**CHAPTER-1**

**INTRODUCTION**

* 1. **INTRODUCTION:**

In an era marked by escalating cyber threats and the increasing complexity of information technology infrastructures, the need for robust security measures is paramount. One of the most effective means of gauging and bolstering an organization's cybersecurity defenses is through penetration testing, commonly referred to as pen testing. Penetration testing involves the simulation of real-world cyberattacks to identify vulnerabilities in an organization's systems, networks, and applications, thus enabling security teams to remediate these weaknesses before malicious actors can exploit them.

While the concept of penetration testing has been instrumental in fortifying cybersecurity practices, the traditional methods have often been arduous, time-consuming, and reliant on the expertise of highly skilled professionals. This has created a growing demand for tools that can streamline and simplify the penetration testing process. Enter the User Interface-Based Automated Penetration Testing Tool (UI-APT), a groundbreaking solution designed to revolutionize how security assessments are conducted.

This tool, with its user-friendly interface and automated capabilities, represents a significant leap forward in the realm of penetration testing. It empowers organizations to conduct comprehensive security assessments more efficiently, making the process accessible to a broader range of security professionals. UI-APT simplifies complex tasks, reduces the need for extensive technical knowledge, and centralizes control, ultimately enhancing the security of an organization's digital assets.

**CHAPTER-2**

**LITERATURE SURVEYS**

**2.1 AUTOMATED PENETRATION TESTING: AN OVERVIEW**

**Farah Abu-Dabaseh and Esraa Alshammari**

**2.1.1 INTRODUCTION:**

Penetration testing is used to check the exploitations and the vulnerability of the organization's system and help the developers to build a protected system that meets the needs. It's very important to any organization and company to protect their data and information from outside attackers and keep monitoring to the prioritize the severity of the security issues. Determining the priorities can help the developers to determine the needed devices in the allocation of the budget for security issues. Additionally, can be used to find the financial loss expected and risks if the attackers achieve their goals and exploited the system and how to mitigate that. The data generated from the test considered confidential and private data because it shows approximately all the holes in the system and how they could be exploited.

PT can be done by attacking the system similar to the action of the outside attackers and find out what can be obtained. The attack might not be as easy as exploiting one vulnerability, many vulnerabilities may be used to achieve the goal by making a sequence of attack chain (Multi-step attack). It’s also considered as a risk assessment and can be used to check the network safety. When penetration test is done, the roles of engagement for that test should be set also, to set the goals and the methodology of the test. Penetration tests companies can be classified into three different types: gray hat, black hat, and white hat. In the white hat, the tester is an ethical hacker that respects the rules of the organization and the employees can help to perform the testing. While the black hat is mainly used to find how the employees interact with the undesired attack, in this approach the administrators are only the ones who know the test is underway. Moreover, we can do a Gray hat which is a combined approach to the previous types into a custom test plan. Penetration testing should be considered as a standard frequent process within the security roadmap.

Traditionally, the organizations used to perform the penetration testing only when they have a product release or a major upgrade.However, it’s suitable to perform the test in these situations:

• New installed software

• Applied system upgrades

• User policy modification

• Applied Security patches

• New infrastructure is added

Although it's important to have a penetration testing in the organization, it’s hard to implement too. Since it should include a security expert with the capability to do such a complex job. That could be an overhead on the organization and could waste time and money without the desired Result in the case that the security team wasn’t as professional as they must. So, the automated approach has seen the light; done by an expert security team in the field. The contribution of this research will consider the standards of penetration testing, the tools used for each phase in the penetration test, the comparison between the automated and manual approaches and the comparison between some of the current approaches for the automated penetration test.

The rest of this research is as follows: in section two the Penetration testing standards will presents. While the comparison between manual and automated techniques in penetration testing are provided in section three. Section four shows an overview of the current automated penetration testing. Finally, the conclusion and future works are presented in section five.

**2.1.2 PENTRATON TESTING STANDARDS**

Standards for penetration testing aimed to provide a basic outline and definition of the penetration testing. Also to give an outline of the steps used for it, many standards are out there having various pros and cons [6]. Choosing one of them should be based on the goal of having the test. There are currently various standards that could be followed, such as ISAAF (Information Systems Security Assessment Framework), the OSSTMM (Open-Source Security Testing Methodology Manual), the NIST SP 800-115, and the PTES (the Penetration Testing Execution Standard), the OISSG (Open Information Systems Security Group)

A diagram of a diagram

Description automatically generated

Figure.2.1.1: The Phases of Penetration Testing (NIST) Standard

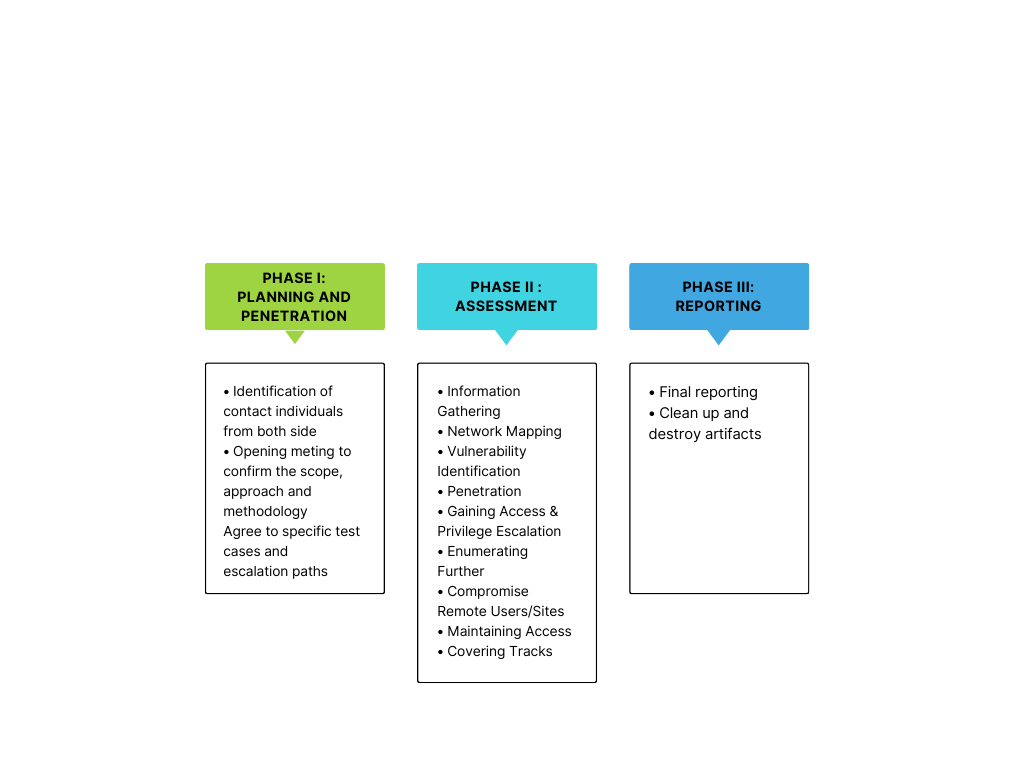


Figure 2.1.2: The Phases of Penetration Testing (ISSAF) Standard

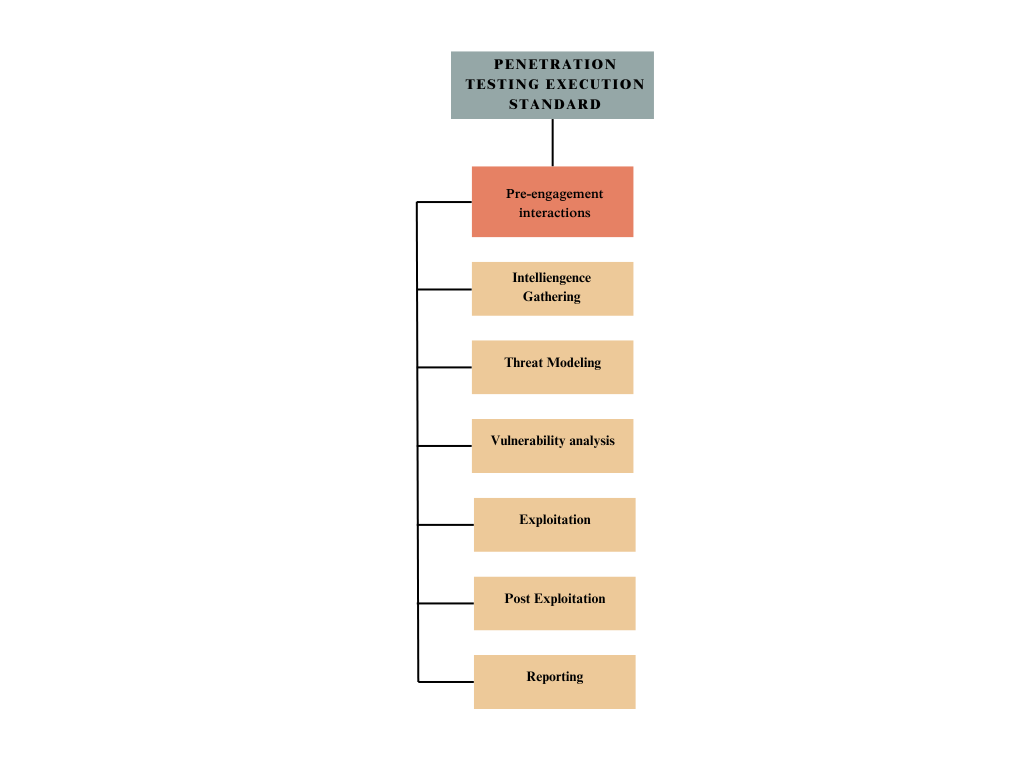


Figure 2.1.3: Steps of Penetration Testing Execution Standard

**2.1.3 CONCLUSION:**

Many organizations need penetration testing to discover the most vulnerabilities that have in their system. To apply the penetration test, there are two approaches that the organizations used to discover the bugs, one is automated penetration test and the other is manual penetration test. The automated pen test is the easiest way to figure out the whole vulnerabilities in the system by implementing a tool that has some patterns to find the vulnerabilities.

While the manual test is the way to discover the vulnerabilities manually through analyzing the system and distinguish the abnormal behavior. Hence, this paper has been done to shows the importance of the penetration testing as well as the importance of automating this process. Additionally, some standards in the penetration testing have been highlighted to help the researchers find the suitable standards to use. Even more, the comparison between the manual and automated penetration testing has been provided in term of the testing process, vulnerability and attack database management, exploit development and management reporting, clean up, network modification, logging, and training. And the result shows that the automated penetration testing is better than the manual penetration in all of the above process, except finding the new or zero day exploits. So that many organizations may go for the automated approach because it seems the better and cheaper way to maintain security in the systems as most of the vulnerabilities that the attackers used to exploit the system are well defined in the automated tools.

