A Novel Tool for Prevention of SQL Injection and XSS Attacks

MSc Research Project
MSc Cybersecurity

Vaibhav Ramdas Vhatkar
21194149

School of Computing
National College of Ireland

Supervisor: Prof. Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Vaibhav Ramdas Vhatkar		
Student ID:	21194149		
Program:	MSc Cybersecurity	**Year:**	2022-23
Module:	MSc Research Project		
Supervisor:	Prof. Vikas Sahni		
Submission Due Date:	14th Aug 2023		
Project Title:	A Novel Tool for prevention of SQL Injection and XSS attacks		
Word Count:	5244 words	**Page Count**	21 pages

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Vaibhav Ramdas Vhatkar

Date: 13th Aug 2023

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 a computer.	□

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

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

A Novel Tool for Prevention of SQL Injection and XSS Attacks

Vaibhav Ramdas Vhatkar
21194149

Abstract

SQL injection is a critical vulnerability that allows attackers to manipulate database queries. This can lead to unauthorized access, data breaches, and other serious consequences. SQLI affects a variety of industries, including finance and healthcare. Since 2017, SQL injection has been in the Top 10 web security risks on OWASP[1]. SQLI has the potential to cause significant data loss, financial losses, and reputation damage. It is therefore a paramount concern for organizations of all sizes. Secure coding practices and regular security assessments are essential for mitigating the risk of SQLI attacks. This work of the developed application can be seamlessly integrated into existing web applications, making it an easy way to improve their security posture. The application is a valuable tool for protecting user data and preventing cyberattacks.

1 Introduction

Web applications are increasingly relied upon in a variety of domains and one of the most prevalent and concerning security threats facing web applications is SQL injection (SQLi) and Cross Site scripting (XSS) attacks. SQL injection and XSS attacks exploit vulnerabilities in web application inputs to execute unauthorized SQL queries, potentially leading to data breaches and system compromise. As cyber attackers continuously evolve their techniques (Gandotra, Bansal and Sofat, 2015), it is imperative to develop effective countermeasures to safeguard user data and maintain the integrity of web applications. In 2017, Equifax suffered a massive data breach due to an SQLI attack, exposing the sensitive data of 143 million people. The final financial loss was estimated to be well above $1.7 billion[2].

This research project presents a web application designed(571b and Survey Paper, 2012) to bolster the security of online systems by detecting SQL injection attacks and implementing IP blacklisting measures. The application leverages PHP and Python to provide a robust and user-friendly platform (Zborowski and Pańczyk, 2023) that empowers users to identify potential SQL injection vulnerabilities within their applications. Using regular expressions (Li *et al.,* 2008), the application proactively identifies and thwarts SQL injection attempts, minimizing the risk of unauthorized data access.

The primary objectives of this research project are twofold:

[1] https://owasp.org/www-project-top-ten/

[2] https://www.housingwire.com/articles/equifax-expects-to-pay-out-another-100-million-for-data-breach/

- To evaluate the effectiveness of the application in detecting and mitigating SQL injection attacks.
- To assess the impact of IP blacklisting as a defence mechanism against repeat offenders.

By investigating these research objectives, this study aims to contribute to the advancement of web application security and strengthen the overall resilience of online systems against emerging cyber threats.

Specific contributions of the research presented in this report are as follows:

- It introduced a web application designed to detect SQL injection attacks and implement IP blacklisting measures.
- It evaluated the effectiveness of the application in detecting and mitigating SQL injection and XSS attacks.
- It assessed the impact of IP blacklisting as a defence mechanism against repeat offenders.
- It provides recommendations for improving the security of web applications against SQL injection and XSS attacks.

Significance of the research:

The research presented in this report is significant because it addresses a critical security challenge for SQL injection and XSS attacks facing web applications. It provides a valuable tool for detecting and mitigating SQL injection and XSS attacks, and it can help to improve the security of web applications.

The research also provides valuable insights into the effectiveness of IP blacklisting (Sreedhar *et al.*, 2012) as a defence mechanism against repeat offenders. IP blacklisting can be an effective way to prevent attackers from repeatedly exploiting vulnerabilities in web applications. However, the research also found that IP blacklisting is not a silver bullet, and it should be used in conjunction with other security measures.

2 Related Work

2.1 SQL Injection

SQL injection has been a critical security threat and a comprehensive review of SQL injection attacks(Halfond, Viegas and Orso, 2006) and research was carried out by Halfond, Vieges, and Orso. They have identified the different types of SQL injection attacks and evaluated the effectiveness of existing techniques in detecting and preventing them. They also analyzed the mechanisms through which SQL injection attacks can be introduced into an application and identify which techniques are best suited to handle each mechanism. The paper further evaluated the deployment requirements of each technique and assessed the extent to which detection and prevention mechanisms could be automated. The difference between prevention-focused and general detection and prevention techniques and explained the difference in their effectiveness. The paper provided recommendations for web application developers to better protect their databases from SQL injection vulnerabilities.

The paper by Byod and Keromytis proposed a practical protection mechanism against SQL injection attacks by applying instruction-set randomization(Boyd and Keromytis, 2004) to SQL. The technique can be easily retrofitted to existing systems and imposes negligible performance overhead. However, the performance evaluation was conducted on a sample database, and the results may not be generalizable to other configurations or environments. The authors acknowledged that modifying the database's interpreter to accept the new set of keywords was a daunting task, which may have limited the practicality of their proposed technique in some cases.

In another paper proposing a Static Analysis Framework for Detecting SQL Injection Vulnerabilities in Web applications(Fu *et al.*, 2007), the framework used symbolic execution and a hybrid constraint solver to identify more sophisticated SQL injection attacks than traditional black-box Web security inspection tools. The paper presented a motivating example of a non-trivial SIA vulnerability that cannot be easily detected by black box testing tools. The proposed approach has the potential to improve the security of Web applications by identifying vulnerabilities that may have been missed by traditional inspected tools.

The research on an automated approach for identifying SQL injection vulnerabilities using a prototype tool called *SQLInjectionGen* (MeiJunjin, 2009) was conducted. The tool combined static analysis, runtime detection, and automatic testing to facilitate the identification of true input manipulation vulnerabilities. The paper presented case studies performed on two small web applications to evaluate the effectiveness of *SQLInjectionGen*. The results showed that the tool is effective in identifying SQL injection vulnerabilities and can be used to complement existing security testing techniques. However, the paper acknowledged the limitations of the evaluation and suggests future work to evaluate larger programs.

To study more about the attacks a paper by Jai Puneet Singh provided an extensive review of advanced SQL Injection attacks (Singh, 2017) , such as Fast Flux SQL Injection, Compounded SQL Injection, and Deep Blind SQL Injection. The paper analysed various detection and prevention techniques, including static code analysis, machine learning, and runtime monitoring. The evaluated different tools for detection and prevention are also discussed. The paper highlighted the challenges faced in researching these types of attacks and emphasizes the importance of proper coding and knowledge of these attacks to prevent their implications on web applications and businesses.

2.2 Cross-Site Scripting (XSS)

For detecting and preventing cross-site scripting (XSS) vulnerabilities in web applications research was made by Lwin Khin Shar and Hee Beng Kuan Tan to discuss different approaches like static string analysis and code-based input validation testing(Shar and Tan, 2012). The paper also highlighted the need for simpler and more flexible security defences, as well as the integration of program analysis, pattern recognition, and AI algorithms to enhance vulnerability detection.

Another study about the evolution of cross-site scripting (XSS) attacks and their potential dangers was conducted. It effectively explained the process of XSS attacks, including the insertion of malicious code and social engineering techniques(Endler, 2002). The paper also highlighted the possibility of automated techniques for session hijacking, which could have made these attacks more prevalent. Additionally, it offered practical solutions and workarounds for web application developers and users to protect against XSS vulnerabilities.

Furthermore, the detection and prevention of cross-site scripting (XSS) attacks in web applications were explored. It discussed various types of XSS attacks and explored different approaches to mitigate them, including bounded model checking, software testing techniques, and browser-enforced embedded policies(Pandian, Nithya and Malarvizhi, 2015). The paper also highlighted the limitations of existing solutions and emphasizes the need for more effective and practical methods to secure web applications against XSS vulnerabilities.

