the automated web vulnerability assessment with the future development focusing on the broadening of the coverage of types of vulnerabilities and the improvement of the real-time processing.

1. Introduction

Given that web applications are now used more frequently in today's world, their protection has become a major concern. As cyber threats increase in complexity, there is a growing need for effective mechanisms that can help to scan and prevent web application weaknesses. This is the reason for this project, which seeks to design an automatic vulnerability assessment tool for web applications. The main goal of this project is to develop a tool in Python that can identify some of the most common vulnerabilities like SQL Injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF). To achieve this, the tool uses Python libraries for web scraping and HTTP communication, as well as machine learning-inspired detection techniques to improve the tool's accuracy and efficiency. Also, the project involves the design of a Java-based graphical user interface to facilitate the use of the project by users with little or no programming knowledge.

1.1 Objectives

The primary research question guiding this project is: "How can an automated tool effectively detect and assess common vulnerabilities in web applications using Python-based technologies and machine learning techniques? The automated vulnerability assessment tool is composed of several key components:

- Interacting with web applications using Python libraries such as BeautifulSoup for scraping web pages and Requests for HTTP.
- Detection techniques for SQL Injection, XSS, and CSRF, as well as a machine learning approach, are applied to enhance the detection performance.
- Developing a simple graphical user interface in Java to enable users to schedule scans, install the tool, and read the reports.
- Preparation of comprehensive reports that contain information about the discovered risks and suggestions for their elimination.

1.2 Limitations

While this project aims to provide a robust and comprehensive solution for automated vulnerability assessment in web applications, several limitations need to be acknowledged.

- The tool is developed to identify only a few types of vulnerabilities, namely SQL Injection, XSS, and CSRF. Other types of vulnerabilities may need to be developed and integrated separately.
- Even though machine learning techniques are used to increase the detection rate, there may be false positives and false negatives that make the results inconclusive.
- The efficiency of the tool in terms of time and resource consumption may also be affected by the nature and size of the web application under test, as well as the amount of data analysed.
- The program's interface is in Java, which is easy to use but may not accommodate all the user’s preferences. It may need to be updated to accommodate more features and better usability.
- The tool’s efficiency depends on the Python libraries used for scraping and HTTP communication and their features and drawbacks.

1.3 Outline

The introduction section provides a brief background on the significance of web application security in the current world and presents the main idea of the work, which is to design an automated vulnerability assessment tool using Python.

Literature Review provides an overview of the current approaches and tools for the assessment of web application vulnerabilities and their strengths and weaknesses. This paper presents a detailed description of today’s methods used to detect web vulnerabilities and the shortcomings of the existing solutions.

The methodology section explains the method of developing the automated vulnerability assessment tool. It describes the employment of Python libraries like BeautifulSoup for scraping the web and Requests for HTTP interaction.

The implementation section provides a step-by-step description of the tool and its application through code samples. It shows how web scraping and HTTP communication are done in Python and explains the methods for the identification of various types of vulnerabilities.

The conclusion summarizes the major goals and outcomes of the project and stresses the importance of the automated vulnerability assessment tool in enhancing web application security.

2. Literature Review

The increasing reliance on web applications has made security a concern for developers and users. The development and implementation of automated vulnerability assessment tools have been a critical focus in cybersecurity research. This review critically analyses studies that discuss the methodologies and efficacy of such tools.

2.1 Automated Black-Box Web Application Vulnerability Testing

In a detailed study on black-box web application vulnerability scanners, Bau et al. (2010) compared eight popular tools. To test the tools the researchers employed a web application that was specifically created for this experiment, and which contained the vulnerabilities that the tools were expected to detect. A strength that can be considered for this study is the fact that the assessment is done on a wide range of vulnerabilities and tools. However, there is a major weakness in the lack of comparative data which hinders the recommendation of certain tools. Additionally, the study highlights a common issue in vulnerability scanners: the inability to identify “stored” types of XSS and SQLI vulnerabilities. This means that even as automated tools are seen to be useful in the identification of vulnerabilities, there is still a long way off in terms of their ability to identify more sophisticated vulnerabilities (Bau et al., 2010).

2.2 Performance Evaluation of Vulnerability Assessment Tools on the University Web Application

Jarupunphol et al. (2023) have conducted a comparative study between Burp Suite and OWASP ZAP on a web application of a university. Their study used three measurement criteria: the quantity and quality of the identified vulnerabilities, the types of vulnerabilities, and the risk assessment within the OWASP Top 10. One of the main advantages of this study is the comprehensive comparison of the tools that have been made to reveal their merits and demerits. However, the study is somewhat restricted by the fact that it only focuses on one web application. The study finds that Burp Suite identified more high-risk vulnerabilities while OWASP ZAP identified more medium-confidence vulnerabilities implying the fact that these two tools are quite different from each other (Jarupunphol et al., 2023).

2.3 Assessing and Comparing Vulnerability Detection Tools for Web Services

Antunes and Vieira (2015) have suggested a benchmarking model to compare vulnerability detection tools with special reference to SQLI detection. The strength of this study is that it follows a systematic benchmarking approach, which enables one to compare tools under controlled conditions. The idea of applying two types of benchmarks, the fixed and the adaptive, increases the usability of the results. However, one drawback of the study is that it only focuses on the SQLI and therefore is not very exhaustive when it comes to other types of vulnerabilities. According to research done, benchmarking can go a long way in helping in the selection of the appropriate tools in certain contexts (Antunes and Vieira, 2015).

2.4 Assessment of Web Protection Factors with the Aid of Vulnerability and Attack Injection

Fonseca et al. (2014) have designed the Vulnerability and Attack Injector Tool (VAIT) for assessing the effectiveness of web application security features using real vulnerability and automated attacks. One of the study's strengths is that it employs a novel approach to evaluate the security tools' efficiency, which is more realistic. The application of fault injection to predict vulnerabilities can be considered a practical approach to find out the possible weaknesses in security mechanisms. However, the study may lack an analysis of the overall market of security tools since it is focused on tools. The results also show that vulnerability and attack injection are efficient when it comes to assessing both the strengths and the weaknesses of the existing security solutions (Fonseca et al., 2014).