Although writing own exploits may be time-consuming as well as ineffective in terms of money. But, the attackers can conceal their activity through their own scripts. Thus, the automated tools still have limitations and vulnerabilities.

**2.2 AUTOMATED PENETRATION TESTING USING DEEP REINFORCEMENT LEARNING.**

**Zhenguo Hu, Razvan Beuran, Yasuo Tan Japan Advanced Institute of Science and Technology Nomi, Ishikawa, Japan.**

**2.2.1 INTRODUCTION:**

With the generalized use of computer networks and the frequent occurrence of security incidents, cybersecurity has become an increasingly prominent problem. An effective way to address this issue is to assess the network security features of a system by employing penetration testing. Penetration testing is an authorized attack methodology used to evaluate the security features of a system, in particular to identify its security weaknesses by conducting ethical attacks on it. The method is well established, and several commercial tools are available to assist the pentesters in performing the more tedious tasks [1]. In the field of cybersecurity education and training, defense training similarly requires that ethical hackers conduct attacks in the training environment, so that trainees can get hands-on experience with handling security incidents. Penetration testing methodology is perfectly suitable for playing the attackers’ role in this kind of scenario too. Generally, penetration testing is performed mostly manually, as pentesters need first to analyze the target system, then exploit the discovered vulnerabilities in different ways in order to penetrate the system and compromise network resources in proof-of-concept attacks.

This process is a laborious, time-consuming and complex task that requires a great deal of tacit knowledge, which cannot be easily formalized, and is also prone to human errors. Therefore, in recent years, more and more people tried to use model based attack planning to generate the attacks by employing a model of the target system. The Core Security company employs this idea commercially since 2010, and in their tool named “Core IMPACT” uses as attack planner a variant of the Metric-FF system. Regarding the use of artificial intelligence (AI) techniques in this area, Boddy et al. first applied AI planning to penetration testing , which lead to the inclusion of the cybersecurity domain into the 2008 International Planning Competition.

Several years later, the Core Security model has been used to simulate external attacks by making individual “attack actions” correspond to known software vulnerability exploits.

However, none of these methods provided a complete penetration testing solution. The attack tree method could play an important role in ensuring that automated penetration testing is close to human actions. This method for modelling security threats on a given system was proposed by Schneier in 1999, and represents an attack against it in the form of a tree structure. By analyzing the attack tree, we can better understand the relationship between each attack method.

Compared with reinforcement learning, deep reinforcement learning (DRL) is a more suitable approach for analyzing the attack tree, as it combines deep learning and reinforcement learning and adopts a trial-and-error approach to find an optimal solution. In this paper we present an automated penetration testing framework that we designed and implemented with the goal of finding the best attack path for a given topology, similarly to a human attacker. Our main contributions are:

• Discuss the use of Shodan to collect actual server data for constructing realistic target networks 2 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)

• Introduce our approach of building an attack representation matrix by using the MulVAL and Depth-First Search (DFS) algorithms

• Present the manner in which the Deep Q-Learning Network method is used to analyze the attack matrix and find the optimal attack path

• Evaluate our approach over realistic network scenarios built using Shodan data to show the effectiveness of DQN for automated penetration testing.

**2.2.2 FRAMEWORK ARCHITECTURE:**

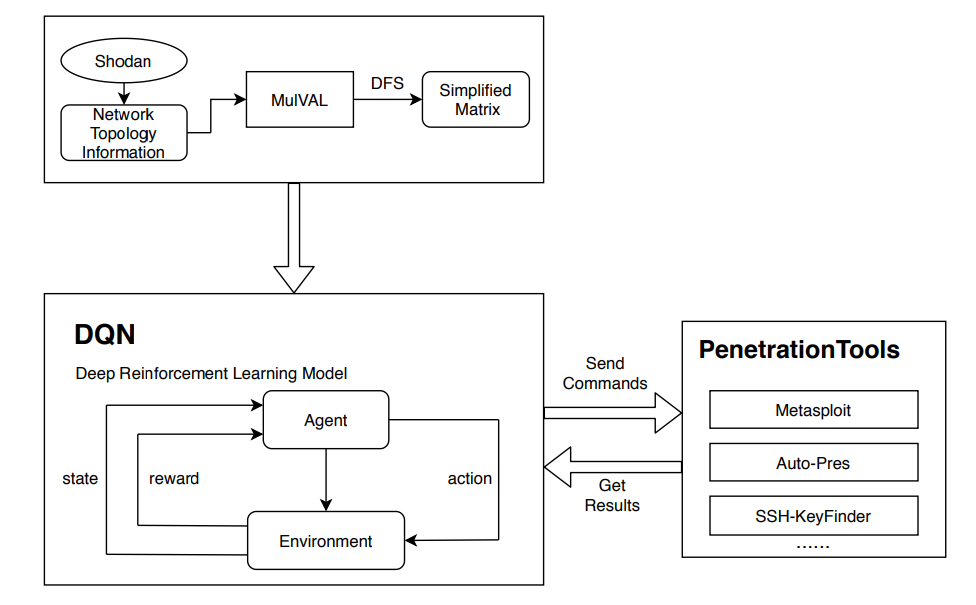
In this section we present the automated penetration testing framework that we designed and implemented based on deep reinforcement learning techniques.

The framework has three main components:

• **Training Data:** Build the training data needed as input by the DQN algorithm

• **DQN Module:** Train the DQN algorithm, then use the trained model for penetration testing

**• Penetration Tools:** Wrapper for external tools used to perform actions on real systems

Figure 2.2.1. Architecture of the automated penetration testing framework

**2.2.3 EXPERIMENT SCENARIO:**

The model represent a small company network which includes a web server, a file server and a workstation. The file server and workstation are in one subnet connected via a router that further connects to a firewall. The web server is in a different subnet and connects via the firewall to the Internet.

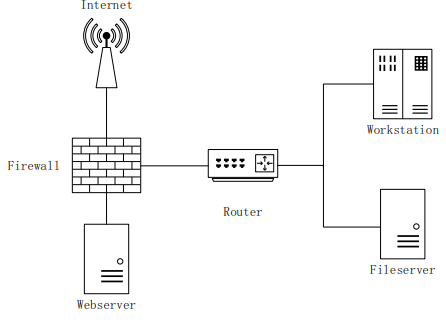
****

Figure 2.2.2 Network topology for the experiment scenario.

**2.2.4 RESULT:**

The simplified matrix above becomes the input of the DQN model, which is then trained to determine the total rewards for all the possible paths. For the network topology we presented, we used Shodan data with different vulnerabilities for the considered protocols into the training template so as to create a totoal of 2000 different attack trees as training data, and 1000 different attack trees as validation data. For validation purposes we define that the optimal path in the simplified matrix is that with the smallest number of steps when the highest reward is selected.

The average reward changes of the DQN model for one of the attack paths in the experiment scenario during a total of 100 training iterations. One can notice that the reward is small at first, then gradually rises after about 30 iterations. After about 60 iterations, the reward becomes stable meaning that the DQN model has found the optimal path. In that case the path is given by the sequence of steps “26 → 24 → 17 → 16 → 15 → 13 → 10 → 9 → 8 → 6 → 5 → 4 → 3 → 2 → 1” (cf. Fig. 9). This path not only makes full use of the vulnerabilities of both the web server and the file server, but also adopts simple attack methods that are easy to use via penetration testing tools.

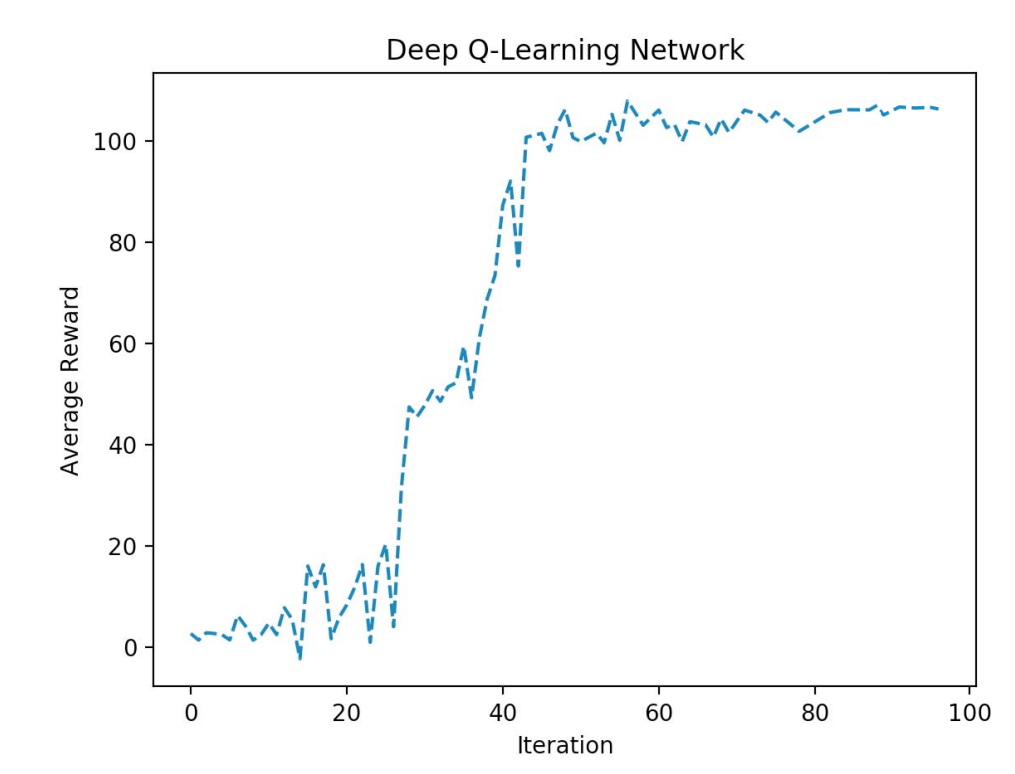
****

Figure 2.2.3. Reward changes for one attack path in the experiment scenario.

**2.2.5 DRAWBACKS:**

Deep learning-based automated penetration testing tools offer the potential for more efficient and accurate security assessments, but they come with several limitations. These include challenges related to generalization, a heavy dependence on data quality and quantity, the risk of false positives and false negatives, model complexity and opacity, vulnerability to adversarial attacks, resource-intensive training and deployment, the lack of domain-specific knowledge, and limited coverage in security assessments. Additionally, ethical and regulatory concerns must be addressed, and the tools may require complementing with traditional testing methods and human expertise to ensure comprehensive and reliable security assessments.

Furthermore, deep learning tools may have difficulty interpreting and responding to the broader organizational context, making it challenging to prioritize security findings effectively. They might excel in the identification of specific vulnerabilities but could lack the understanding of post-exploitation activities or the scope of a security breach. As the threat landscape evolves, there's a risk of model drift, where the performance of deep learning models deteriorates over time as they struggle to keep pace with emerging threats. Therefore, while deep learning-based automated penetration testing tools hold promise, organizations must carefully consider their limitations and use them as part of a comprehensive security assessment strategy that combines the strengths of both automated tools and human expertise to ensure robust cybersecurity defenses.