A similar research paper by Philipp Vogt and Nentwich was studied for a combination of dynamic and static analysis techniques. They proposed an anomaly-based intrusion detection system(Vogt *et al.*, no date) that analyzed web server logs and compared incoming user requests to typical parameter profiles. They also introduced a static analysis approach for web applications to detect XSS vulnerabilities. The paper highlighted the importance of using a hybrid analysis approach for precision and efficiency. They implemented their techniques in the Firefox web browser and conducted a large-scale evaluation and demonstrated reliable protection against XSS attacks.

A different research analysis of a machine learning-based approach to detect blind cross-site scripting (XSS) ('Detecting Blind Cross-Site Scripting Attacks Using Machine Learning', 2018) attacks was conducted. The proposed approach uses a linear Support Vector Machine (SVM) classifier to identify malicious payloads likely to be stored in databases through web applications. Experimental results show the feasibility of the proposed approach in detecting blind XSS attacks.

2.3 Web Application Firewall (WAF)

Web application firewalls (WAF) are used in penetration testing and security assessment. A comparative analysis of various web application firewalls (WAFs) using a comprehensive evaluation framework (Ghanbari *et al.*, 2016). The WAFs studied were ModSecurity, NAXSI, PHP-IDS, Applicure dotDefender, Barracuda WAF, F5 BIG-IP Application Security Manager, Imperva SecureSphere WAF and Trustwave WAF. The paper evaluated the effectiveness of each WAF in terms of security features, performance, and usability. It also conducted penetration testing and vulnerability assessment to identify the strengths and weaknesses of each WAF. The results showed that no single WAF is perfect, and each has its strengths and weaknesses.

A similar study of comparison of different WAF solutions and detailed analysis of their features, limitations, and effectiveness (Razzaq *et al.*, 2013) in preventing web application attacks was conducted. The paper highlighted the importance of web application security and the vulnerabilities that exist in current technologies. It compared different WAF solutions and provided a detailed analysis of their features, limitations, and effectiveness in preventing web application attacks. The WAFs compared in this paper are F5, Barracuda, SecurSphere, and WebDefend along with ModSecurity. The paper aims to help users select the most suitable WAF solution for their environments by providing a comprehensive overview of the available options. Overall, the paper provided valuable insights into the current state of web application security and the importance of deploying an effective WAF solution.

Apart from comparisons of the existing firewalls, a new approach was studied proposing a hybrid web application firewall (Tekerek, Gemci and Bay, 2014) that combined signature-based detection and anomaly detection methods to prevent web-based attacks. The paper highlighted the limitations of traditional network layer firewalls and the need for more sophisticated methods to secure web applications. They provided a detailed analysis of the proposed system, including the use of character frequency values, request length, and load distribution to detect anomalies. The study concluded that the hybrid system is effective in removing the deficiencies of the two methods and can prevent zero-day attacks.

Another novel approach towards securing WAFs was a self-healing web application firewall called TokDoc (Krueger *et al.*, 2010), which used local, anomaly-based models to detect and prevent web-based attacks. The paper introduced a data-driven setup procedure that automatically assigned data types to extracted tokens based on structural and statistical features. The paper evaluated the detection performance and runtime of TokDoc using real HTTP traffic and demonstrated its robustness against contaminated data. Overall, the paper presented a promising approach to web application security that differed from traditional signature-based detection techniques.

Finally, the WAF-A-MoLE (Demetrio et al., 2020) is is a tool that used adversarial machine learning to evade web application firewalls (WAFs). The authors developed a set of syntactical mutations to produce adversarial examples against WAFs and evaluated the performance of various machine learning classifiers against these examples. They found that WAF-A-MoLE was effective at bypassing WAFs and outperformed other classifiers in terms of accuracy, recall, and precision. However, the authors noted that WAF-A-MoLE required a guided approach and might not be effective against more sophisticated WAFs.

3 Research Methodology

SQL injection is a serious web security vulnerability that allows attackers to inject malicious code into database queries. This can lead to the exposure of sensitive data, the modification of records, or even the complete control of the application. The 2015 TalkTalk breach[3] is a prime example of the damage that SQL injection can cause.

Cross-site scripting (XSS) is a web security vulnerability that allows attackers to inject malicious code into websites. This code can then be executed in the victim's browser, leading to a variety of problems, such as data theft, session hijacking, and malware distribution. XSS is a serious threat to web security and user data. Organizations need to implement preventive measures such as input validation, output encoding, and security headers to protect themselves from XSS attacks. For example, in 2018, British Airways[4] was breached with an XSS attack that allowed attackers to steal the session cookies of thousands of users.

A comprehensive solution was developed to mitigate the risks of SQL injection and XSS attacks. The solution involves detecting and preventing these attacks, and then integrating a web application firewall (WAF) to enhance security measures further. This innovative approach ensures the identification and prevention of SQL injection and XSS attacks, offering an advanced safeguarding mechanism for web applications.

The patterns of the attacks were studied as discussed below:

SQL injection basic patterns inserted in the script of the tool (Shar and Tan, 2013):

[3] https://www.comparitech.com/blog/information-security/cross-site-scripting
[4] https://www.bbc.com/news/technology-45446529

- ***(union|select|from|where|and|or)*** **and** ***(select|update|insert|delete)***
  These are set of SQL keywords which if entered as a combination, would be marked as potentially harmful.
- ***(and|or)*** **and** ***(=|>|<|>=|<=|<>|!=)***
  These are set of Logical operators followed by numeric comparisons which can be used to manipulate SQL queries.
- ***(and|or)*** **and** ***(=|>|<|>=|<=|<>|!=)***
  These detect ‘and’ or ‘or’ followed by non-whitespace characters, then a comparison operator, and more non-whitespace characters. This pattern captures variations in whitespace.
- ***“exec”***
  This pattern identifies the presence of the ‘exec’ keyword followed by parentheses. The exec command is often exploited to execute arbitrary code.
- ***“create”***
  This pattern identifies the presence of the ‘create’ keyword followed by parentheses. The ‘create’ command is often used to create malicious objects or modify the database structure.
- ***“drop”***
  This pattern identifies the presence of the ‘drop’ keyword followed by parentheses. The ‘drop’ command is often exploited to delete database objects.
- ***“delete”***
  This pattern identifies the presence of the ‘delete’ keyword followed by parentheses. The ‘delete’ command is often exploited to delete rows in database tables.
- ***“insert”***
  This pattern identifies the presence of the ‘insert’ keyword followed by parentheses. The ‘insert’ command is used to insert commands into database tables.
- ***“update”***
  This pattern identifies the presence of the ‘update’ keyword followed by parentheses. The ‘update’ command is used to modify existing data in the database.

XSS basic patterns are inserted in the script of the tool (Bozic and Wotawa, 2013)

- <script[\s\S]*?>
  This pattern detects the presence of the “<script>” tag in HTML, which can be used to include malicious JavaScript code on a webpage.

- r"on\w+\s*=\s*['\"].*['\"]"
  This pattern identifies event handlers like “onclick”, “onload”, etc., followed by an equal sign and a value wrapped in single or double quotes. Attackers may exploit these attributes to execute JavaScript code when the event is triggered.
- r"<img\s+src\s*=\s*['\"]\s*javascript\s*:\s*[^'\"\s]+":
  This pattern detects the use of the “<img>” tag with a javascript: URL in the “src” attribute. This can be abused to execute JavaScript in certain cases.

- ***r"<a\s+href\s*=\s*['\"]\s*javascript\s*:\s*[^'\"\s]+":***
  Similar, to the previous pattern, this identifies the use of the “<a>” tag with a javascript: URL in the “href” attribute.

- ***r"<iframe\s+src\s*=\s*['\"]\s*javascript\s*:\s*[^'\"\s]+":***
  This pattern detects the “<iframe>” tag with a javascript: URL in the “src” attribute. Attackers can use “iframes” to load malicious content.

- ***r"<object\s+data\s*=\s*['\"]\s*javascript\s*:\s*[^'\"\s]+":***
  Similar to the previous patterns, this detects the “<object>” tag with a javascript: URL in the data attribute.

- ***r"expression\s*\(":***
  This pattern identifies the “expression()” function, which is used in old versions of Internet Explorer for CSS. It can be used to execute JavaScript code in some contexts.