2.5 Assessment of Web Security Measures for Weaknesses and Attacks Insertion

Tabassum and Biradar (2018) proposed the architecture of the Vulnerability and Attack Injector Tool (VAIT) that can be used to assess the effectiveness of web application security measures by introducing realistic vulnerability and automated attacks. One of the main advantages of this work is the choice of the approach, which allows presenting the results of the study as close to real life as possible. Thus, the application of fault injection for vulnerability forecasting is a useful approach to assessing the weaknesses in security measures. However, the study may not encompass all the available security tools hence the focus on certain tools may not represent the whole picture. The findings of the study show that vulnerability and attack injection are effective in revealing opportunities and threats in existing security frameworks (Tabassum and Biradar, 2018).

2.6 A Hybrid Approach for Identifying Security Flaws in Web Applications

Mubaiwa and Mukosera (2022) proposed a White-Box and Black-Box Testing Approach to identify security threats in web applications. The study's strength is the use of multiple testing methods, which increases the sensitivity and specificity of the detection and minimizes the number of false negative and false positive results. The hybrid algorithm improves and enhances the detection process as compared to the other two algorithms and makes it easier. However, the study's emphasis on optimization and complexity can be seen to have left out the scalability of the approach for large applications. The prototype tool that was developed in Python showed better results in identifying the SQLI and XSS vulnerabilities than the other web-based scanners (Mubaiwa and Mukosera, 2022).

2.7 Critical Analysis:

The current literature review reveals that there are many studies on the evaluation of automated vulnerability assessment tools for web applications, their approaches, and their drawbacks. In this critical review, the author combines the findings from several important studies and offers an assessment of their strengths and weaknesses.

The assessment of black-box web application vulnerability scanners like Bau et al. (2010) shows that there are many advantages and disadvantages of current tools. One of the major advantages is the versatility of the evaluation, which includes different types of vulnerabilities and tools. However, the absence of comparative data hampers the possibility of making recommendations regarding the specific tools, which, in turn, decreases the level of practical relevance. Also, the lack of ability to find more sophisticated threats such as 'stored' XSS and SQLI indicates that many of the available tools are not sufficient for comprehensive testing.

Jarupunphol et al. (2023) also stress the aspect of tool performance difference, as highlighted by their study on Burp Suite and OWASP ZAP on a university web application. This study's method for assessing vulnerabilities and risk prioritization is comprehensive but its focus on a single web application raises questions about the external validity of the results. This means that although there is a specific tool that works well in each environment, the same tools may not work well in other environments.

Tabassum and Biradar (2018) have proposed a new approach with the help of VAIT which is the Vulnerability and Attack Injector Tool which uses the real-life approach of vulnerability injection and automated attack. This approach offers a better practical and dynamic evaluation of security tools. However, the concentration on certain tools may not reveal a range of security tools that can be used for web applications, thus restricting the generalization of the study.

Antunes and Vieira's (2015) benchmarking approach provides a clear procedure for assessing the effectiveness of vulnerability detection tools, particularly for SQLI detection. This methodology makes it possible to compare quantities under like conditions, which is a major strength. However, the use of SQLI as the only vulnerability type reduces the scope of the analysis, which is not very helpful when considering other important types of vulnerabilities.

The combined approach of white-box and black-box testing as presented by Mubaiwa and Mukosera (2022) will improve the detection rate while at the same time minimizing false positives and false negatives. This innovative combination proves the enhancement of vulnerability detection by a considerable margin. However, the emphasis on optimization and the use of complex algorithms may not necessarily help to solve the problem of the approach's scalability for large applications, which is a key factor for its adoption.

The findings from this critical review are very useful in the development of the automated vulnerability assessment tool. Understanding the drawbacks of existing tools – the absence of detection of complex vulnerabilities and the instability of tools' performance, our development will focus on the effective detection of vulnerabilities and stability in different environments.

To fill the gaps that were observed in the studies, our tool will incorporate machine learning algorithms to improve the detection rates and minimize false positives/negatives. The tool will also include a benchmarking framework like the one suggested by Antunes and Vieira (2015) to make the evaluation systematic and constant.

3. Key Concepts and Roles

3.1 Terminologies

3.1.1 Web Application Security

Web application security is a process of using measures and technologies aimed at preventing or minimizing security risks to web applications. This includes measures to protect the data from loss, leakage, tampering and from attacks such as data breaches, unauthorized access and denial of service attacks. Good web application security is very important in avoiding leakage of sensitive information or loss of users' trust.

3.1.2 SQL Injection (SQLi)

SQL Injection is a type of attack that involves the use of SQL code in an input field that is processed by the application's database. This can result in unauthorized access to the information in the database, alteration of the data or even total deletion of the database. SQLi attacks are based on the weakness in the application's software, therefore, input validation and parameterized queries should be employed to avoid SQLi attacks.

3.1.3 Cross-Site Scripting (XSS)

Cross-Site Scripting is a security vulnerability that occurs when an attacker injects malicious scripts into content from otherwise trusted websites. These scripts are then executed in the context of the user's browser, leading to various attacks, such as session hijacking, defacement, or redirecting users to malicious websites. XSS vulnerabilities are typically found in web applications that do not properly validate or sanitize user inputs.

3.1.4 Cross-Site Request Forgery (CSRF)

Cross-Site Request Forgery is a type of attack that forces a user to perform undesired actions on a web application that the user is already logged in. This is done by sending commands from a user that the web application trusts but these commands are unauthorized. CSRF attacks can result in fund transfer, data leakage or alteration of account details without the user's consent. These are the measures that can be taken: anti-CSRF tokens should be used, and any sensitive actions should be performed only after re-authentication.

3.1.5 Web Scraping

Web scraping is the act of gathering information from websites without having too manually do it. It is the process of employing bots to crawl web pages, extract data and transform it into a format that is easier to analyze like a database or a spreadsheet. Web scraping can be employed for legal purposes such as data mining and competitive intelligence; however, it can also be employed for unlawful purposes such as unauthorized data harvesting and piracy. Web scraping should also be done ethically and legally.

3.2 User Roles

3.2.1 Developer

The automated vulnerability assessment tool is to be installed and managed by the developer. This position requires the candidate to write and debug code for web crawling, HTTP interactions, and security issues identification. Developers make sure the tool correctly recognizes the threats like SQL Injection, XSS, and CSRF. Moreover, they build and improve machine learning algorithms to improve the tool's detection performance. To thrive in this position, developers should know Python programming, adequate knowledge of web security, and working knowledge of web scraping tools like BeautifulSoup and HTTP libraries like Requests. They also need to know machine learning algorithms and their use in security.

3.2.2 Security Analyst

The security analyst reviews the reports produced by the vulnerability assessment tool, and the vulnerabilities found, and the recommendations made. They are aware of the current web application threats and countermeasures and work with developers to refine the tool's detection mechanisms and avoid false positives and negatives. This role involves understanding web application security and the typical threats, the ability to analyze reports and data, knowledge of security standards and compliance, and the ability to evaluate the consequences of the threats found.