In addition, organizations should be aware of the resource requirements associated with deep learning-based penetration testing tools, both in terms of hardware and computational resources for model training and the ongoing maintenance and fine-tuning. They should also invest in rigorous data collection and curation processes to ensure the quality and reliability of the training data, which is critical for the tool's effectiveness.

Despite these drawbacks and considerations, the incorporation of deep learning into penetration testing holds the potential to augment security practices, especially in areas such as anomaly detection, pattern recognition, and rapid vulnerability assessment. Overcoming these limitations and harnessing the benefits of deep learning tools requires a balance between automated testing solutions and the insight, adaptability, and ethical judgment provided by human security professionals, working in tandem to bolster cybersecurity measures and protect organizations against evolving threats.

**2.3 ENSURING WEB APPLICATION SECURITY IN MCUXPRESSO: A COMPREHENSIVE EXPLORATION**

**Peter Selinger**

**2.3.1 INTRODUCTION:**

In the contemporary digital landscape, the proliferation of web applications has redefined the way we interact with technology. From e-commerce platforms to social networking sites, these applications have become integral to our daily lives. However, the increasing reliance on web applications brings forth a parallel rise in cyber threats. The vulnerability of these applications to various forms of cyberattacks necessitates a thorough exploration of web application security. This thesis focuses on scrutinizing and fortifying the security aspects within the MCUXpresso environment, a prominent integrated development environment for NXP microcontrollers. As the backbone of embedded systems development, MCUXpresso's security measures are of paramount importance in safeguarding against potential exploits.

**2.3.2 Web Application Security:**

Web application security is a dynamic and complex field that encompasses a myriad of techniques and practices aimed at safeguarding web applications from an ever-expanding array of cyber threats. From the classic vulnerabilities like SQL injection and cross-site scripting to more contemporary challenges such as API security and serverless architecture vulnerabilities, the landscape is continuously evolving. Within the MCUXpresso framework, understanding and mitigating these vulnerabilities become critical for ensuring the integrity and security of embedded systems.

This thesis delves into the multifaceted dimensions of web application security within MCUXpresso. It explores secure coding practices, emphasizing the importance of adhering to best practices to prevent common pitfalls. Encryption protocols take center stage in discussions surrounding data integrity and confidentiality, especially in the context of IoT (Internet of Things) applications where sensitive information is transmitted between devices.

Authentication mechanisms form another crucial layer of defense, ensuring that only authorized entities can access sensitive functionalities. The study examines various authentication methods, their implementation within MCUXpresso, and the potential risks associated with weak authentication.

Furthermore, the research delves into threat modeling—a proactive approach to identifying and mitigating potential security threats during the application design phase. Continuous monitoring and the role of penetration testing are also explored, underscoring the importance of identifying vulnerabilities in real-world scenarios.

**2.3.3 CONCLUSION:**

In conclusion, the exploration of web application security within MCUXpresso is not merely an exercise in fortifying a development environment but a commitment to creating a secure foundation for the future of embedded systems. As the digital landscape evolves, the integration of robust security measures becomes imperative to mitigate the ever-growing spectrum of cyber threats.

The significance of this research lies not only in its immediate applicability to MCUXpresso users but also in its broader implications for the field of embedded systems development. By fostering a culture of security awareness and best practices, developers can ensure that their applications withstand the test of time and the ever-persistent efforts of cyber adversaries.

The commitment to web application security within MCUXpresso echoes the broader industry sentiment—an acknowledgment that security is not a one-time consideration but an ongoing process. This thesis advocates for a holistic approach to security, acknowledging that it is not solely the responsibility of developers but a collective effort involving continuous collaboration, learning, and adaptation to emerging threats.

In essence, by comprehensively addressing web application security within MCUXpresso, this research contributes to the broader discourse on secure embedded systems development, setting the stage for a future where technology not only advances but does so securely and responsibly.

**2.4 AUTO – OSINT & VULNERABILITY CHECK OREOLUWATOMIWA IBIKUNLE SETU CARLOW**

**2.4.1 INTRODUCTION:**

In an era dominated by digital connectivity, individuals and organizations are increasingly reliant on technology for various aspects of their lives and operations. As technology advances, so do the associated risks and vulnerabilities. Oreoluwatomiwa Ibikunle, a seasoned professional in the field, has embarked on a mission to enhance cybersecurity through the development of an innovative tool—AUTO (Open Source Intelligence & Vulnerability Check). This tool aims to address the ever-growing challenges posed by cyber threats and vulnerabilities, particularly in the context of information security.

### **Security in the Digital Age:**

As we navigate the complexities of the digital age, security becomes a paramount concern. The interconnected nature of systems, coupled with the proliferation of sophisticated cyber threats, necessitates proactive measures to safeguard sensitive information. AUTO, conceptualized by Oreoluwatomiwa Ibikunle, stands at the forefront of these efforts, offering a comprehensive solution that combines Open Source Intelligence (OSINT) and vulnerability checks. This dual approach seeks to fortify digital infrastructures by leveraging intelligence from publicly available sources while identifying and addressing potential weaknesses.

#### **Open Source Intelligence (OSINT):**

OSINT serves as a foundational pillar of AUTO, enabling the tool to gather information from publicly accessible sources. This not only enhances the understanding of potential threats but also provides insights into the broader cybersecurity landscape. By harnessing OSINT, AUTO empowers users to make informed decisions, anticipate emerging risks, and stay ahead of malicious actors.

#### **Vulnerability Check:**

The vulnerability check component of AUTO is designed to systematically assess digital systems for weaknesses that could be exploited by cyber threats. Through automated scans and analysis, the tool **identifies** vulnerabilities, enabling organizations to proactively patch and secure their systems. This proactive approach is crucial in mitigating the risk of cyberattacks and ensuring the resilience of digital infrastructures.

### **2.4.2 CONCLUSION:**

In conclusion, Oreoluwatomiwa Ibikunle's AUTO represents a significant stride in the realm of cybersecurity. By integrating OSINT and vulnerability checks, the tool provides a holistic solution to the multifaceted challenges posed by cyber threats. As digital landscapes evolve, AUTO stands as a testament to the innovative efforts aimed at fortifying our digital defenses.

### **2.4.3 Drawbacks and Future Considerations:**

While AUTO holds promise in enhancing cybersecurity, it is essential to acknowledge potential drawbacks. One such concern may be the ethical implications of extensive OSINT gathering, raising questions about privacy and data protection. Striking a balance between effective threat intelligence and respecting individuals' privacy rights will be a critical consideration for the tool's ongoing development.

Moreover, as cyber threats continually evolve, AUTO must adapt to emerging challenges. Regular updates and continuous improvement will be imperative to ensure the tool remains effective in identifying and mitigating new vulnerabilities. Additionally, collaboration with the cybersecurity community and adherence to industry standards will enhance AUTO's credibility and resilience.

In summary, while AUTO presents a robust solution to contemporary cybersecurity challenges, addressing ethical concerns and staying abreast of evolving threats will be crucial for its sustained success and relevance in an ever-changing digital landscape. Oreoluwatomiwa Ibikunle's innovation is a commendable step forward, and the future holds opportunities for refinement and expansion in the pursuit of a more secure digital environment.

**2.5 IMPLEMENTATION OF AUTOMATED END-TO-END TESTING**

**IN WEB APPLICATIONS**

**Karina Raiküla**

**2.5.1 INTRODUCTION:**

In the ever-evolving landscape of web applications, ensuring the reliability and functionality of software is paramount. One of the key methodologies employed to achieve this is Automated End-To-End Testing. This approach involves automating the entire user flow of a web application, simulating real user interactions and verifying that the application behaves as expected. Karina Raiküla's research delves into the implementation of this testing methodology, exploring its significance in enhancing the quality assurance process.

### **Security in Automated End-To-End Testing:**

Security is a critical aspect that cannot be overlooked in the realm of automated testing. While automated testing provides a robust mechanism for identifying functional defects, it also introduces potential vulnerabilities. The test environment, scripts, and data are all susceptible to security breaches. Raiküla's work meticulously addresses these concerns, shedding light on best practices to secure automated testing frameworks. From secure coding practices in test scripts to safeguarding test data, the research examines how to mitigate the risks associated with implementing automated end-to-end testing in web applications.

#### **2.5.2 Key Security Considerations:**

1. **Access Controls:** Strict access controls must be in place to regulate who can execute tests, access test environments, and view test results. This ensures that only authorized personnel can interact with the testing infrastructure.
2. **Secure Configuration:** The configuration of testing tools and environments must align with security best practices. Raiküla's research provides insights into configuring automated testing setups securely to prevent unauthorized access and potential exploits.
3. **Continuous Monitoring:** Security is an ongoing process. The research advocates for the implementation of continuous monitoring mechanisms to detect and respond to security incidents promptly.

### **2.5.3 CONCLUSION:**

In conclusion, the implementation of Automated End-To-End Testing in web applications, as explored by Karina Raiküla, presents a promising avenue for improving software quality. The research not only highlights the significance of automated testing in ensuring functional reliability but also underscores the importance of addressing security concerns in the testing process. By adopting the recommended security measures, organizations can confidently leverage automated testing to streamline their development pipelines and deliver more robust and secure web applications.

### **2.5.4 Drawbacks of the Given Thesis:**

While Raiküla's research provides invaluable insights into the world of Automated End-To-End Testing, it is essential to acknowledge certain drawbacks:

1. **Resource Intensiveness:** Implementing automated testing frameworks can be resource-intensive, requiring significant time and expertise. Small organizations or those with limited resources may find it challenging to adopt and maintain such a testing infrastructure.
2. **Tool Dependency:** The effectiveness of automated testing often depends on the tools used. Raiküla's research may not fully address the challenges associated with tool compatibility, licensing costs, and the potential need for continuous training of personnel.
3. **Dynamic Environments:** Web applications are subject to frequent updates and changes. Raiküla's thesis may not extensively cover strategies for adapting automated tests to the dynamic nature of web applications, leading to potential test maintenance challenges.
4. **False Positives/Negatives:** Automated testing is not foolproof, and false positives or negatives can occur. The research could provide more in-depth exploration of strategies to minimize these inaccuracies and enhance the reliability of test results.

In acknowledging these drawbacks, it is important to view Raiküla's work as a foundation for further research and refinement, pushing the boundaries of knowledge in the ever-evolving field of automated testing.

**CHAPTER-3**

**EXISTING SYSTEM**

### **3.1 OWASP ZAP (Zed Attack Proxy):**

* **Description:**
  + OWASP ZAP is an open-source security testing tool designed to find vulnerabilities in web applications. It's developed by the Open Web Application Security Project (OWASP).
* **Key Features:**
  + **Automated Scanning:** ZAP includes automated scanners for common vulnerabilities such as SQL injection, XSS, and CSRF.
  + **Manual Testing:** It provides tools for manual testing, allowing security professionals to actively analyze and manipulate requests and responses.
  + **Active and Passive Scanning:** ZAP supports both active scanning (actively sending requests to find vulnerabilities) and passive scanning (analyzing traffic without actively interacting with the application).