- ***r"url\s*\(\s*['\"]\s*javascript\s*:\s*[^'\"\s]+":***
  This pattern detects the “url()” function in CSS with a javascript: URL. Attackers can exploit this to inject JavaScript code into a webpage's CSS.

- ***r"\beval\s*\(":***
  This pattern identifies the use of the “eval()” function, which is commonly used to dynamically execute code. Attackers can use it to execute arbitrary code on the page.

- ***r"\balert\s*\(":***
  This pattern detects the use of the “alert()” function, which displays pop-up alerts in JavaScript. It's often used in XSS payloads to test a vulnerability.

4 Design Specification

The design specification for the tool outlines the use of PHP, Python, and WAF. The requirements focus on performance, user experience, security, scalability, and maintenance to ensure the tool's effectiveness and long-term viability.

Techniques and Architecture:

The web application security tool utilizes a combination of techniques, architecture, and frameworks to provide robust protection against SQL injection and Cross-Site Scripting (XSS) attacks:

- **Front-End**:
  Language: PHP
  Framework: Basic PHP for frontend interaction, form input handling, and user interface.
  User Input Validation: Frontend validation for basic input sanity checks.

- **Back-End**:
  Language: Python
  Framework: Custom Python script for advanced pattern detection and attack prevention.
  Pattern Detection: Manually entered attack patterns for identifying SQL injection and XSS patterns.
  Logging: Python's logging module for capturing attack details.
  Blacklisting: Logic to manage IP blacklisting based on attack frequency.

- **Security Enhancements**:
  Web Application Firewall (WAF): Integration of WAF rules for real-time detection and blocking of suspicious requests.

Associated Requirements:

- **Performance**:
  Speed: Pattern detection is efficient and quick to ensure minimal impact on user experience.
  Resource Usage: The application is light and optimized to prevent excessive resource utilization during pattern matching.

- **User Experience**:
  User Interface (UI): The UI is user-friendly, indicating the status of input safety and providing meaningful feedback.
  Minimal False Positives: The tool avoids flagging legitimate user input as malicious to ensure accurate results.

- **Security**:
  Pattern Accuracy: The attack patterns are accurately identified as malicious patterns while minimizing false negatives.
  IP Blacklisting: Effective handling of IP blacklisting to prevent repeat attacks from the same source.

- **Scalability**:
  Scaling Consideration: Architecture is designed to accommodate future scalability needs as the user base grows.

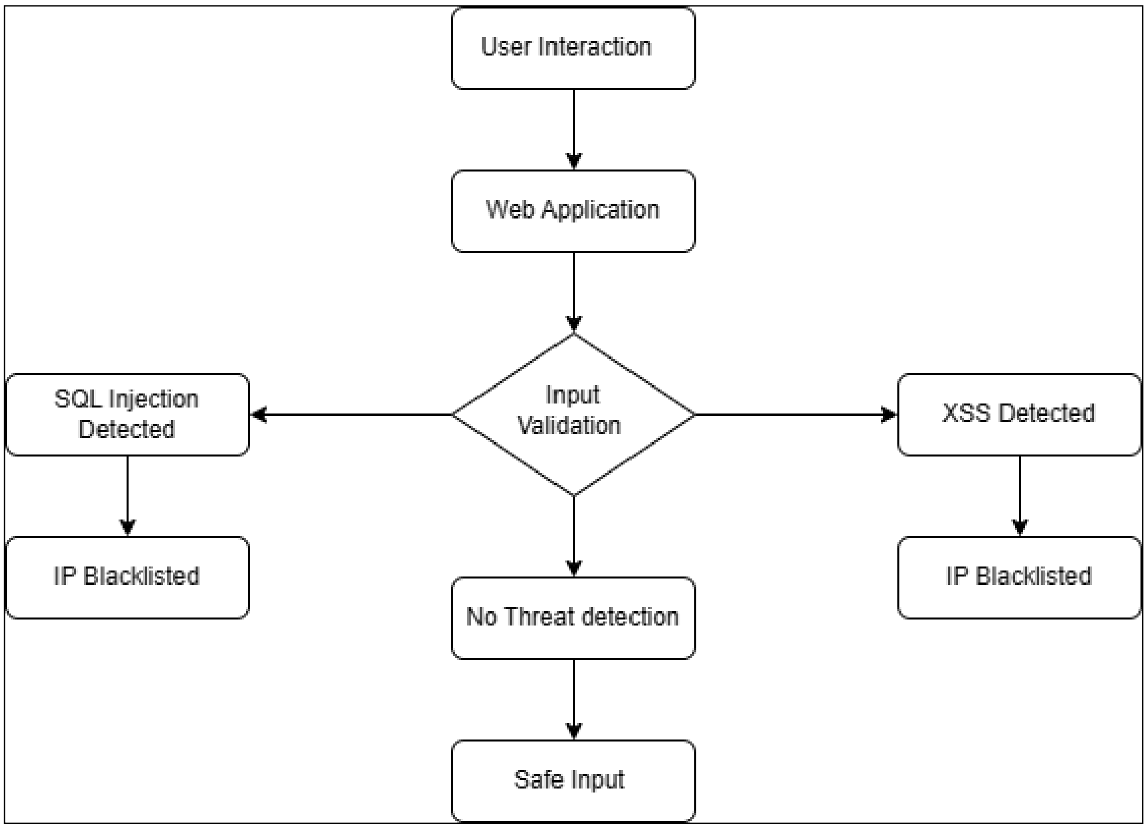


Figure 1: Design flow of the application.

5 Implementation

The tool detects and prevents SQL injection and XSS attacks works by identifying potential attack patterns in user input, and determining whether an input is safe or malicious and suspicious inputs. This helps to protect user data, maintain application integrity, and uphold user trust.

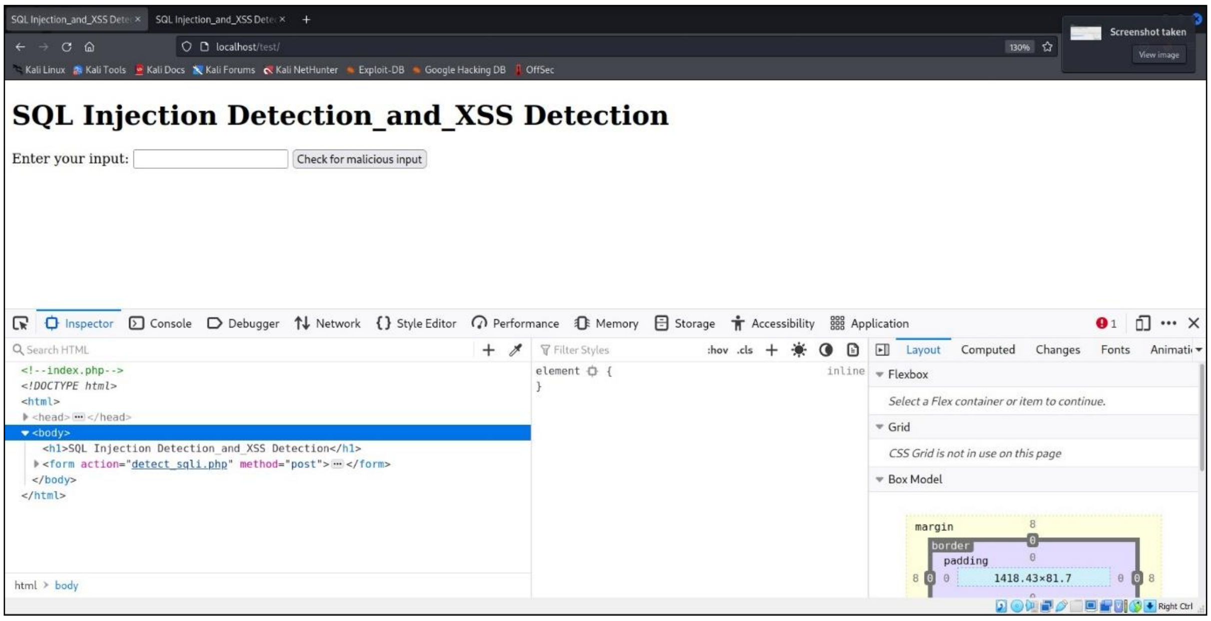
The final outputs of the tool can be summarised below.