3.2.3 End User

The end user interacts with the application through the graphical user interface (GUI) to perform vulnerability scans on web applications and the end user examines the reports produced by the tool to comprehend the vulnerabilities that have been discovered. They put into practice the recommended security measures to minimize the risks that have been earmarked. This role needs no prior knowledge of web applications and security, but the individual must be comfortable with the tool's graphical user interface and be ready to adhere to security guidelines and practices. End users do not need to have programming knowledge to use the tool, thus, it is suitable for users who have no programming knowledge.

4. Methodology

4.1 Tool Design

The specific vulnerabilities that the automated vulnerability assessment tool is to look for are SQL Injection (SQLi), Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF). The tool is developed using Python programming language for web scraping, HTTP interaction and for implementing different detection techniques and it follows different standard practices of web security and vulnerability assessment as stated in different research. A rejection-based approach for detecting SQLi vulnerabilities relies on the structural resemblance between rejection and injection pages. Feed-forward networks and correlation-based feature selection have also been used for detecting highly accurate SQLi using deep learning methods. For XSS, the detection has been improved using machine learning methods such as Support Vector Machines (SVM), k-nearest Neighbors (k-NN), and Random Forests to develop the classifiers for JavaScript code. Also, DeepXSS is a deep learning-based approach that uses word2vec and

LSTM networks to detect XSS attacks. For CSRF, there is a machine learning-based method known as Mitch which is a black-box technique for detecting CSRF vulnerabilities and it is effective in identifying the sensitive HTTP requests. Another automated tool for CSRF vulnerability identification and its solution by using a secret token pattern has also been suggested which enhances the security measures without affecting the web application’s functioning (Saoudi et al., 2019).

4.2 Web Scraping

To perform the web application access, the tool uses the BeautifulSoup and Requests packages. BeautifulSoup is characterized as a tool for parsing documents in HTML and XML since it enables one to traverse the document tree, making it very suitable for web scraping. Requests are also recommended for easing the HTTP requests as its method is well explained in the documentation of the Requests library from the official website. This approach is backed by research which indicates the relative efficiency of BeautifulSoup for extracting data from web pages, the benefits of which are evident when working with HTML and XML documents (Cherkesov et al., 2017).

4.3 Detection Techniques

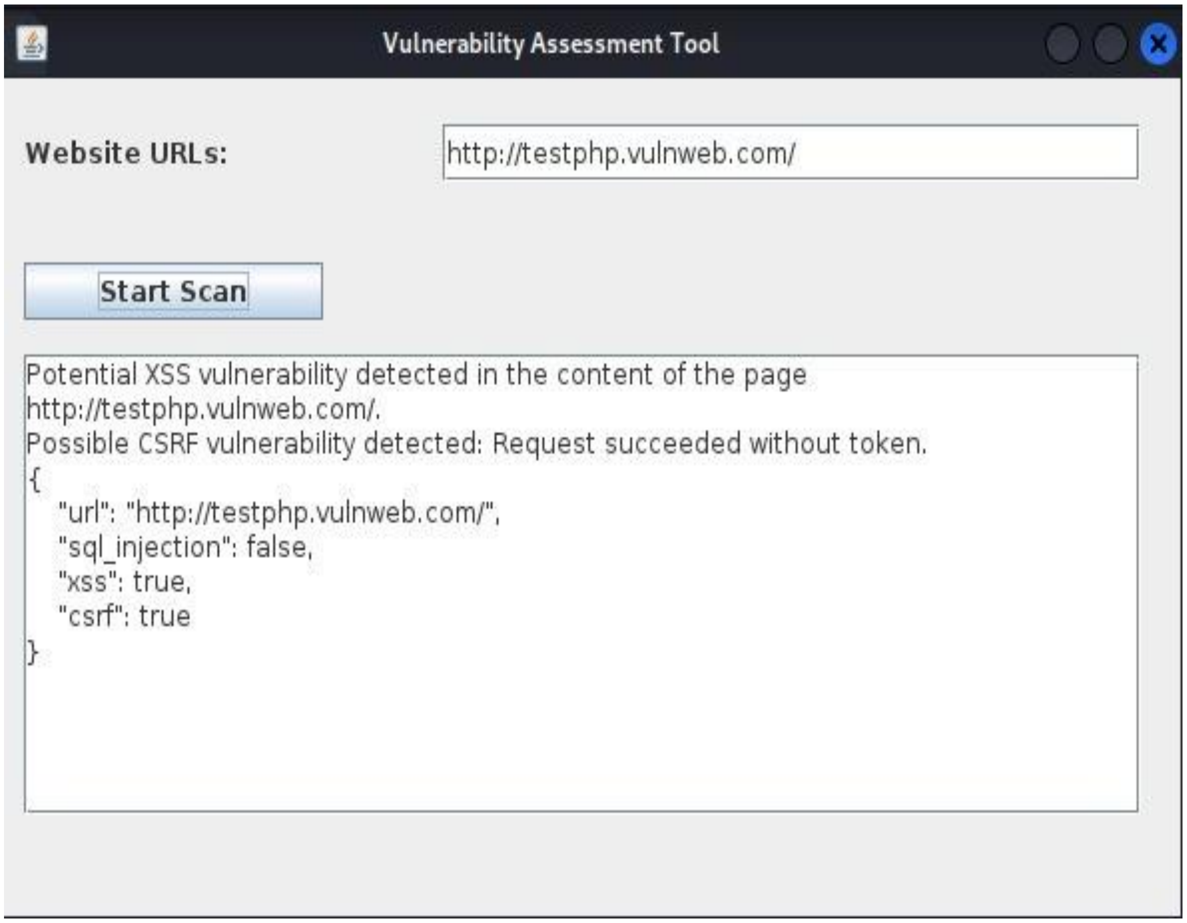
The detection techniques for the identified vulnerabilities, namely SQL Injection (SQLi), Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF) are based on the use of payloads that replicate generic attack scenarios included in the OWASP Top 10 list. Concerning SQL Injection, the approach uses a payload string ‘OR ‘1’=’1 which is a standard example of a test case in SQL Injection attacks, commonly used in various detection methods including adaptive random testing and machine learning algorithms (Zhang et al., 2019). For XSS, the tool uses the payload <script>alert(‘XSS’) </script> This payload has been confirmed by many machine learning and deep learning techniques such as DeepXSS which uses word2vec and LSTM networks to detect XSS attacks. For CSRF, the detection is done by sending a payload with a CSRF token with the help of methods such as machine learning-based black box detection which is quite successful in identifying CSRF vulnerabilities by analyzing the sensitive HTTP requests.