### **3.2 Burp Suite:**

* **Description:**
  + Burp Suite is a widely used platform for security testing of web applications, known for its versatility and extensive feature set.
* **Key Features:**
  + **Intercepting Proxy**
  + **Automated Scanning**
  + **Advanced**
  + **Extensive Reporting**

**3.3 Acunetix:**

* **Description:**
  + Acunetix is a web vulnerability scanner designed to automatically scan and audit complex web applications, providing insights into potential security risks.

**Key Features:**

* + **Automated Scanning:** Acunetix performs automated scanning for OWASP Top 10 vulnerabilities and other common security issues.
  + **Deep Scanning:** It supports deep scanning for complex web applications, identifying vulnerabilities that may not be apparent in a surface-level scan.
  + **Integration with CI/CD:** Acunetix can be integrated into Continuous Integration/Continuous Deployment (CI/CD) pipelines for automated security testing in the development process.
  + **Comprehensive Reporting:** The tool offers comprehensive reporting, helping users understand the impact and severity of identified vulnerabilities.

### **3.4 Drawbacks Commonly Found in Such Tools:**

1. **Learning Curve:** Many tools have a steep learning curve, making it challenging for new users to effectively use all the features.
2. **False Positives/Negatives:** Automated scanners may produce false positives or negatives, requiring manual verification and adjustment.
3. **Resource Intensive:** Some tools can be resource-intensive, affecting performance during scanning, especially for large applications.
4. **Complexity:** Comprehensive tools may have a complex interface, which can be overwhelming for users not familiar with web application security testing.
5. **Cost:** While many tools offer free versions, advanced features and scalability often come with a cost.
6. **Integration Challenges:** Integrating these tools into existing development processes can be challenging.

**CHAPTER-4**

**PROPOSED SYSTEM**

**4.1 INTRODUCTION:**

the "GUI Based Web Application Automated Pentester for Top OWASP Vulnerabilities," is a comprehensive web application security testing tool designed to simplify the process of identifying and addressing security vulnerabilities in web applications. By incorporating a graphical user interface (GUI), your tool aims to enhance user experience and accessibility, making it more user-friendly for both experienced security professionals and those new to penetration testing.

**4.2 Features of Project:**

1. **User-Friendly Interface:**

The graphical user interface provides an intuitive platform for users to navigate and interact with the tool's features easily. This design choice aims to reduce the learning curve associated with traditional command-line tools.

1. **Automated Scanning for OWASP Top 10 Vulnerabilities:**

Tool automates the scanning process for the most critical web application vulnerabilities listed in the OWASP Top 10, including SQL injection, XSS, WHOIS, CSRF, and HTTP Tampering. This ensures that users can efficiently identify and address common security issues.

1. **Versatility in Attack Types:**

By supporting a variety of attack types, including SQL injection, XSS, WHOIS, CSRF, and HTTP Tampering, your tool provides users with a versatile solution for testing against a broad range of potential vulnerabilities.

1. **Integrated External Tools:**

The tool seamlessly integrates with external security testing tools such as SQLMap, XXStrike, See-Surf, Bolt, and Smuggler. This integration allows users to leverage the capabilities of well-established tools while benefiting from the convenience of a unified interface.

1. **File Upload and Analysis:**

Your tool offers a feature for users to upload files for analysis, leveraging the VirusTotal API. This functionality enhances the tool's capability to identify potential threats within uploaded files, contributing to a more comprehensive security assessment.

1. **URL Validation:**

The URL validation feature leverages the VirusTotal API to check whether a URL is already in the VirusTotal database and allows users to submit new URLs for analysis. This ensures a proactive approach to identify and address potential security risks associated with URLs.

**4.3 Advantages:**

* **User-Friendly Design**
* **Versatility and Automation**
* **Integration with External Tools**
* **File Upload and URL Validation**

**CHAPTER-5**

**SOFTWARE SPECIFICATIONS**

**5.1 HARDWARE/SOFTWARE REQUIREMENTS**

|  |  |
| --- | --- |
| * Processor | Intel core Processor 2.71GHz |
| * RAM | 4 GB (2GB MINIMUM) |
| * Hard Disk Space | 256MB |
| * GPU | RTX2050 |
| * Internet connection | Stable |
| * Python | Version 3.11.05 |

**CHAPTER-6**

**GUI BASED WEB APPLICATION AUTOMATED PENTESTER FOR TOP OWASP VULNERABILITIES**

**6.1.1 INTRODUCTION:**

In today's digital age, web applications are an integral part of our daily lives, from online shopping platforms to social media networks and financial services. As the reliance on web applications grows, so does the need for robust security measures to protect sensitive data and ensure the integrity and availability of these services. Automated penetration testing of web applications is a crucial component of maintaining a strong security posture, and this process plays a vital role in identifying vulnerabilities, mitigating risks, and ultimately safeguarding both the application and its users.

Web applications, and their user interfaces in particular, represent one of the primary attack surfaces for malicious actors seeking to exploit vulnerabilities. These applications handle a wide range of user data, from personal information to financial transactions, making them attractive targets for cyberattacks. Automated penetration testing of the user interface (UI) is a proactive approach to finding and addressing security weaknesses before they can be exploited by attackers.

**6.1.2 OBJECTIVES**

The primary objectives of UI-Web automated penetration testing are as follows:

1. Vulnerability Identification
2. Risk Assessment:
3. Compliance and Best Practices
4. Automated Testing Efficiency

**6.2 FLOWCHART:**

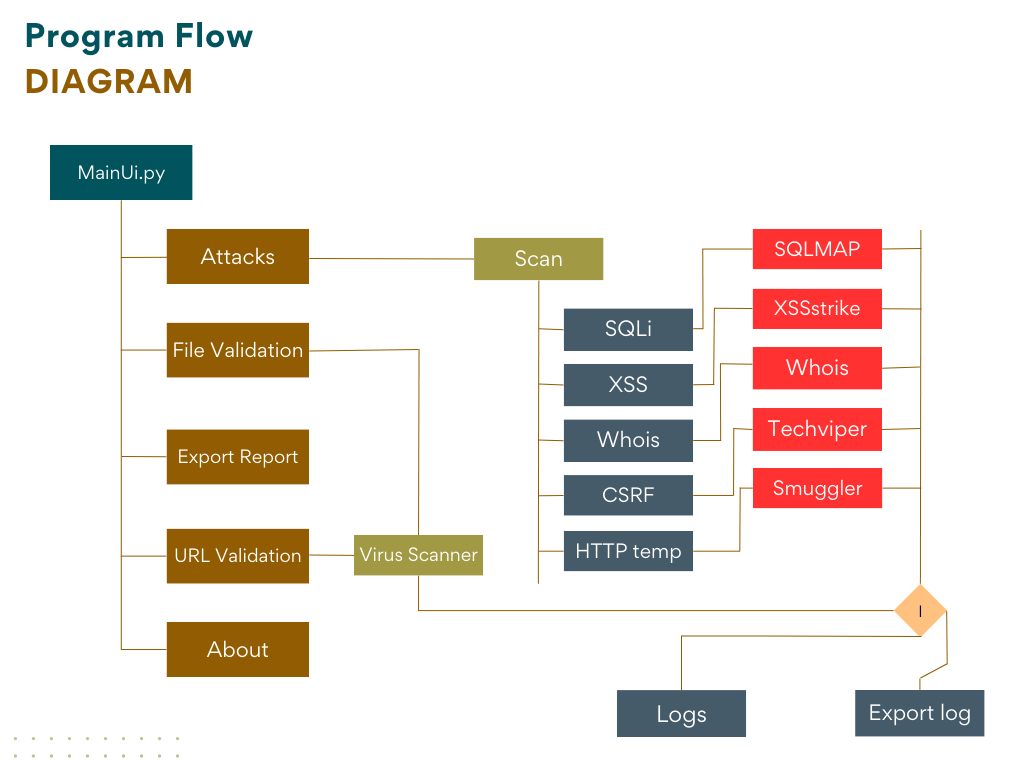


Fig 6.2.1 Flow chart for UI-WAPT

**6.3 HTTP TEMPERING:**

**6.3.1 INTRODUCTION:**

HTTP tampering, often referred to as HTTP request tampering or HTTP response tampering, is a type of cyber attack in which an attacker intercepts and modifies the HTTP (Hypertext Transfer Protocol) requests or responses exchanged between a client (usually a web browser) and a server. HTTP is the foundation of data communication on the World Wide Web, and tampering with it can lead to various security risks and malicious activities.

**6.3.2 COMMON WAYS:**

1. **Request Tampering:** In this scenario, an attacker intercepts the HTTP request sent by a client to a web server. They may modify parameters, headers, or other data within the request to manipulate the server's response.
2. **Response Tampering:** Attackers can also intercept the HTTP response from a web server to a client. They may modify the content of the response, inject malicious scripts or malware, or tamper with cookies, leading to various attacks like Cross-Site Scripting (XSS), data injection, or session hijacking.
3. **Man-in-the-Middle (MitM) Attacks:** HTTP tampering is often facilitated by MitM attacks, where an attacker intercepts and monitors the communication between a client and a server. By doing this, they can tamper with both requests and responses as they pass through, enabling various attacks.
4. **Cookie Tampering:** Cookies are commonly used for session management and user authentication in web applications. Attackers can tamper with cookies by intercepting and modifying them to gain unauthorized access to user accounts or manipulate application behavior.

**6.3.3 MEASURES TO PROTECT:**

1. **HTTPS (HTTP Secure):** Use HTTPS to encrypt the communication between clients and servers, making it more difficult for attackers to intercept and tamper with data.
2. **Input Validation:** Implement input validation and sanitization on the server side to prevent malicious input from affecting the application.
3. **Content Security Policies (CSP):** Use CSP headers to control which scripts and resources can be executed or loaded on a web page, reducing the risk of XSS attacks.
4. **Secure Cookies:** Set secure and HttpOnly flags on cookies to prevent tampering. Secure cookies can only be transmitted over HTTPS, and HttpOnly cookies are inaccessible to JavaScript, reducing the risk of XSS attacks.
5. **Transport Layer Security (TLS):** Employ strong TLS configurations to ensure secure encryption during data transmission.
6. **Web Application Firewalls (WAFs):** WAFs can help filter and block malicious traffic and requests, including those related to HTTP tampering.
7. **Monitoring and Intrusion Detection:** Implement real-time monitoring and intrusion detection systems to detect and respond to suspicious activities.