- ***Pattern detection***: The tool uses patterns to identify potential attacks in user input. This includes looking for common keywords and phrases, as well as specific character sequences that are often used in attacks.
- ***Attack detection status***: The tool determines whether an input is safe or malicious based on the results of the pattern detection, upon detection of a potential attack, the tool re-directs to the blocked screen.
- ***IP blacklisting and logging***: If the tool detects an attack, it will log relevant information such as the attacker's IP address, and the attacker's IP address is automatically added to a blacklist, which will prevent them from accessing the web application in the future.
- ***User-friendly interface feedback***: The tool provides feedback to users regarding the outcome of their inputs.

Tools and Languages Used:

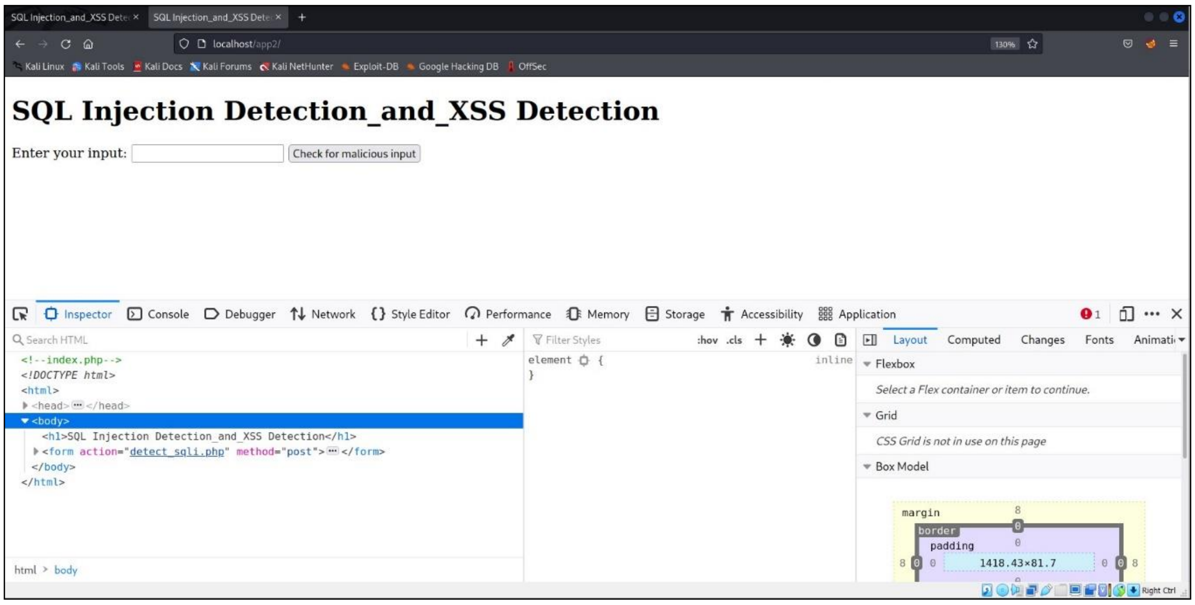
The developed tool combines the capabilities of multiple tools and languages:

- ***Frontend***: PHP is used to create the user interface where the user input data and receives feedback.
- ***Backend Detection Script***: Python is used to implement the backend script responsible for pattern detection and attack prevention.
- ***Pattern Detection***: Manual SQL injection attack and XSS attack patterns are entered within the Python script to identify attack patterns based.
- ***Logging and IP Blacklisting***: Python handles the logging of attack information and adding attacker IPs to the blacklist.
- ***User Interface***: PHP is used to create the user interface that presents the detection results and feedback to users.



```
<!--index.php-->
<!DOCTYPE html>
<html>
 <head>…</head>
 <body>
   <h1>SQL Injection Detection_and_XSS Detection</h1>
   <form action="detect_sqli.php" method="post">…</form>
 </body>
</html>
```

Figure 2: Frontend of the application without detection capability using PHP.


SQL Injection Detection_and_XSS Detection
Enter your input:
Check for malicious input
localhost/app2/

```
<!--index.php-->
<!DOCTYPE html>
<html>
 <head>…</head>
 <body>
   <h1>SQL Injection Detection_and_XSS Detection</h1>
  <form action="detect_sqli.php" method="post">…</form>
  </body>
</html>
```

Figure 3: Frontend of the application with detection capability using PHP.

6 Evaluation

To validate the application, experiments were conducted using manual SQL injection attacks and XSS patterns as input. The application was tested with both detection capabilities turned on and off.

In the experiments with detection capability turned on, the application was able to successfully detect all the attack patterns. In the experiments with detection capability turned off, the application was vulnerable to the attack patterns and allowed them to execute.

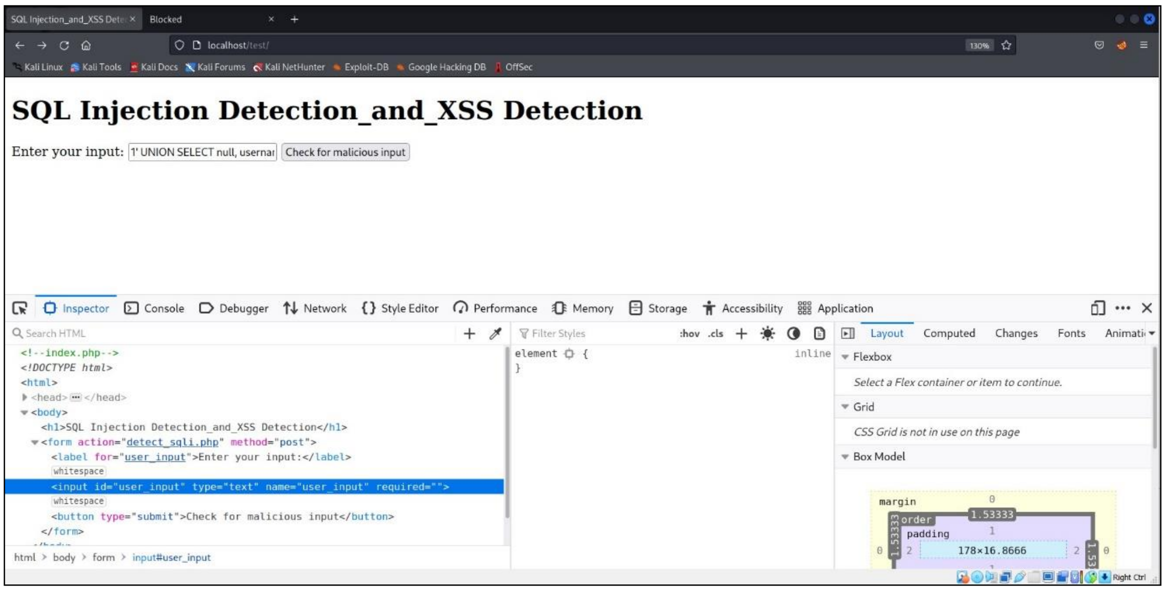
6.1 Experiment 1: The SQL injection attack without the detection and prevention

In this experiment, an SQL injection attack pattern is the user input without the defence mechanism and the attack is not blocked.

1' UNION SELECT null, username || '-' || password FROM admin --

Now, in this the input

- *(1' UNION SELECT null, username || '-' || password FROM admin --)* is concatenated into the SQL query.
- The UNION SELECT is used to add a new '*row*' to the result set as an attempt to retrieve the username and password columns from the admin table.
- *'null'* is used to fill in the id column for the first part of the query.
- The || operator concatenates the username and password columns and separates them with a hyphen.