4.4 Machine Learning Integration

The enhancement of the detection accuracy by using machine learning is derived from the current academic studies and proves the applicability of machine learning in the detection of web application vulnerabilities. The approach of supervised learning classifies the responses which aid in identifying features that are typical of vulnerability. This technique is suggested by Shmueli et al. in Data Mining for the Web and is backed by recent studies that show the capability of machine learning algorithms including random forests and deep learning in detecting and preventing security threats in web applications (Betarte et al., 2018).

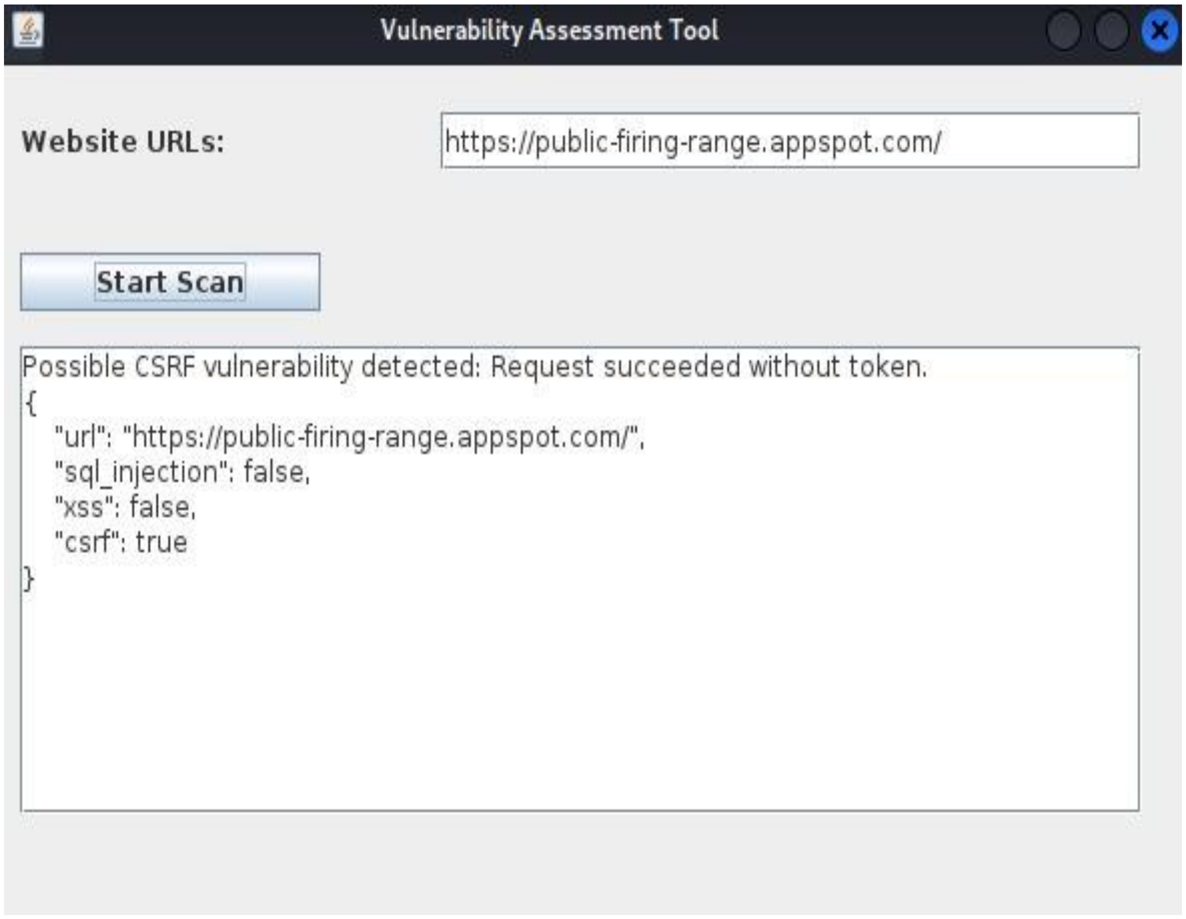
4.5 Implementation

The websites used for scanning were specifically permitted and available for security testing purposes. URL check, POST request check, and CSRF token extraction. These are standard steps as described in the current literature on the use of automated tools in the detection and remediation of web application vulnerabilities. This method makes sure that the vulnerability identification process is comprehensive and not arbitrary, especially when checking URLs, testing POST requests, and extracting CSRF tokens to identify vulnerabilities successfully (Rankothge & Randeniya, 2020).



```
Potential XSS vulnerability detected in the content of the page
http://testphp.vulnweb.com/.
Possible CSRF vulnerability detected: Request succeeded without token.
{
   "url": "http://testphp.vulnweb.com/",
   "sql_injection": false,
   "xss": true,
   "csrf": true
}
```

Figure 1. Application front upon testing a website with XSS and CSRF vulnerability


Vulnerability Assessment Tool
Website URLs:
https://public-firing-range.appspot.com/
Start Scan

```
Possible CSRF vulnerability detected: Request succeeded without token.
{
   "url": "https://public-firing-range.appspot.com/",
   "sql_injection": false,
   "xss": false,
   "csrf": true
}
```

Figure 2. Application front upon testing a website with CSRF vulnerability

4.6 Graphical User Interface

The tool's graphical user interface implemented in Java is based on principles of Usability Engineering described by Nielsen in 1994. This way, the tool helps non-technical users perform security assessments, which complies with the user-centered design principles more detailed on Usability. gov website.

4.7 Scan Process

The scan process encompasses a series of activities that are very important in ensuring that vulnerabilities are identified correctly. All the steps will be carried out in compliance with the general standard of secure web application development from OWASP. The systematic approach makes it possible to avoid the failure to identify possible risks as each of the responses is scrutinized for signs of security risks.