**6.4 CROSS SITE SCRIPTING:**

**6.4.1 INTRODUCTION:**

Cross-Site Scripting (XSS) is a type of security vulnerability or attack that occurs when an attacker injects malicious scripts (usually JavaScript) into web pages viewed by other users. The main objective of an XSS attack is to execute the injected script within the context of a victim's web browser, allowing the attacker to steal information, manipulate the web page, or perform other malicious actions on behalf of the victim.

**6.4.2 TYPES:**

**1. Stored XSS (Persistent XSS):** In this type of attack, the malicious script is permanently stored on the target web server. When a user visits a page containing the injected script, it is retrieved from the server and executed in the context of the user's browser. This type of XSS attack can have long-lasting effects and can affect multiple users who access the compromised page.

**2. Reflected XSS (Non-Persistent XSS): Reflected** XSS occurs when the injected script is reflected off a web server or another source and is immediately executed in the user's browser. It is typically delivered through a URL or input fields. The attacker often entices victims to click on a specially crafted link or visit a particular web page. Unlike stored XSS, the malicious script is not permanently stored on the server and only affects users who access the manipulated URL.

**3. DOM-based XSS:** DOM (Document Object Model) XSS is a variant of XSS in which the attack is performed entirely on the client side, and the malicious script manipulates the Document Object Model of a web page. The attacker typically exploits vulnerable client-side code or JavaScript frameworks to execute the attack.

**6.4.3 MEASURES TO PREVENT:**

**1. Input Validation and Sanitization:** Validate and sanitize user inputs on the server side to prevent any untrusted data from being included in web pages. Use libraries and frameworks that provide protection against XSS, such as Content Security Policy (CSP).

**2. Output Encoding:** Encode or escape data before rendering it in HTML, JavaScript, or other contexts. This makes it difficult for attackers to inject malicious code.

**3. Content Security Policy (CSP):** Implement CSP headers to specify which sources of content are allowed to be loaded and executed on a web page. CSP can block or report attempts at executing malicious scripts.

**4. Use HTTPS:** Ensure that your web application uses HTTPS to encrypt data in transit, reducing the risk of man-in-the-middle attacks that might lead to XSS.

**5. Web Application Firewalls (WAFs):** Employ a WAF to filter and block malicious requests, including those that may be related to XSS.

XSS attacks can have serious consequences, including data theft, session hijacking, defacement of web pages, and more. It's crucial for web developers and administrators to be aware of XSS vulnerabilities and take measures to prevent them in their applications.

**6.5 SQL INJECTION:**

**6.5.1 INTRODUCTION**

SQL Injection (SQLi) is a type of security vulnerability and cyber attack that occurs when an attacker is able to manipulate a web application's SQL query through user inputs. This vulnerability arises when user inputs, which are not properly validated or sanitized, are directly incorporated into SQL queries used by the application to interact with a database. By injecting malicious SQL code, attackers can gain unauthorized access to the database, view, modify, or delete data, and potentially execute administrative tasks on the database server.

**6.5.2 SAMPLE QUERIES:**

```sql

SELECT \* FROM products WHERE product\_name = '<user\_input>';

```

If the application doesn't properly validate or sanitize the user input and an attacker enters the following input:

```plaintext

' OR '1'='1

```

The SQL query would become:

```sql

SELECT \* FROM products WHERE product\_name = '' OR '1'='1';

```

In this case, the attacker's input effectively turns the query into a true statement ('1'='1'), which results in the application retrieving all the records from the "products" table, potentially exposing sensitive data.

**6.5.3 MEASURES TO PREVENT:**

**1. Parameterized Statements (Prepared Statements):** Instead of embedding user inputs directly into SQL queries, use parameterized statements provided by your programming language or framework. These statements automatically handle the proper escaping and quoting of user inputs.

**2. Input Validation and Sanitization:** Ensure that user inputs are validated and sanitized on the server side to eliminate or neutralize any potentially malicious input.

**3. Least Privilege Principle:** Ensure that your application's database connections have the least privilege necessary to perform their tasks. Avoid using superuser accounts for application access.

**4. Web Application Firewall (WAF):** Implement a WAF to filter and block malicious requests, including those attempting SQL injection.

**5. Database Security:** Keep your database software and servers up to date, apply security patches, and use security configurations to minimize potential attack vectors.

**6. Error Handling:** Avoid displaying detailed error messages to users, as these messages may provide valuable information to attackers. Log error messages internally for debugging purposes.

**7. Regular Security Audits and Penetration Testing:** Periodically audit your application's code and perform penetration testing to identify and fix vulnerabilities, including potential SQL injection issues.

SQL injection can have severe consequences, including data breaches, data manipulation, and potentially complete compromise of a web application and its associated database.

**6.6 USER INTERFACE:**

**6.6.1 INTRODUCTION:**

**Tool: Tkinter**

Tkinter is a standard Python library (included with most Python installations) for creating graphical user interfaces (GUIs). It provides a set of tools and widgets for building windows, dialog boxes, buttons, menus, and other GUI elements for your Python applications. Tkinter is based on the Tk GUI toolkit, which is widely used for building GUI applications.

**6.6.2 MAIN CODE:**

import tkinter as tk

from tkinter import ttk

from tkinter import filedialog

from tkinter import scrolledtext

from ttkthemes import ThemedStyle

import subprocess

import sys

import json

from bs4 import BeautifulSoup

import requests

class Vulnerability:

def \_\_init\_\_(self, name, severity, description):

self.name = name

self.severity = severity

self.description = description

class VulnerabilityScanner:

def \_\_init\_\_(self, root):

self.root = root

self.root.title('Web Vulnerability Scanner')

self.root.geometry('800x550')

self.tabs = ttk.Notebook(root)

self.tab1 = ttk.Frame(self.tabs)

self.tab2 = ttk.Frame(self.tabs)

self.tab3 = ttk.Frame(self.tabs)

self.tab4 = ttk.Frame(self.tabs)

self.tab6 = ttk.Frame(self.tabs)

self.create\_scan\_tab()

self.create\_upload\_tab()

self.create\_report\_tab()

self.create\_about\_tab()

self.create\_url\_validation\_tab()

self.tabs.add(self.tab1, text="Scan")

self.tabs.add(self.tab2, text="Upload Files")

self.tabs.add(self.tab3, text="Report")

self.tabs.add(self.tab6, text="URL Validation")

self.tabs.add(self.tab4, text="About")

self.tabs.pack(fill="both", expand=True)

def create\_scan\_tab(self):

self.url\_label = ttk.Label(self.tab1, text='Enter URL:')

self.url\_edit = ttk.Entry(self.tab1)

self.scan\_button = ttk.Button(self.tab1, text='Scan', command=self.scan)

self.log\_box = scrolledtext.ScrolledText(self.tab1, wrap=tk.WORD)

self.export\_button = ttk.Button(self.tab1, text='Export Log')

self.attacks\_label = ttk.Label(self.tab1, text='Select Attacks:')

# Create a frame to contain the checkboxes on the left

checkbox\_frame = ttk.Frame(self.tab1)

checkbox\_frame.grid(row=2, column=0, padx=5, pady=5, sticky='w')

self.attack\_checkboxes = {

'SQL Injection': tk.IntVar(),

'XSS': tk.IntVar(),

'WHOIS': tk.IntVar(),

'CSRF': tk.IntVar(),

'HTTP Tampering': tk.IntVar()

}

# Arrange checkboxes on the left side

for i, (attack, var) in enumerate(self.attack\_checkboxes.items()):

ttk.Checkbutton(checkbox\_frame, text=attack, variable=var).grid(row=i, column=0, padx=5, pady=5, sticky='w')

self.url\_label.grid(row=0, column=1, padx=5, pady=5, sticky='w')

self.url\_edit.grid(row=0, column=2, padx=5, pady=5)

self.scan\_button.grid(row=0, column=3, padx=5, pady=5)

self.attacks\_label.grid(row=1, column=1, padx=5, pady=5, sticky='w')

self.log\_box.grid(row=2, column=1, columnspan=3, padx=5, pady=5)

self.export\_button.grid(row=3, column=1, padx=5, pady=5, sticky='w')

def create\_upload\_tab(self):

self.file\_label = ttk.Label(self.tab2, text='Select File for Analysis:')

self.file\_edit = ttk.Entry(self.tab2)

self.browse\_button = ttk.Button(self.tab2, text='Browse', command=self.browse\_file)

self.upload\_button = ttk.Button(self.tab2, text='Upload and Analyze', command=self.upload\_and\_analyze)

self.upload\_status = tk.StringVar()

upload\_status\_label = ttk.Label(self.tab2, textvariable=self.upload\_status)

self.file\_label.grid(row=0, column=0, padx=5, pady=5, sticky='w')

self.file\_edit.grid(row=0, column=1, padx=5, pady=5)

self.browse\_button.grid(row=0, column=2, padx=5, pady=5)

self.upload\_button.grid(row=1, column=0, padx=5, pady=5, sticky='w')

upload\_status\_label.grid(row=1, column=1, columnspan=2, padx=5, pady=5, sticky='w')

def create\_report\_tab(self):

self.report\_button = ttk.Button(self.tab3, text='Generate Report', command=self.generate\_report)

self.report\_log = scrolledtext.ScrolledText(self.tab3, wrap=tk.WORD)

self.export\_report\_button = ttk.Button(self.tab3, text='Export Report', command=self.export\_report)

self.report\_button.grid(row=0, column=0, padx=5, pady=5, sticky='w')

self.report\_log.grid(row=1, column=0, columnspan=3, padx=5, pady=5)

self.export\_report\_button.grid(row=2, column=1, padx=5, pady=5, sticky='w')

def create\_about\_tab(self):

about\_text = """