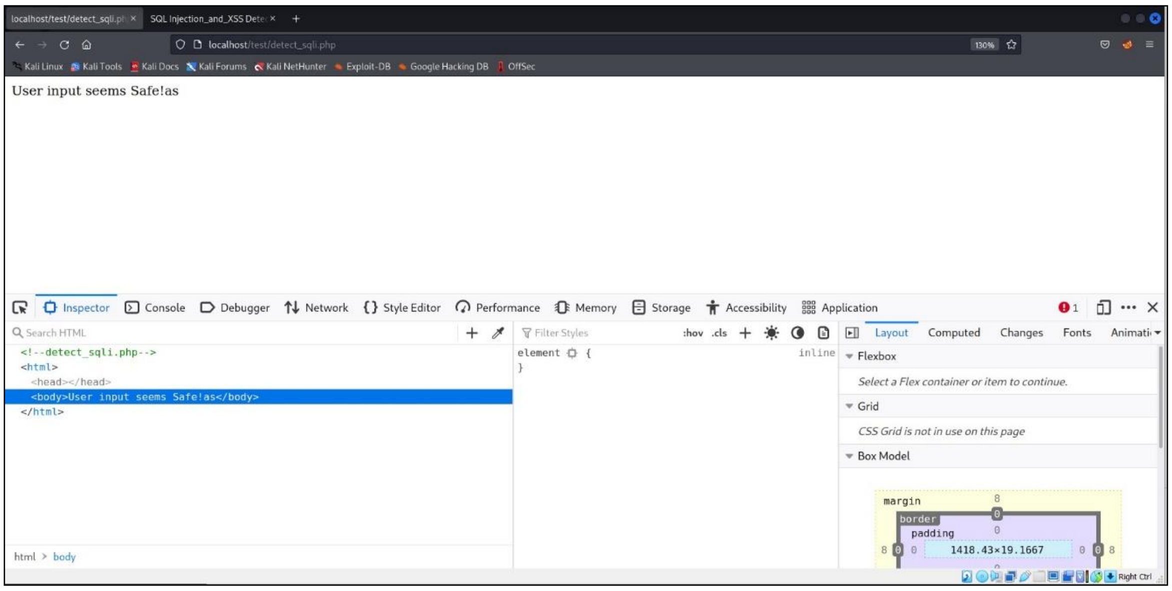
SQL Injection Detection_and_XSS Detection

Enter your input: 1' UNION SELECT null, usernar [Check for malicious input]

```
<!--index.php-->
<!DOCTYPE html>
<html>
▸ <head>…</head>
▾ <body>
    <h1>SQL Injection Detection_and_XSS Detection</h1>
  ▾ <form action="detect_sqli.php" method="post">
      <label for="user_input">Enter your input:</label>
      whitespace
      <input id="user_input" type="text" name="user_input" required="">
      whitespace
      <button type="submit">Check for malicious input</button>
    </form>
```

Figure 4: SQL injection attack without detection capability

Further, this input is accepted by the application without any detection and registered as input.



User input seems Safe!as

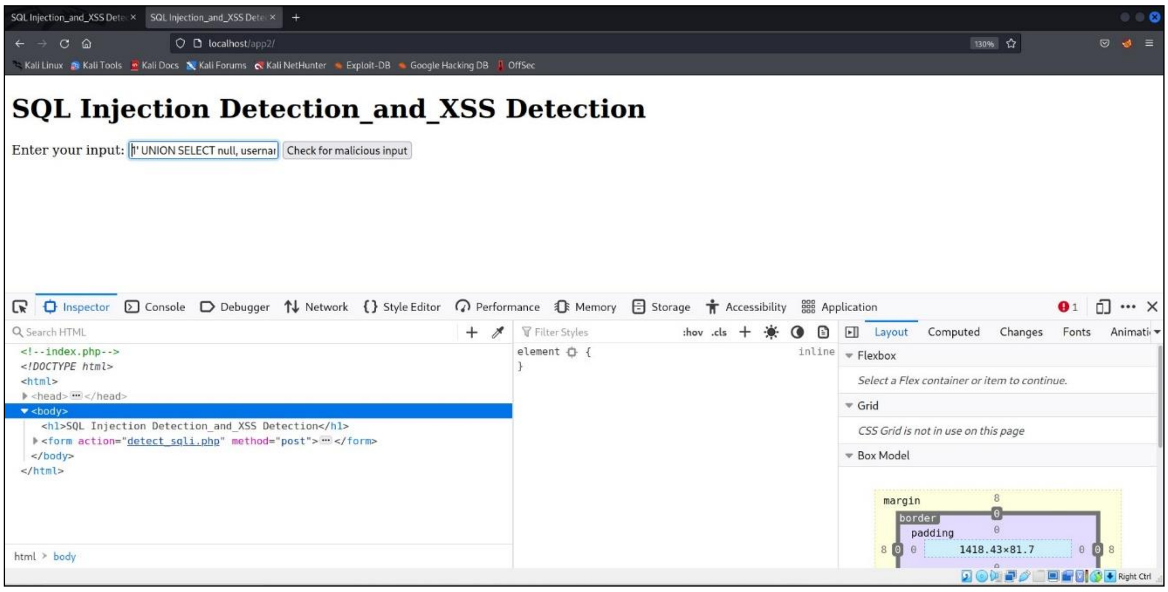
```
<!--detect_sqli.php-->
<html>
  <head></head>
  <body>User input seems Safe!as</body>
</html>
```

Figure 5: SQL injection attack successful without detection

6.2 Experiment 2: The SQL injection attack with the detection and prevention

In this experiment, the same SQL injection attack pattern is the user input but now with the defence mechanism, the attack is blocked.

1' UNION SELECT null, username || '-' || password FROM admin –



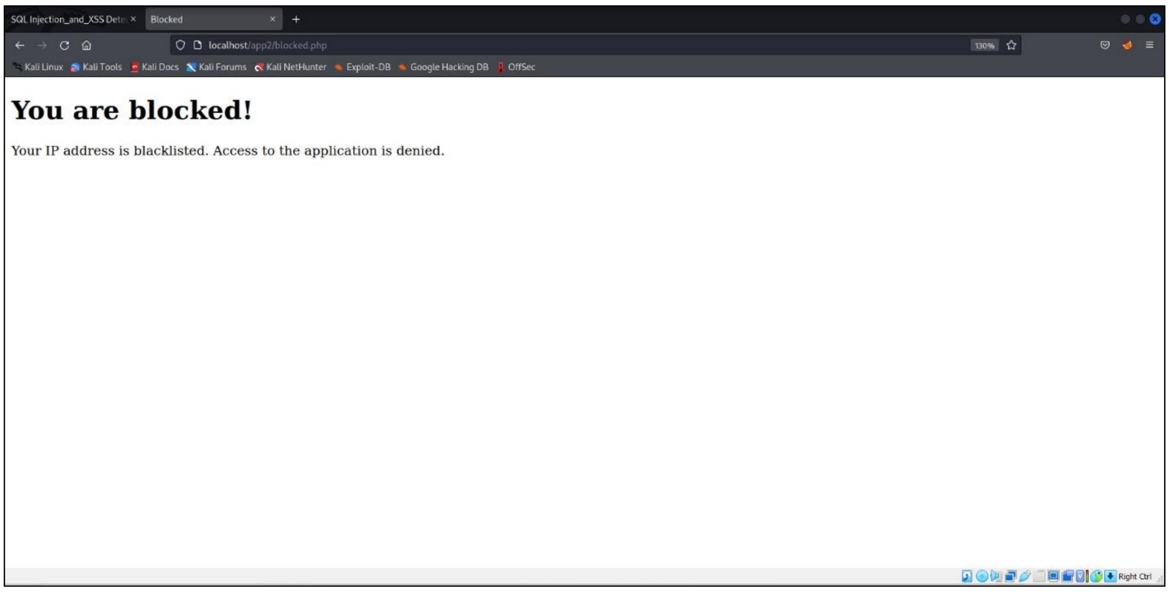
SQL Injection Detection_and_XSS Detection

Enter your input: 1' UNION SELECT null, usernar

```
<!--index.php-->
<!DOCTYPE html>
<html>
<head>…</head>
<body>
  <h1>SQL Injection Detection_and_XSS Detection</h1>
  <form action="detect_sqli.php" method="post">…</form>
</body>
</html>
```

Figure 6: SQL injection attack with detection capability

Further, this input is detected as a potential threat by the application and IP is blacklisted.



You are blocked!

Your IP address is blacklisted. Access to the application is denied.