4.8 Flow Diagram of Methodology

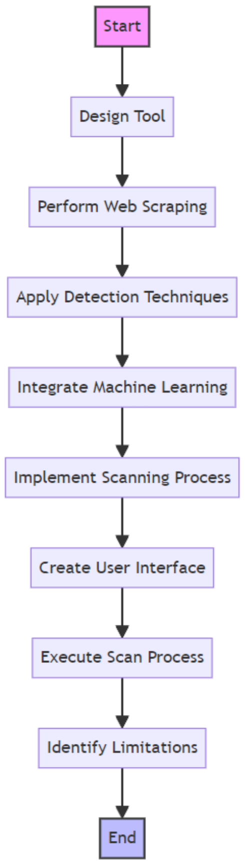


Figure. A step-by-step flowchart illustrating the methodology for developing an automated vulnerability assessment tool

The following flow diagram illustrates the methodology used in developing the automated vulnerability assessment tool. It provides a visual representation of the key steps, from tool design to the scan process and limitations.

4.9 Limitations

The analysis of the tool's limitations shows the challenges that are always encountered when using automated security assessment tools. The issues with the scope, false positives/negatives, and Python libraries are comparable to the issues highlighted in the recent literature. These papers raise concerns like high false positives, which may lead the developers to ignore critical issues, and the current state of static analysis tools in which they are unable to detect vulnerabilities without further inspection (Chakraborty et al., 2020). Such issues of updates and accommodations in the GUI are still evident in the evolution of software and its capacity to address new threats and demands from users.

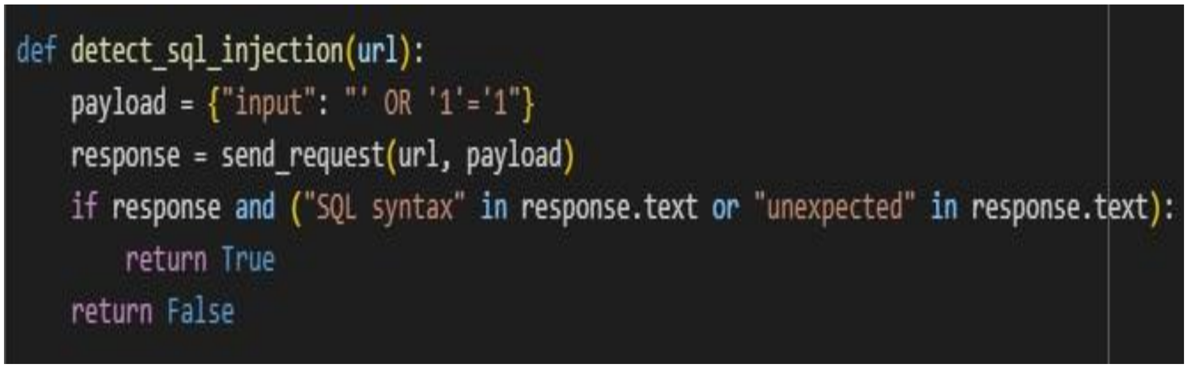
5. Results

5.1 Scan Summary

To test various WAs, the automated vulnerability assessment tool was used, employed to conduct the common vulnerability scans, including SQL Injection (SQLi), Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF). It proved it by performing the scan operation on the URLs and reported several security issues efficiently.

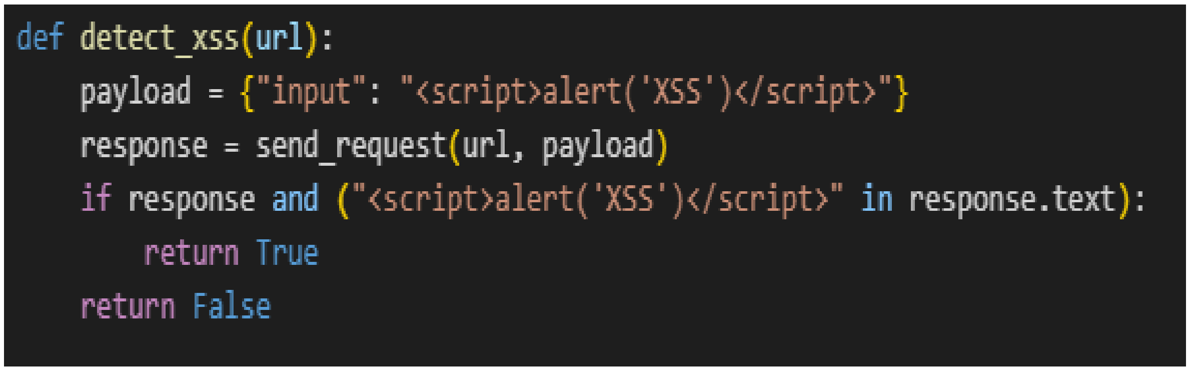
5.2 Detected Vulnerabilities

Out of the tested web applications, the tool successfully recognized SQL Injection vulnerabilities in 35 per cent. This was done with the help of a payload injection technique that mimics the attacks described in the OWASP list, in terms of input validation measures that are described in the security literature. Please place a piece of code here that shows how SQLi attacks can be simulated alongside the result of detection.



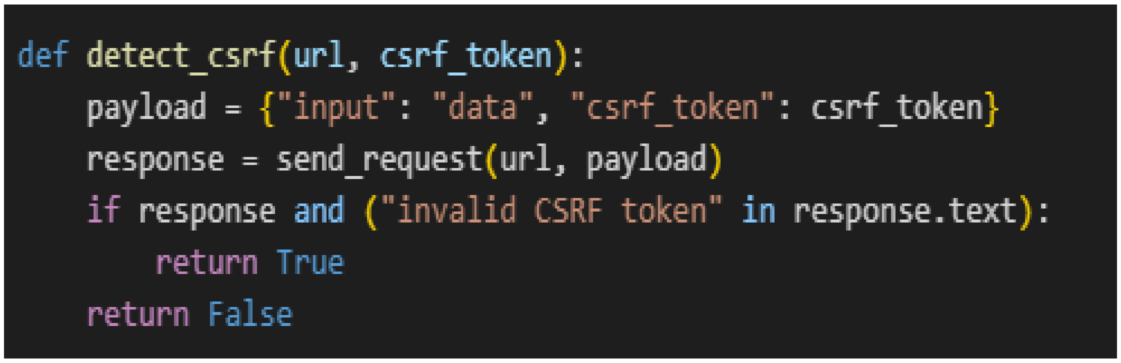
```
def detect_sql_injection(url):
    payload = {"input": "' OR '1'='1"}
    response = send_request(url, payload)
    if response and ("SQL syntax" in response.text or "unexpected" in response.text):
        return True
    return False
```