Developed by Dekode Security Team

V.SHANMUGAM

R.KAPILSURYA

R.SARANKUMAR

4th Year Cyber Security

Paavai Engineering College

"""

about\_label = ttk.Label(self.tab4, text=about\_text, font=('Helvetica', 14))

about\_label.grid(padx=20, pady=20)

def create\_live\_ip\_monitor\_tab(self):

self.ip\_label = ttk.Label(self.tab5, text='Enter IP Address:')

self.ip\_edit = ttk.Entry(self.tab5)

self.search\_button = ttk.Button(self.tab5, text='Search IP Details', command=self.search\_ip\_details)

self.ip\_details\_log = scrolledtext.ScrolledText(self.tab5, wrap=tk.WORD)

self.ip\_label.grid(row=0, column=0, padx=5, pady=5, sticky='w')

self.ip\_edit.grid(row=0, column=1, padx=5, pady=5)

self.search\_button.grid(row=1, column=0, padx=5, pady=5, sticky='w')

self.ip\_details\_log.grid(row=2, column=0, columnspan=3, padx=5, pady=5)

def scan(self):

url = self.url\_edit.get()

selected\_attacks = [attack for attack, var in self.attack\_checkboxes.items() if var.get() == 1]

vulnerabilities = self.check\_vulnerabilities(url, selected\_attacks)

self.log\_vulnerabilities(vulnerabilities)

if 'SQL Injection' in selected\_attacks:

self.run\_sqlmap(url)

if 'XSS' in selected\_attacks:

self.run\_xxs\_strike(url)

if 'WHOIS' in selected\_attacks:

self.run\_see\_surf(url)

if 'CSRF' in selected\_attacks:

self.run\_bolt(url)

if 'HTTP Tampering' in selected\_attacks:

self.run\_smuggler\_master(url)

def run\_xxs\_strike(self, url):

path = r'C:\Users\shanm\OneDrive\Desktop\Project WEB PENETRATER\XSStrike-master\xsstrike.py'

xxs\_strike\_command = [sys.executable, path, '-u', url]

try:

xxs\_strike\_output = subprocess.check\_output(xxs\_strike\_command, universal\_newlines=True)

self.log\_box.insert(tk.END, f'XXStrike Output:\n{xxs\_strike\_output}\n')

except subprocess.CalledProcessError as e:

self.log\_box.insert(tk.END, f'Error running XXStrike: {e}\n')

def run\_sqlmap(self, url):

path = r'C:\Users\shanm\OneDrive\Desktop\Project WEB PENETRATER\sqlmap-master\sqlmap.py'

sqlmap\_command = [sys.executable, path, '-u', url]

try:

sqlmap\_output = subprocess.check\_output(sqlmap\_command, universal\_newlines=True)

self.log\_box.insert(tk.END, f'SQLMap Output:\n{sqlmap\_output}\n')

except subprocess.CalledProcessError as e:

self.log\_box.insert(tk.END, f'Error running SQLMap: {e}\n')

def run\_see\_surf(self, url):

path=r'C:\Users\shanm\OneDrive\Desktop\Project WEB PENETRATER\whois\whoislookup.py'

see\_surf\_command = [sys.executable, path, '-u', url]

try:

see\_surf\_output = subprocess.check\_output(see\_surf\_command, universal\_newlines=True)

self.log\_box.insert(tk.END, f'See-Surf Output:\n{see\_surf\_output}\n')

except subprocess.CalledProcessError as e:

self.log\_box.insert(tk.END, f'Error running See-Surf: {e}\n')

def run\_bolt(self, url):

path1= r'C:\Users\shanm\OneDrive\Desktop\Project WEB PENETRATER\TechViper-main\TechViper.py'

bolt\_command = ['python', path1, '-u' ,url]

try:

bolt\_output = subprocess.check\_output(bolt\_command, universal\_newlines=True, stderr=subprocess.STDOUT)

self.log\_box.insert(tk.END, f'bolt Master Output:\n{bolt\_output}\n')

except subprocess.CalledProcessError as e:

self.log\_box.insert(tk.END, f'Error running bolt Master: {e}\n')

def run\_smuggler\_master(self, url):

path= r'C:\Users\shanm\OneDrive\Desktop\Project WEB PENETRATER\smuggler-master\smuggler-master\smuggler.py'

smuggler\_master\_command = [sys.executable, path, '-u' ,url]

try:

smuggler\_master\_output = subprocess.check\_output(smuggler\_master\_command, universal\_newlines=True)

self.log\_box.insert(tk.END, f'Smuggler Master Output:\n{smuggler\_master\_output}\n')

except subprocess.CalledProcessError as e:

self.log\_box.insert(tk.END, f'Error running Smuggler Master: {e}\n')

def check\_vulnerabilities(self, url, selected\_attacks):

# Existing vulnerability check logic

vulnerabilities = [

#Vulnerability('SQL Injection', 'High', 'This is an SQL injection vulnerability.'),

#Vulnerability('XSS', 'Medium', 'This is a Cross-Site Scripting vulnerability.'),

]

return vulnerabilities

def log\_vulnerabilities(self, vulnerabilities):

self.log\_box.delete(1.0, tk.END)

for vuln in vulnerabilities:

self.log\_box.insert(tk.END,

f'Vulnerability: {vuln.name}\nSeverity: {vuln.severity}\nDescription: {vuln.description}\n\n')

def export\_log(self):

log\_text = self.log\_box.get(1.0, tk.END)

with open('vulnerability\_log.txt', 'w') as log\_file:

log\_file.write(log\_text)

def browse\_file(self):

file\_path = filedialog.askopenfilename(filetypes=[('All Files', '\*.\*')])

self.file\_edit.delete(0, tk.END)

self.file\_edit.insert(0, file\_path)

def create\_upload\_tab(self):

self.file\_label = ttk.Label(self.tab2, text='Select File for Analysis:')

self.file\_edit = ttk.Entry(self.tab2)

self.browse\_button = ttk.Button(self.tab2, text='Browse', command=self.browse\_file)

self.upload\_button = ttk.Button(self.tab2, text='Upload and Analyze', command=self.upload\_and\_analyze)

self.upload\_status = tk.StringVar()

upload\_status\_label = ttk.Label(self.tab2, textvariable=self.upload\_status)

# ScrolledText widget to display the log for upload and analyze action

self.upload\_log = scrolledtext.ScrolledText(self.tab2, wrap=tk.WORD)

self.file\_label.grid(row=0, column=0, padx=5, pady=5, sticky='w')

self.file\_edit.grid(row=0, column=1, padx=5, pady=5)

self.browse\_button.grid(row=0, column=2, padx=5, pady=5)

self.upload\_button.grid(row=1, column=0, padx=5, pady=5, sticky='w')

upload\_status\_label.grid(row=1, column=1, columnspan=2, padx=5, pady=5, sticky='w')

# Grid the upload log widget below the button

self.upload\_log.grid(row=2, column=0, columnspan=3, padx=5, pady=5)

def upload\_and\_analyze(self):

file\_path = self.file\_edit.get()

if not file\_path:

self.upload\_status.set("Please select a file to analyze.")

return

try:

api\_key = '4d8f690e8cbae5ad547e9a9712a0713b38206dfd1b9d0ecdaaa512d6817e37fa'

# Upload the file to VirusTotal

url = 'https://www.virustotal.com/vtapi/v2/file/scan'

params = {'apikey': api\_key}

files = {'file': (file\_path, open(file\_path, 'rb'))}

response = requests.post(url, files=files, params=params)

if response.status\_code == 200:

result = response.json()

resource = result.get('resource')

# Check the analysis report for the uploaded file

report\_url = f'https://www.virustotal.com/gui/file/{resource}/detection'

# Fetch the analysis report

report\_api\_url = f'https://www.virustotal.com/vtapi/v2/file/report'

report\_params = {'apikey': api\_key, 'resource': resource}

report\_response = requests.get(report\_api\_url, params=report\_params)

if report\_response.status\_code == 200:

report\_data = report\_response.json()

analysis\_output = json.dumps(report\_data, indent=4)

self.upload\_log.delete(1.0, tk.END)

self.upload\_log.insert(tk.END, analysis\_output)

self.upload\_status.set(f"File uploaded to Successfully")

else:

self.upload\_status.set("Error fetching the report from VirusTotal.")

else:

self.upload\_status.set("Error uploading the file to VirusTotal.")

self.upload\_log.delete(1.0, tk.END)

self.upload\_log.insert(tk.END, "Error uploading the file to VirusTotal.")

except Exception as e:

self.upload\_status.set(f"Check your connection / System not connected to internet")

self.upload\_log.delete(1.0, tk.END)

self.upload\_log.insert(tk.END, f"Error: {str(e)}")

def generate\_report(self):

log\_text = self.log\_box.get(1.0, tk.END)

self.report\_log.delete(1.0, tk.END)

self.report\_log.insert(tk.END, log\_text)

def export\_report(self):

report\_text = self.report\_log.get(1.0, tk.END)

with open('vulnerability\_report.txt', 'w') as report\_file:

report\_file.write(report\_text)

def create\_url\_validation\_tab(self):

self.url\_validate\_label = ttk.Label(self.tab6, text='Enter URL to Validate:')

self.url\_validate\_edit = ttk.Entry(self.tab6)

self.validate\_button = ttk.Button(self.tab6, text='Validate URL', command=self.validate\_url)

self.validation\_log = scrolledtext.ScrolledText(self.tab6, wrap=tk.WORD)

self.validation\_log.grid(row=1, column=0, columnspan=3, padx=5, pady=5)

self.url\_validate\_label.grid(row=0, column=1, padx=5, pady=5, sticky='w')

self.url\_validate\_edit.grid(row=0, column=2, padx=5, pady=5)

self.validate\_button.grid(row=0, column=3, padx=5, pady=5)

def validate\_url(self):

url = self.url\_validate\_edit.get()

if not url:

self.validation\_log.delete(1.0, tk.END)

self.validation\_log.insert(tk.END, 'Please enter a URL to validate.')

return

try:

api\_key = '4d8f690e8cbae5ad547e9a9712a0713b38206dfd1b9d0ecdaaa512d6817e37fa'

# Check if the URL is already in the database

url\_report\_url = 'https://www.virustotal.com/vtapi/v2/url/report'

url\_report\_params = {'apikey': api\_key, 'resource': url}

url\_report\_response = requests.get(url\_report\_url, params=url\_report\_params)

url\_report\_data = url\_report\_response.json()

if url\_report\_data['response\_code'] == 1:

self.validation\_log.delete(1.0, tk.END)

self.validation\_log.insert(tk.END, 'This URL is already in the VirusTotal database. Here is the analysis report:\n')

# Extract and format the report data

report = url\_report\_data['verbose\_msg']

scan\_results = url\_report\_data.get('scans', {})

for engine, result in scan\_results.items():

report += f"\nEngine: {engine}, Result: {result['result']}"

self.validation\_log.insert(tk.END, report)

else:

# Submit the URL for analysis

url\_scan\_url = 'https://www.virustotal.com/vtapi/v2/url/scan'

url\_scan\_params = {'apikey': api\_key, 'url': url}

url\_scan\_response = requests.post(url\_scan\_url, data=url\_scan\_params)

url\_scan\_data = url\_scan\_response.json()

if url\_scan\_data['response\_code'] == 1:

self.validation\_log.delete(1.0, tk.END)

self.validation\_log.insert(tk.END, 'The URL has been successfully submitted for analysis. You can check the report later.\n')

else:

self.validation\_log.delete(1.0, tk.END)

self.validation\_log.insert(tk.END, 'Error submitting the URL for analysis.\n')

except Exception as e:

self.validation\_log.delete(1.0, tk.END)

self.validation\_log.insert(tk.END, f'Error: {str(e)}\n')

if \_\_name\_\_ == '\_\_main\_\_':

root = tk.Tk()