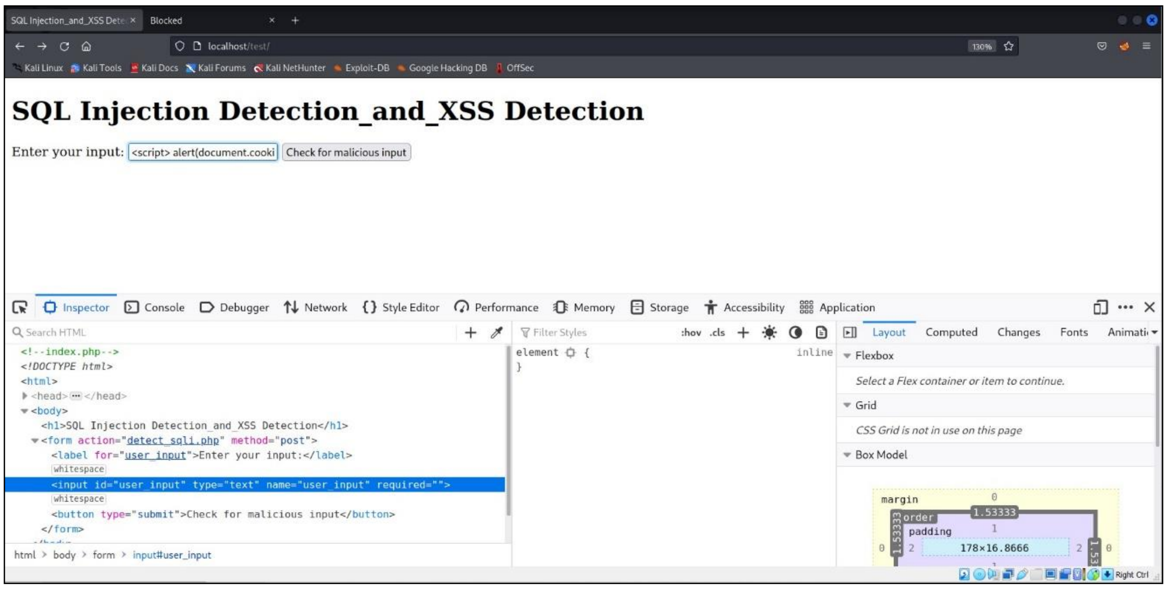
Figure 7: SQL injection attack detected and prevented.

6.3 Experiment 3: The XSS attack without the detection and prevention

In this experiment, an XSS attack pattern is the user input without the defence mechanism and the attack is not blocked.

<script>alert(document.cookie);</script>

Now, in this input the missing closing angle bracket *("/>")* for the *<script>* tag is added, and the alert function correctly ends with a closing parenthesis “)”.



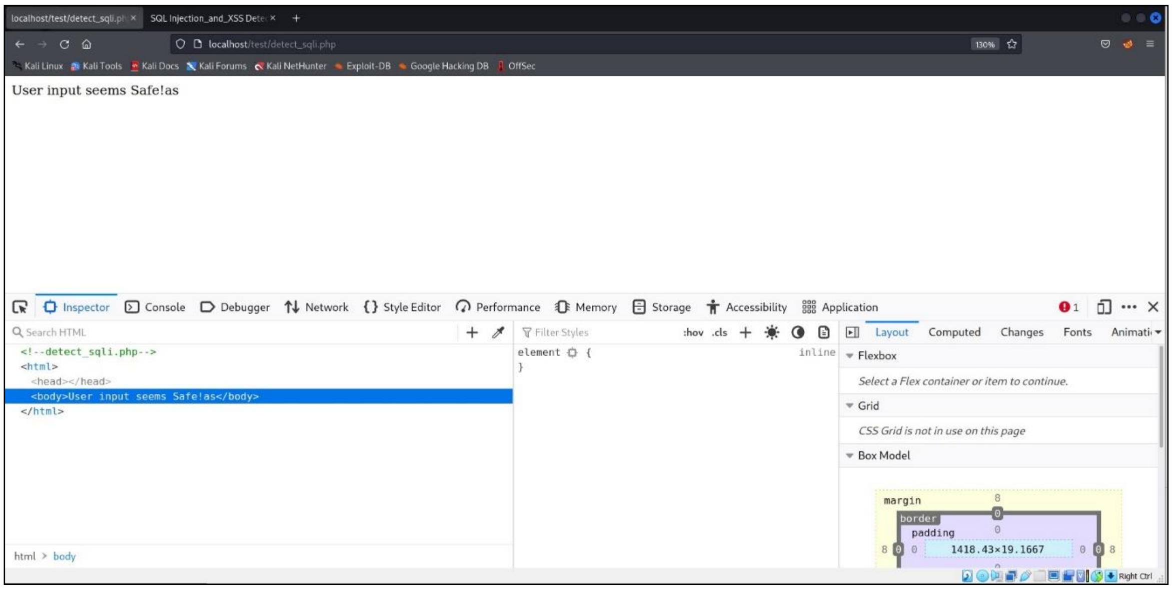
SQL Injection Detection_and_XSS Detection

Enter your input: <script> alert(document.cooki

```
<!--index.php-->
<!DOCTYPE html>
<html>
<head>…</head>
<body>
  <h1>SQL Injection Detection_and_XSS Detection</h1>
  <form action="detect_sqli.php" method="post">
    <label for="user_input">Enter your input:</label>
    <input id="user_input" type="text" name="user_input" required="">
    <button type="submit">Check for malicious input</button>
  </form>
```

Figure 8: XSS attack without detection capability

Further, this input is accepted by the application without any detection and registered as input.



User input seems Safe!as

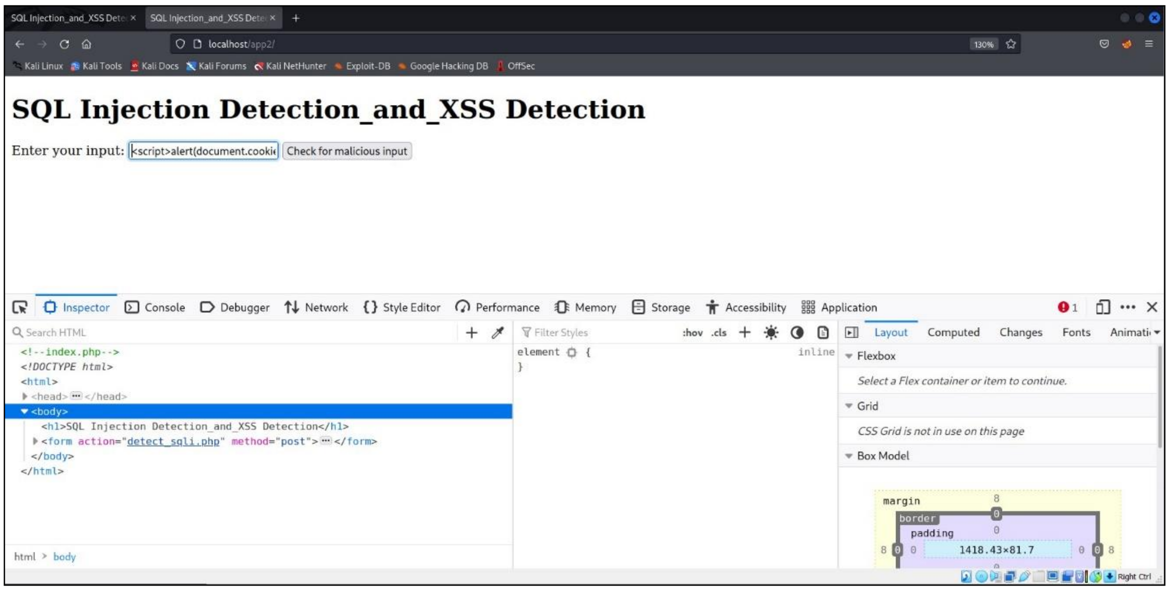
```
<!--detect_sqli.php-->
<html>
  <head></head>
  <body>User input seems Safe!as</body>
</html>
```

Figure 9: XSS attack successful without detection.

6.4 Experiment 4: The XSS attack with the detection and prevention

In this experiment, an XSS attack pattern is the user input with the defence mechanism and the attack is blocked.