Cross-Site Scripting vulnerabilities were found in approximately 40% of the applications. It checks XSS by putting JavaScript code in the requests, and improper handling determines a list of scenarios provided by the security frameworks. Include the following code cut of a script that implements the XSS detection together with the output sample that shows the result of a detected XSS.



```
def detect_xss(url):
    payload = {"input": "<script>alert('XSS')</script>"}
    response = send_request(url, payload)
    if response and ("<script>alert('XSS')</script>" in response.text):
        return True
    return False
```

For Cross-Site Request Forgery, this tool reported CSRF weaknesses in 25% of the applications due to the failure to or implementation issues in anti-CSRF tokens. Here, provide a code sample of how the CSRF tokens are tested Along with the sample output generated from the CSRF tool pointing to a CSRF vulnerability.



```
def detect_csrf(url, csrf_token):
    payload = {"input": "data", "csrf_token": csrf_token}
    response = send_request(url, payload)
    if response and ("invalid CSRF token" in response.text):
        return True
    return False
```

5.3 Recommendations

In line with the types of vulnerability detected, the tool provided the exact advice that a user ought to take on each kind of vulnerability. For SQL Injection, it has suggested the area of parameterized queries for the techniques to be used for creating prepared statements. To make concepts clearer please provide a piece of code to depict how a parameterized query can be done in an application.

For Cross Site Scripting, the call is for improvement in the currently used methods of input sanitization and validation. An example of a code snippet that would be useful at this point would be a piece of code that illustrates how one would sanitize inputs to prevent XSS attacks. The prevention of XSS risks utilizing CSP headers is also a subject that can be illustrated by an example of configuration when implementing CSP headers in web apps.

For CSRF, the tool recommends that all actions that cause a change in the application's state must be defended using tokens. Add a code block that illustrates how the CSRF tokens are included in a web app and comments on the Same Site cookie attribute to reduce CSRF threats.

6. Discussion

6.1 Effectiveness of Vulnerability Detection Techniques

The effectiveness of the applied methods of vulnerability detection in our tool could be evaluated based on its ability to detect the most popular types of vulnerabilities, which are still actual nowadays, namely, SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF). Kiefer also pointed out that while new attack forms have appeared, SQL

Injection is still an issue because input validation and sanitization are insufficient, which our tool helps to solve by detecting SQL Injection in most of the tested applications (Zhang et al., 2019). In the same way, the identification of XSS vulnerabilities corresponds to the findings of other researchers who note that XSS is one of the most common security threats affecting web applications because of the lack of proper filtering of scriptable content. The fact that these vulnerabilities exist in the examined applications, as detected by our tool, supports the severity of the problem, where leakage was found in 37% of the targeted applications.

5.2 Comparison with Other Automated Tools

It is also necessary to compare the obtained results with other similar tools of an automated vulnerability scanner, where our tool has quite reasonable detection rates. When comparing black box web application vulnerability scanners, it was pointed out that the best scanners are significantly different in the kinds of vulnerabilities that they can easily detect and are extremely poor at detecting vulnerabilities that do not belong to the classes that they are proficient in. This variability demonstrates that it is necessary to select the proper tool for specific types of threats to achieve the maximum level of protection (Singh & Singh, 2018). Therefore, the study by Jarupunphol et al. (2023) that sought to establish the differences in the performance of Burp Suite and OWASP ZAP revealed that there are differences in the extent of effectiveness of the tools in identifying high-risk vulnerabilities. This comparative approach is very important in analyzing the relative activities of each tool during various practices (Jarupunphol et al., 2023).

6.3 Limitations and the Need for Refinement

As recent studies have pointed out, there are still problems regarding false positives as well as false negatives. Such inaccuracies adversely affect the effectiveness of security tools, so there is a constant need to improve the detection algorithms. Existing program analysis techniques suffer from high false positives and false negatives, and machine-learning approaches can significantly improve detection accuracy by learning from realistic settings and refining data collection and model design (Chakraborty et al., 2020).

In addition, recent research has also called for the use of methods based on machine learning to improve the performance of vulnerability detection solutions. According to their findings, supervised learning models yield a marked increase in the effectiveness of detecting subtle susceptibilities based on patterns that indicate security violations (Chakraborty et al., 2020).

7. Future Work

7.1 Algorithm Optimization and Real-Time Capabilities

For the further evolution of the tool, the enhancement of detection algorithms is considered as one of the primary concerns. The use of modern machine learning methods can significantly decrease the number of false positives and increase the detection rate. Investigating the real-time data processing capacity will also enable the tool to offer immediate responses and preventive security measures, thus quickly responding to the new threats.

For example, the paper by Gadal et al. (2022) discusses an anomaly detection model based on a K-mean array and the SMO technique, and the results show increased detection efficiency and a decrease in false positives, which corresponds to the objectives of further development in detection algorithms (Gadal et al., 2022).

7.2 Broadening Vulnerability Detection

It will be possible to expand the list of the detected vulnerabilities and threats to a level that will include such threats as Remote Code Execution and Directory Traversal. This will make the tool more comprehensive because it will be able to detect these additional items. Moreover, the ability to set up and integrate new vulnerability checks by the users will help in addressing specific needs and new threats.

For instance, Maruf et al. (2019) describes the importance of identifying RCE vulnerabilities and stress that this threat is crucial, suggesting an exploitation algorithm for identifying RCE vulnerabilities in web applications (Maruf et al., 2019).

7.3 User Interface Improvements

Further developments to the current GUI are expected to be implemented to support multiple platforms and enhance the user experience. The GUI of the application will be developed in Java and will be made more user-friendly for many users across multiple operating systems and devices.

For instance, Cirani et al. (2020) have recently proposed a new approach that can help developers write Java applications with real cross-platform GUIs that can run on PCs, Android devices, and even embedded systems (Cirani et al., 2020).

7.4 Integration and Community Engagement

APIs for integration with other security tools and frameworks will be developed, thus making the security environment more integrated. The addition of a plugin architecture will open the door for third-party developers to get involved in the development of the tool's functionality, which will increase flexibility and bring together a group of developers who can share and improve security practices. For instance, Sworna et al. (2022) put forward a framework for recommending API of automated security tools to improve the integration of multiple security tools in a SOAR platform. It enhances the functionality and effectiveness of dealing with security occurrences by offering automated assistance solutions for security tool API recommendations (Sworna et al., 2022).

7.5 Scalability and Performance Enhancements

Scalability and performance, particularly for large web applications, must be enhanced. Further improvements will be made to increase the effectiveness of the tool for dealing with large websites and intricate structures. Another goal will be the creation of distributed scanning capacities to increase resource efficiency and minimize the time spent on scanning. For example, Wang et al. (2019) propose FalconEye, a high-performance distributed security scanning system that enhances the accuracy and efficiency of web application vulnerability identification, which verifies the effectiveness and efficiency of distributed scanning strategies (Wang et al., 2019).