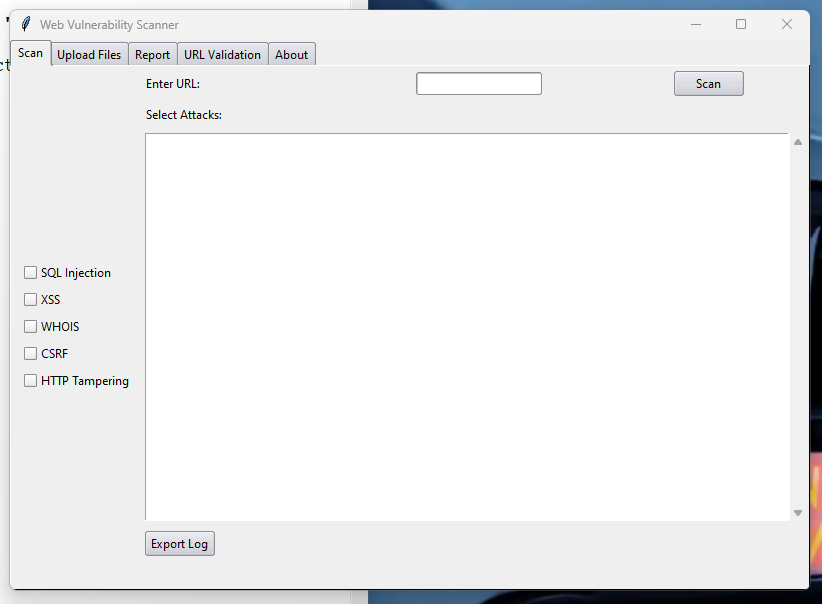
# Create a themed style for ttk widgets

style = ThemedStyle(root)

style.set\_theme("plastik") # You can change the theme here (e.g., "clam", "alt", "plastik", etc.)

app = VulnerabilityScanner(root)

root.mainloop()

**6.6.4 OUTPUT:**

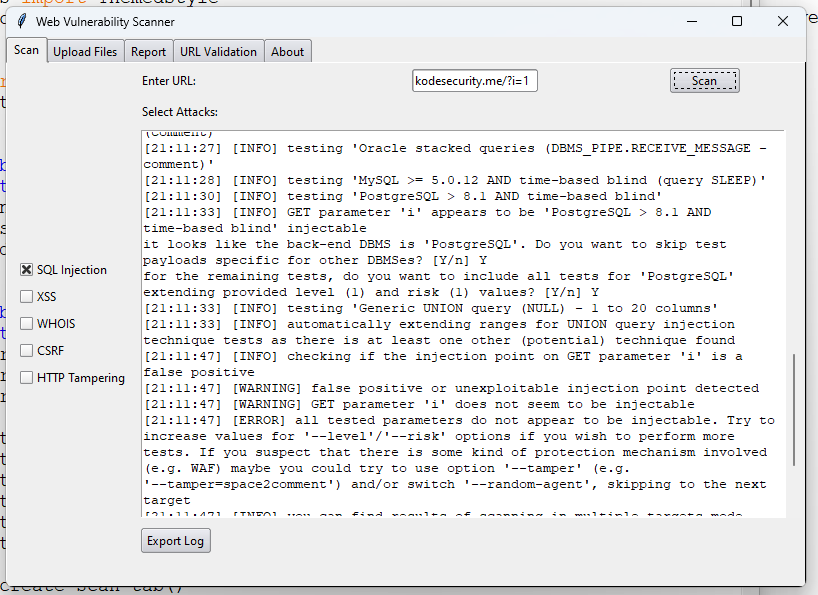
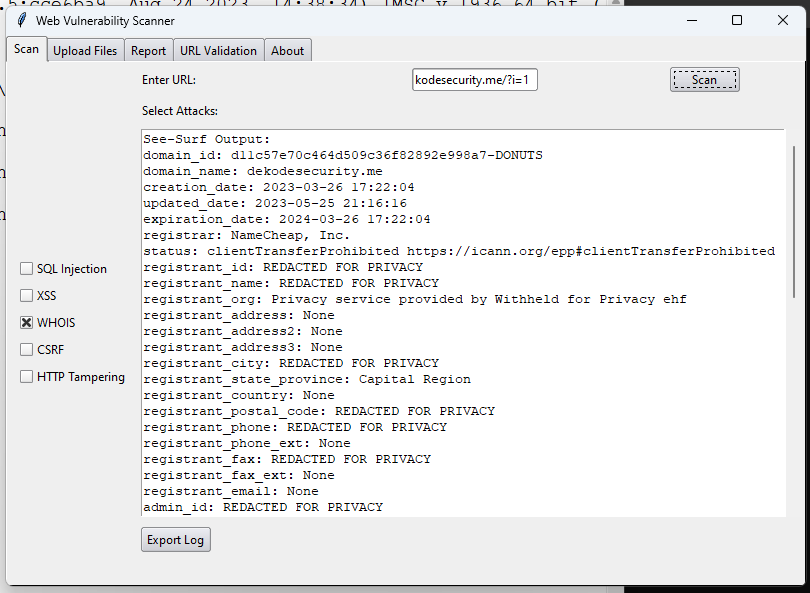
Fig 6.6.1 Home page

Fig 6.6.2 SQLi page

Fig 6.6.3 WHO-Is page

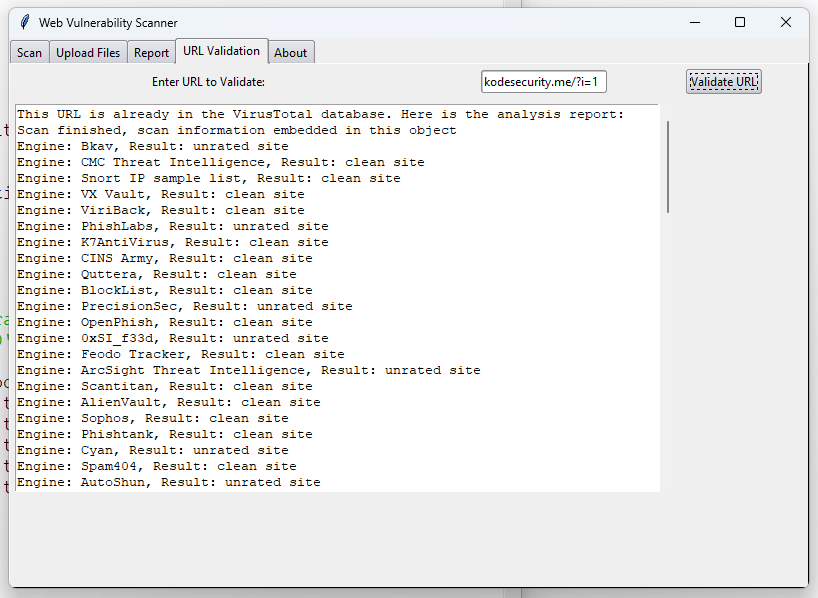


Fig 6.6.4 URL Validation Page

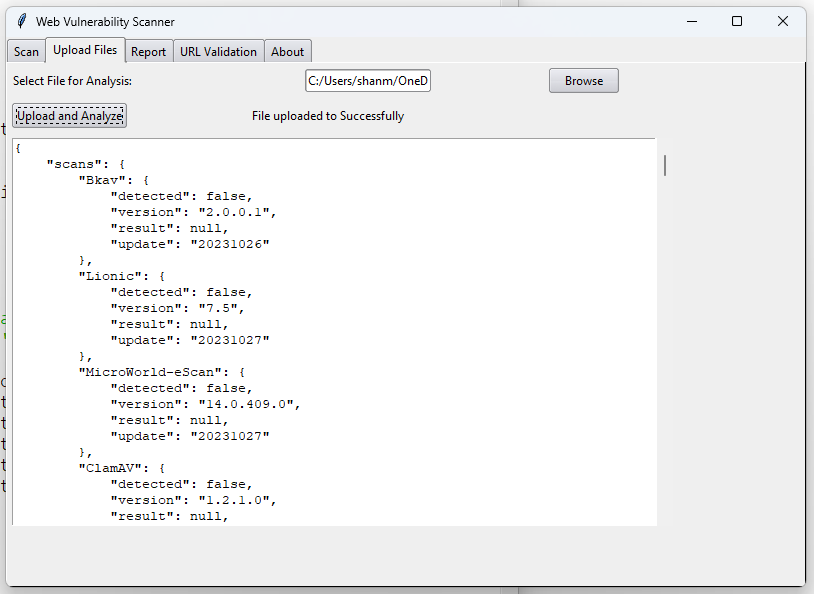


Fig 6.6.5 Upload and Analysis Page

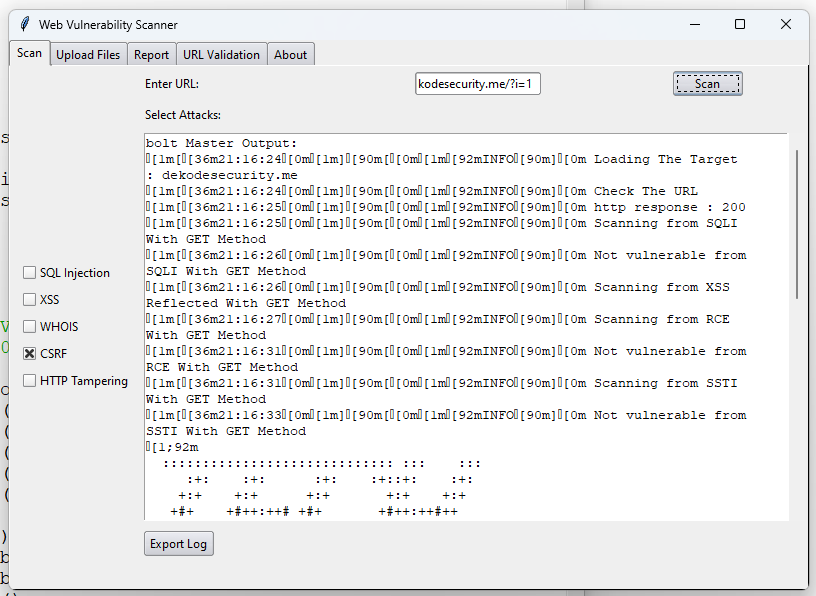
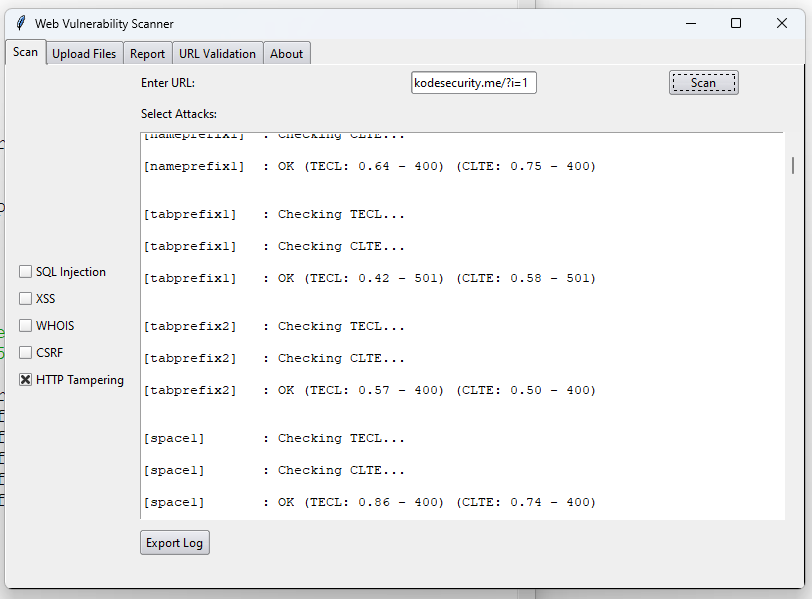