<script>alert(document.cookie);</script>

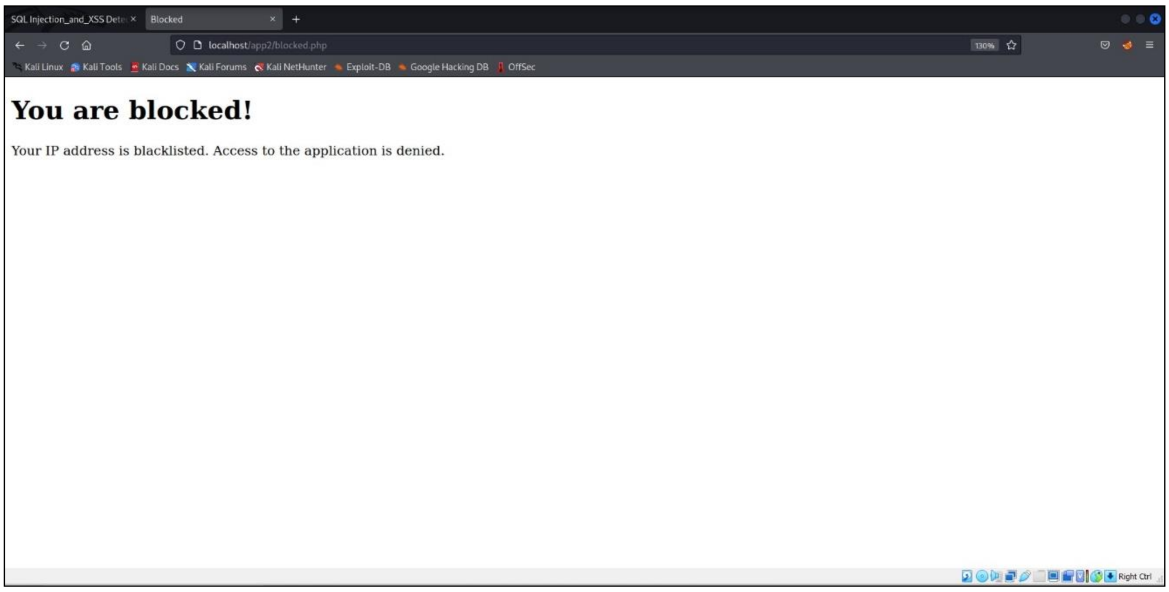

SQL Injection Detection_and_XSS Detection

Enter your input: <script>alert(document.cookie

```
<!--index.php-->
<!DOCTYPE html>
<html>
 <head>…</head>
 <body>
   <h1>SQL Injection Detection_and_XSS Detection</h1>
   <form action="detect_sqli.php" method="post">…</form>
 </body>
</html>
```

Figure 10: XSS attack with detection capability

Further, this input is detected as a potential threat by the application and IP is blacklisted.


localhost/app2/blocked.php

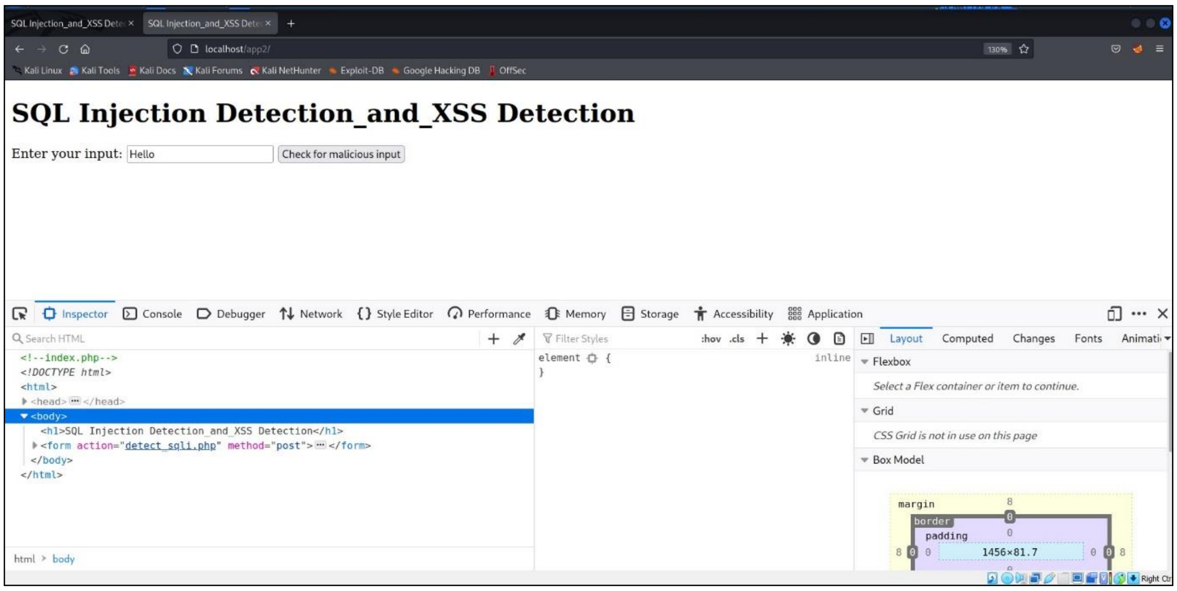
You are blocked!

Your IP address is blacklisted. Access to the application is denied.

Figure 11: XSS attack detected and prevented.

6.5 Experiment 5: The Safe input with the detection and prevention

In this experiment, a simple ‘hello’ word is used as input, and it is registered as safe input.

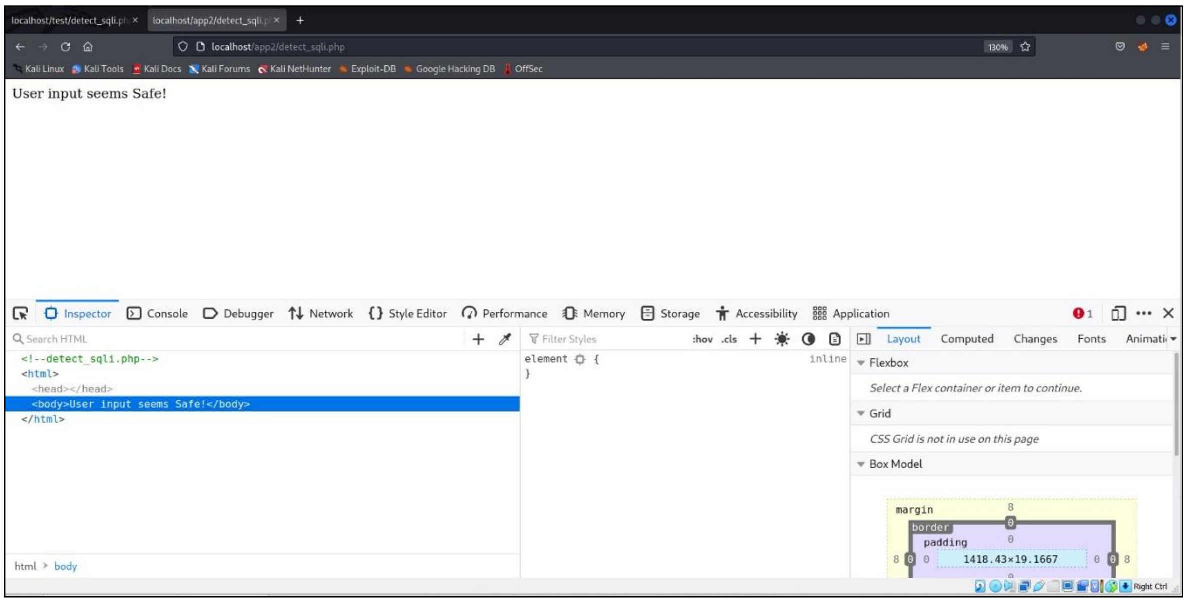


SQL Injection Detection_and_XSS Detection

Enter your input: Hello [Check for malicious input]

```
<!--index.php-->
<!DOCTYPE html>
<html>
<head>⋯</head>
<body>
  <h1>SQL Injection Detection_and_XSS Detection</h1>
  <form action="detect_sqli.php" method="post">⋯</form>
</body>
</html>
```

Figure 13: Safe input ‘’.



User input seems Safe!

```
<!--detect_sqli.php-->
<html>
  <head></head>
  <body>User input seems Safe!</body>
</html>
```

Figure 13: Safe input has been successfully registered.

6.6 Discussion

The developed application is a novel tool to mitigate both SQL injection and XSS attacks. Further, these attacks when detected can be blocked by identifying the IP address of the attacker, and for security purposes, the IP is blacklisted. The application has been tested manually for both SQL injection and XSS attacks with IP blacklisting.

The work could have been made more robust and enhanced with suggested improvements for the application.