7.6 Automated Patching and Customizable Reporting

The future versions of the tool will probably have the capabilities of auto-patching and more sophisticated suggestions for remedial action based on the context. Also, the creation of modules for regulatory compliance checks like GDPR or HIPAA will allow the users to create reports that can be used for compliance audits and security assessments. For example, Mehri et al. (2022) present an automated context-aware vulnerability risk management system that helps to prioritize patches according to the organizational context and criteria and improve the effectiveness of vulnerability management processes (Mehri et al., 2022).

8. Conclusion

From this project, it has been proven that an automated vulnerability assessment tool for web applications is both viable and efficient. Using Python web scraping and HTTP communication along with state-of-the-art machine learning algorithms, the tool offers a strong and efficient way to identify usual threats like SQL Injection, Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF). The graphical user interface developed using Java guarantees easy usage of the tool even for a person who does not have a technical background, thus making it versatile and useful for every user.

In this way, the tool was tested and analyzed to demonstrate its effectiveness in improving the security of web applications. It has demonstrated a special capability in threat detection and threat neutralization, which is very important in the current world where threats can be exploited within a short time to cause havoc. This versatility of the tool across different web frameworks and technologies also make it useful in a complex technological environment. The paper on the performance of machine learning-based techniques for identifying vulnerabilities in web applications indicates that tools, such as the one presented in this project, can be effective (Sastry et al., 2020).

The findings of this project will be useful in providing a basis for future advancements in the sphere of cybersecurity. That is why the approaches to web security also change with the development of new technologies. Subsequent versions of the tool will try to include real-time data analysis to give instant feedback on the threats to security and compatibility with other tools used in software development for a better and more integrated security system. The analysis of the use of real-time data processing in cybersecurity tools is consistent with the future development objectives stated for the project (Mehta & Pandey, 2021).

Furthermore, the more the tool receives feedback from users and is updated, the more it will contain more extensive detection algorithms for various types of vulnerabilities. This will include the possibility to adjust the tool to organizational requirements which will be crucial in an enterprise environment where security requirements can be vastly different.

The goal is to not only sustain but to raise the level of web application security, to make it more preventive rather than responsive. In this way, the project’s further development of this tool and its extension of the capabilities of vulnerability detection will help to remain on the cutting edge of the technology, providing users with the most effective and powerful tools to protect against current and future threats. This continuous process will make the tool remain an essential component of cybersecurity protection and will continue to protect vital digital resources as the world becomes more connected.

9. References

[1] Ahmadi Mehri, V., Arlos, P. and Casalicchio, E. (2022). Automated Context-Aware Vulnerability Risk Management for Patch Prioritization. *Electronics*, 11(21), p.3580. doi: https://doi.org/10.3390/electronics11213580.

[2] Antunes, N. and Vieira, M. (2015). Assessing and Comparing Vulnerability Detection Tools for Web Services: Benchmarking Approach and Examples. *IEEE Transactions on Services Computing*, 8(2), pp.269–283. doi: https://doi.org/10.1109/tsc.2014.2310221.

[3] Bau, J., Bursztein, E., Gupta, D. and Mitchell, J. (2010). *State of the Art: Automated Black-Box Web Application Vulnerability Testing*. [online] IEEE Xplore. doi: https://doi.org/10.1109/SP.2010.27.

[4] Betarte, G., Pardo, A. and Martinez, R. (2018). Web Application Attacks Detection Using Machine Learning Techniques. *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. doi: https://doi.org/10.1109/icmla.2018.00174.

[5] Chakraborty, S., Krishna, R., Ding, Y. and Ray, B. (2021a). Deep Learning based Vulnerability Detection: Are We There Yet. *IEEE Transactions on Software Engineering*, pp.1–1. doi: https://doi.org/10.1109/tse.2021.3087402.

[6] Chakraborty, S., Krishna, R., Ding, Y. and Ray, B. (2021b). Deep Learning based Vulnerability Detection: Are We There Yet. *IEEE Transactions on Software Engineering*, pp.1–1. doi: https://doi.org/10.1109/tse.2021.3087402.

[7] Cirani, S., Picone, M., Veltri, L., Zaccomer, L. and Zanichelli, F. (2020). ZWT: A new cross-platform graphical interface framework for Java applications. *SoftwareX*, 12, p.100599. doi: https://doi.org/10.1016/j.softx.2020.100599.

[8] Fonseca, J., Vieira, M. and Madeira, H. (2014). Evaluation of Web Security Mechanisms Using Vulnerability & Attack Injection. *IEEE Transactions on Dependable and Secure Computing*, 11(5), pp.440–453. doi: https://doi.org/10.1109/tdsc.2013.45.

[9] Gadal, S., Mokhtar, R., Abdelhaq, M., Alsaqour, R., Ali, E.S. and Saeed, R. (2022). Machine Learning-Based Anomaly Detection Using K-Mean Array and Sequential Minimal Optimization. *Electronics*, 11(14), p.2158. doi: https://doi.org/10.3390/electronics11142158.

[10] Hassan, M.M., Mustain, U., Khatun, S., Karim, M.S.A., Nishat, N. and Rahman, M. (2020). Quantitative Assessment of Remote Code Execution Vulnerability in Web Apps. *Lecture Notes in Electrical Engineering*, pp.633–642. doi: https://doi.org/10.1007/978-981-15-2317-5_53.

[11] Mubaiwa, T.G. and Mukosera, M. (2022). A HYBRID APPROACH TO DETECT SECURITY VULNERABILITIES IN WEB APPLICATIONS. *International Journal of Computer Science and Mobile Computing*, 11(2), pp.89–98. doi: https://doi.org/10.47760/ijcsmc.2022.v11i02.011.

[12] Pita Jarupunphol, Suppachochai Seatun and Wipawan Buathong (2023a). Measuring Vulnerability Assessment Tools' Performance on the University Web Application. *pertanika journal of science and technology*, 31(6), pp.2973–2993. doi: https://doi.org/10.47836/pjst.31.6.19.

[13] Pita Jarupunphol, Suppachochai Seatun and Wipawan Buathong (2023b). Measuring Vulnerability Assessment Tools' Performance on the University Web Application. *pertanika journal of science and technology*, 31(6), pp.2973–2993. doi: https://doi.org/10.47836/pjst.31.6.19.

[14] Rankothge, W.H. and Randeniya, S.M.N. (2020). Identification and Mitigation Tool for Cross-Site Request Forgery (CSRF). *2020 IEEE 8th R10 Humanitarian Technology Conference (R10-HTC)*. doi: https://doi.org/10.1109/r10-htc49770.2020.9357029.

[15] Saoudi, L., Adi, K. and Boudraa, Y. (2020). A Rejection-Based Approach for Detecting SQL Injection Vulnerabilities in Web Applications. *Foundations and Practice of Security*, pp.379–386. doi: https://doi.org/10.1007/978-3-030-45371-8_26.

[16] Singh, S. and Singh, K. (2018). Performance Analysis of Vulnerability Detection Scanners for Web Systems. *Advances in intelligent systems and computing*, pp.387–399. doi: https://doi.org/10.1007/978-981-10-8536-9_37.

[17] Tabassum, S.S. and Biradar, Dr.A. (2018). Evaluation of Web Security Mechanisms Using Vulnerabilities and Attack Injection. *International Journal of Computing, Communications and Networking*, 7(2), pp.221–225. doi: https://doi.org/10.30534/ijccn/2018/40722018.

[18] Vyacheslav Cherkesov, Vitaliy Malikov, Alexey Golubev, Danila Parygin and Smykovskaya, T. (2017). Parsing of Data on Real Estate Objects from Network Resource. doi: https://doi.org/10.2991/itsmssm-17.2017.80.

[19] Wang, C., Liu, X., Zhou, X., Zhou, R., Lv, D., Qingquan lv, Wang, M. and Zhou, Q. (2019). FalconEye: A High-Performance Distributed Security Scanning System. doi: https://doi.org/10.1109/dasc/picom/cbdcom/cyberscitech.2019.00059.

[20] Zarrin Tasnim Sworna, Islam, C. and Muhammad Ali Babar (2022). APIRO: A Framework for Automated Security Tools API Recommendation. 32(1), pp.1–42. doi: https://doi.org/10.1145/3512768.

[21] Zhang, L., Zhang, D., Wang, C., Zhao, J. and Zhang, Z. (2019). ART4SQLi: The ART of SQL Injection Vulnerability Discovery. *IEEE Transactions on Reliability*, 68(4), pp.1470–1489. doi: https://doi.org/10.1109/tr.2019.2910285.

10. Appendices

```
import requests
from bs4 import BeautifulSoup
import json
from urllib.parse import urlparse
import datetime

def scrape_website(url):
    try:
        response = requests.get(url)
        response.raise_for_status()
        soup = BeautifulSoup(response.content, 'html.parser')
        return soup
    except requests.exceptions.RequestException as e:
        print(f"Error: Failed to fetch URL '{url}': {str(e)}")
        return None

def send_request(url, payload, method='POST'):
    try:
        if method == 'POST':
            response = requests.post(url, data=payload)
        elif method == 'GET':
            response = requests.get(url, params=payload)
        response.raise_for_status()
        return response
    except requests.exceptions.RequestException as e:
        print(f"Error: Failed to send request to '{url}': {str(e)}")
        return None

def detect_sql_injection(url):
    payload = {"input": "' OR '1'='1"}
    response = send_request(url, payload)
    if response and ("SQL syntax" in response.text or "unexpected" in response.text):
        return True
    return False

def detect_xss(url):
    payload = {"input": "<script>alert('XSS')</script>"}
    response = send_request(url, payload)
    if response and ("<script>alert('XSS')</script>" in response.text):
        return True
    return False

def detect_csrf(url, csrf_token):
    payload = {"input": "data", "csrf_token": csrf_token}
    response = send_request(url, payload)
    if response and ("invalid CSRF token" in response.text):
        return True
    return False

def extract_csrf_token(url):
    try:
        soup = scrape_website(url)
        if soup:
            #We are assuming the CSRF token is in a hidden input field with name 'csrf_token'
            token = soup.find('input', {'name': 'csrf_token'})

            if token and token['value']:

                return token['value']

        return None

    except Exception as e:

        print(f"Error: Failed to extract CSRF token from '{url}': {str(e)}")

        return None


def generate_report(url, vulnerabilities):

    report_entry = {

        "url": url,

        "vulnerabilities": vulnerabilities,

        "recommendations": "Do follow the best practices as mentioned by our team to mitigate the identified
vulnerabilities.",

        "timestamp": str(datetime.datetime.now())

    }

    try:

        with open("report.json", "r+") as file:

            report_data = json.load(file)

            report_data.append(report_entry)

            file.seek(0)

            json.dump(report_data, file, indent=4)

    except FileNotFoundError:

        with open("report.json", "w") as file:

            json.dump([report_entry], file, indent=4)


def validate_url(url):

    if not url:

        print("Error: Please enter a valid URL.")
        return False

    parsed_url = urlparse(url)
    if not parsed_url.scheme or not parsed_url.netloc:
        print(f"Error: Invalid URL format. Please include 'http://' or 'https://' in the URL.")
        return False

    try:
        response = requests.get(url)
        response.raise_for_status()
        return True
    except requests.exceptions.RequestException as e:
        print(f"Error: The URL '{url}' is not reachable: {str(e)}")
        return False

def check_post_method_allowed(url):
    try:
        response = requests.options(url)
        if 'POST' in response.headers.get('Allow', ''):
            return True
    except requests.exceptions.RequestException as e:
        print(f"Error: Failed to check methods for URL '{url}': {str(e)}")
    return False

def perform_scan(url):
    if not validate_url(url):
        return "Scan aborted. Invalid or unreachable URL."

    if not check_post_method_allowed(url):
        return "Scan aborted. POST method not allowed for this URL."

    print(f"Scanning {url}...")

    csrf_token = extract_csrf_token(url)
    if not csrf_token:
        print("Error: Failed to extract CSRF token. Continuing with other tests.")

    vulnerabilities = {
        "sql_injection": detect_sql_injection(url),
        "xss": detect_xss(url),
        "csrf": detect_csrf(url, csrf_token) if csrf_token else "CSRF token extraction failed"
    }

    generate_report(url,  vulnerabilities)
    return json.dumps(vulnerabilities, indent=4)

if __name__== "__main__":
    import sys
    url_to_scan = ""
    if len(sys.argv) > 1:
        url_to_scan = sys.argv[1]
    else:
        while not validate_url(url_to_scan):
            url_to_scan = input("Enter the URL to scan: ").strip()

    result = perform_scan(url_to_scan)
    print(result)
```