- ***Enhanced Pattern Detection*** (Meharaj Begum and Arock, 2021): The pattern detection mechanism can be updated and expanded to cover emerging attack vectors and evasion techniques. This will help us to detect and block attacks more effectively.
- ***Real-time Alerts*** (Pereira, Campos and Vieira, 2019): Real-time alerts and notifications for the administrator when an attack is detected can be implemented allowing the administrator to take swift action to mitigate the attack.
- ***Machine Learning*** (Bhardwaj *et al.*, 2022): The integration of machine learning algorithms to enhance the accuracy of attack detection and minimize false positives can be done. This can help detect attacks more accurately without generating unnecessary alerts.
- ***WAF Integration*** (Robinson, Akbar and Ridha, 2018): The Web Application Firewall (WAF) can further be refined to allow adaptivity to adjust WAF rules based on incoming threats, providing even more protection.
- ***Security Reports*** (Martin and Lam, no date): The generation of automated detailed security reports, including attack logs and analysis can be implemented to help the administrator with the threat landscape to make informed security decisions.
- ***Community Contribution*** (Schryen, 2011): The application can be open-sourced for encouraging community contributions allowing collaboration with others from the security field to continuously improve the application.

7 Conclusion and Future Work

The developed application is a powerful and innovative solution to protect web applications from two critical security vulnerabilities: SQL injection and cross-site scripting (XSS). The application uses a combination of PHP and Python scripts to effectively detect and mitigate potential attacks, safeguarding user data. The implementation of IP blacklisting adds a layer of defence, further strengthening the security posture.

For future work, web application security can be continuously improved by enhancing pattern detection, implementing real-time alerts, integrating machine learning, refining WAF integration, providing security reports, and encouraging community contributions.

References

571b, E. and Survey Paper, T. (2012) 'Approaches to detect SQL injection and XSS in web applications'.

Bhardwaj, A., Chandok, S.S., Bagnawar, A., Mishra, S. and Uplaonkar, D. (2022) 'Detection of Cyber Attacks: XSS, SQLI, Phishing Attacks and Detecting Intrusion Using Machine Learning Algorithms', *2022 IEEE Global Conference on Computing, Power and Communication Technologies, GlobConPT 2022* [Preprint]. Available at: https://doi.org/10.1109/GLOBCONPT57482.2022.9938367.

Boyd, S.W. and Keromytis, A.D. (2004) 'SQLrand: Preventing SQL injection attacks', *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3089, pp. 292–302. Available at: https://doi.org/10.1007/978-3-540-24852-1_21/COVER.

Demetrio, L., Valenza, A., Costa, G. and Lagorio, G. (2020) 'WAF-A-MoLE: Evading web application firewalls through adversarial machine learning', *Proceedings of the ACM Symposium on Applied Computing*, pp. 1745–1752. Available at: https://doi.org/10.1145/3341105.3373962.

'Detecting Blind Cross-Site Scripting Attacks Using Machine Learning' (2018). Available at: https://doi.org/10.1145/3297067.3297096.

Endler, D. (2002) 'The Evolution of Cross-Site Scripting Attacks'.

Fu, X., Lu, X., Peltsverger, B., Chen, S., Qian, K. and Tao, L. (2007) 'A static analysis framework for detecting SQL injection vulnerabilities', *Proceedings - International Computer Software and Applications Conference*, 1, pp.

87–94. Available at: https://doi.org/10.1109/COMPSAC.2007.43.

Gandotra, E., Bansal, D. and Sofat, S. (2015) 'Computational Techniques for Predicting Cyber Threats', *Advances in Intelligent Systems and Computing*, 308 AISC(VOLUME 1), pp. 247–253. Available at: https://doi.org/10.1007/978-81-322-2012-1_26/COVER.

Ghanbari, Z., Rahmani, Y., Ghaffarian, H. and Ahmadzadegan, M.H. (2016) 'Comparative approach to web application firewalls', *Conference Proceedings of 2015 2nd International Conference on Knowledge-Based Engineering and Innovation, KBEI 2015*, pp. 808–812. Available at: https://doi.org/10.1109/KBEI.2015.7436148.

Halfond, W.G.J., Viegas, J. and Orso, A. (2006) 'A Classification of SQL-Injection Attacks and Countermeasures'.
Krueger, T., Gehl, C., Rieck, K. and Laskov, P. (2010) 'TokDoc: A self-healing web application firewall', *Proceedings of the ACM Symposium on Applied Computing*, pp. 1846–1853. Available at: https://doi.org/10.1145/1774088.1774480.

Li, Y., Krishnamurthy, R., Raghavan, S., Vaithyanathan, S. and Jagadish, H. V (2008) 'Regular Expression Learning for Information Extraction', pp. 21–30.

Martin, M. and Lam, M.S. (no date) 'Automatic Generation of XSS and SQL Injection Attacks with Goal-Directed Model Checking', (1). Available at: http://example.com/search_ (Accessed: 13 August 2023).

Meharaj Begum, B.A. and Arock, M. (2021) 'Efficient Detection of SQL Injection Attack(SQLIA) Using Pattern-based Neural Network Model', *Proceedings - IEEE 2021 International Conference on Computing, Communication, and Intelligent Systems, ICCCIS 2021*, pp. 343–347. Available at: https://doi.org/10.1109/ICCCIS51004.2021.9397066.

MeiJunjin (2009) 'An approach for SQL injection vulnerability detection', *ITNG 2009 - 6th International Conference on Information Technology: New Generations*, pp. 1411–1414. Available at: https://doi.org/10.1109/ITNG.2009.34.

Pandian, S.L., Nithya, V. and Malarvizhi, C. (2015) 'A Survey on Detection and Prevention of Cross-Site Scripting Attack Question and Answering System for Tamil Documents View project A Survey on Detection and Prevention of Cross-Site Scripting Attack', *Article in International Journal of Security and Its Applications*, 9(3), pp. 139–152. Available at: https://doi.org/10.14257/ijsia.2015.9.3.14.

Pereira, J.D.A., Campos, J.R. and Vieira, M. (2019) 'An exploratory study on machine learning to combine security vulnerability alerts from static analysis tools', *2019 9th Latin-American Symposium on Dependable Computing, LADC 2019 - Proceedings* [Preprint]. Available at: https://doi.org/10.1109/LADC48089.2019.8995685.

Razzaq, A., Hur, A., Shahbaz, S., Masood, M. and Ahmad, H.F. (2013) ‘Critical analysis on web application firewall solutions’, pp. 1–6. Available at: https://doi.org/10.1109/ISADS.2013.6513431.

Robinson, Akbar, M. and Ridha, M.A.F. (2018) ‘SQL Injection and Cross Site Scripting Prevention using OWASP ModSecurity Web Application Firewall’, *JOIV : International Journal on Informatics Visualization*, 2(4), pp. 286–292. Available at: https://doi.org/10.30630/JOIV.2.4.107.

Schryen, G. (2011) ‘Is open source security a myth?’, *Communications of the ACM*, 54(5), pp. 130–140. Available at: https://doi.org/10.1145/1941487.1941516.

Shar, L.K. and Tan, H.B.K. (2012) ‘Defending against cross-site scripting attacks’, *Computer*, 45(3), pp. 55–62. Available at: https://doi.org/10.1109/MC.2011.261.

Singh, J.P. (2017) ‘Analysis of SQL Injection Detection Techniques’, *Theoretical and Applied Informatics*, 28(1 & 2), pp. 37–55. Available at: https://doi.org/10.20904/281-2037.

Sreedhar, V., Ji, P., Luo, L., Yang, S., Zhang, Y. and Zurko, M.E. (2012) ‘IBM Research Report User Friendly Web Application Firewall with Client Pre-Checking User Friendly Web Application Firewall with Client Pre-checking’, *Computer Science*, pp. 25297–1207.

Tekerek, A., Gemci, C. and Bay, O.F. (2014) ‘Development of a hybrid web application firewall to prevent web based attacks’, *8th IEEE International Conference on Application of Information and Communication Technologies, AICT 2014 - Conference Proceedings* [Preprint]. Available at: https://doi.org/10.1109/ICAICT.2014.7035910.

Vogt, P., Nentwich, F., Jovanovic, N., Kirda, E., Kruegel, C. and Vigna, G. (no date) ‘Cross-Site Scripting Prevention with Dynamic Data Tainting and Static Analysis’.

Zborowski, J. and Pańczyk, M. (2023) ‘Comparative analysis of web applications implemented in: PHP and Python’, *Journal of Computer Sciences Institute*, 26, pp. 18–22. Available at: https://doi.org/10.35784/JCSI.3071.