Fig 6.6.6 CSRF Page

Fig 6.6.7 HTTP Tampering Page

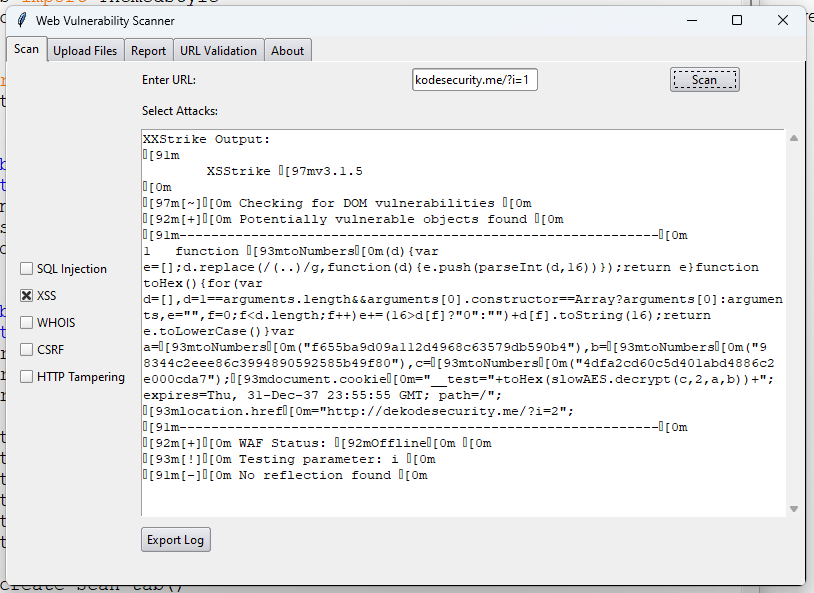
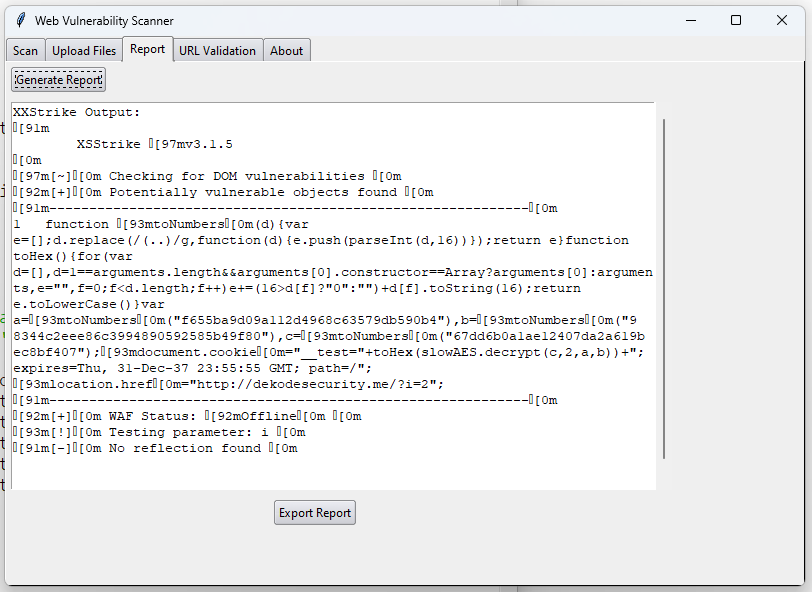
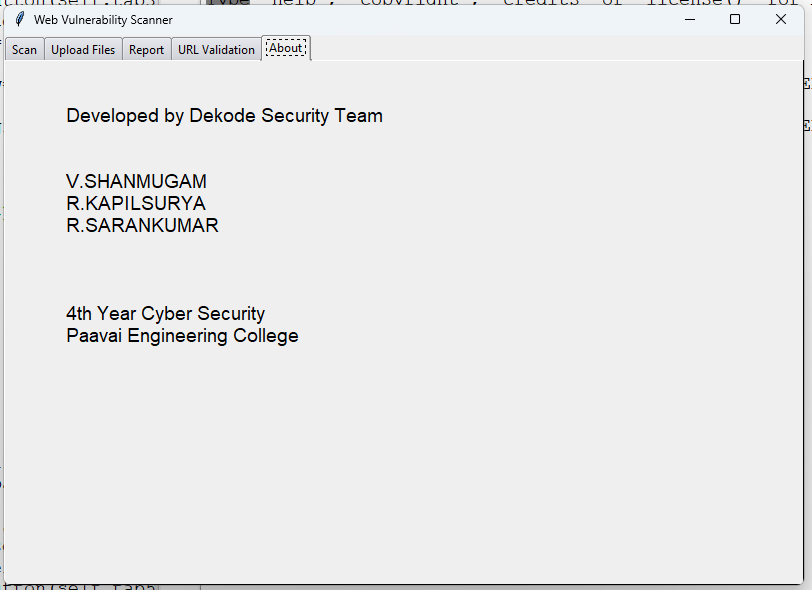


Fig 6.6.8 XSS page



6.6.9 Report Generating Page



6.6.10 About Page

**CHAPTER-7**

**CONCLUSION**

**7.1 CONCLUSION**

Penetration testing, or pen testing, is a critical process in assessing the security of a web application's user interface (UI). After conducting a thorough assessment, it is essential to draw meaningful conclusions that can inform decision-making, prioritize security improvements, and enhance the overall security posture of the web application.

* Identification of Vulnerabilities
* Severity Assessment
* Common Vulnerabilities
* User Data Protection
* Authentication and Authorization
* Recommendations for Mitigation
* Testing Coverage
* Compliance and Best Practices
* Continuous Improvement
* Post-Testing Actions

In conclusion, the UI web application penetration testing has successfully identified vulnerabilities and provided a roadmap for enhancing security. It is crucial to take the findings seriously and prioritize remediation efforts to protect the application, its users, and the organization from potential security threats. Security is an ongoing journey, and it's important to continually assess and improve the security posture of the web application to stay ahead of emerging threats.

**7.2 ADVANTAGES:**

Automated penetration testing for web application UI offers several advantages:

1. **Scalability:** Automation allows for testing at scale, making it suitable for large and complex web applications with numerous features and functionalities.
2. **Efficiency:** Automated tools can quickly and thoroughly scan the application, identifying vulnerabilities that might be challenging to find through manual testing alone.
3. **Consistency:** Automated tests can be executed consistently and repeatedly, reducing the risk of human error in the testing process.
4. **Timeliness:** Automated testing can be integrated into the development and deployment pipeline, providing real-time feedback to development teams, allowing for swift remediation.
5. **Cost-Effectiveness:** While there may be an initial investment in tools and expertise, automated testing can be more cost-effective in the long run compared to manual testing, especially for applications that undergo frequent updates and changes.

**REFERENCES**

|  |  |
| --- | --- |
|  | Karina Raiküla., April,2023 “Implementation of Automated End-To-End Testing in Web Applications” , Metropolia. |
|  | Lari Harjunen., June,2023 “Optimizing Web Application Security Testing Process in the Financial Sector” , LAUREA. |
|  | Shaik Heena, Bingi Naga Lalitha, Devarasetty Sai Umesh , April,2023 , “Machine learning for web vulnerability detection: the case of cross-site request forgery” , Industrial Engineering Journal. |
|  | Aman Kumar Singh, Gulshan Kumar, Prof. Dr. Prabhat Kumar Vishwakarma , May,2023 , “Automation in manual penetration testing usingbash shell script” , International Research Journal of Modernization in Engineering Technology and Science. |
|  | Aditya Kurniawan1, Mohamad Yusof Darus, Muhammad Azizi Mohd Ariffin,Yohan Muliono1 and Chrisando Ryan Pardomuan , 2022 , “Automation of Quantifying Security Risk Level on Injection Attacks Based on Common Vulnerability Scoring System Metric” , Science and Technology. |
|  | Baillie, C., Standen, M., Schwartz, J., Docking, M., Bowman, D., & Kim, J. (2020) “Cyborg: An autonomous cyber operations research gym”, arXiv preprint arXiv:2002.10667, 2020. |
|  | Brown, A., & Wilson, D. (2018) “Integrating Vulnerability Analysis and Penetration Testing in Cybersecurity Assessments “ |
|  | Chandak, Y., Theocharous, G., Kostas, J., Jordan, S. M., & Thomas, P. S. (2019)“Learning action representations for reinforcement learning” , CoRR, abs/1902.00183, 2019. |
|  | Doe, J., & Smith, S. (2019) “Automating Penetration Testing: A Survey “ , Published on 2019. |
|  | Fabio Massimo Zennaro and Laszlo Erdodi. (2022) , “ Modeling penetration testing with reinforcement learning using capture-the-flag challenges and tabular q-learning. “ , arXiv preprint arXiv:2005.12632 |
|  | Ghanem, M. C., & Chen, T. M. (2020) “ Reinforcement learning for efficient network penetration testing “ ,Published in 2020. |
|  | Hu, Z., Beuran, R., & Tan, Y. (2020) “Automated penetration testing using deep reinforcement learning “ , In 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), pages 2–10. IEEE, 2020. |
|  | Lee, M., & Clark, E. (2020) “ A Machine Learning Approach for Automated Vulnerability Analysis and Penetration Testing “ , Published on 2020. |
|  | Moore, C., & Anderson, S. (2021) “ Towards Continuous Automated Penetration Testing in Cloud Environments “ , Published on 2021. |
|  | Roberts, A., & White, J. (2020) “ Vulnerability Analysis and Automated Penetration Testing in Industrial Control Systems “ , Published on 2020. |
|  | Schwartz, J., & Kurniawati, H. (2019) “ Autonomous penetration testing using reinforcement learning” , Published in 2019. |
|  | Zennaro, F. M., & Erdodi, L. (2022) “ Modeling penetration testing with reinforcement learning using capture-the-flag challenges and tabular q-learning “ , arXiv preprint arXiv:2005.12632, 2022. |
|  | Christopher Farquhar, Laura Gustafson, Zeming Lin, Shimon Whiteson, Nicolas Usunier, and Gabriel Synnaeve. (2019) “ Growing action spaces. “ , CoRR, abs/1906.12266, 2019. |
|  | Mohamed C. Ghanem and Thomas M. Chen. (2020) “ Reinforcement learning for efficient network penetration testing. “ , Published in 2020. |
|  | Zhenguo Hu, Razvan Beuran, and Yasuo Tan. (2020) ‘ Automated penetration testing using deep reinforcement learning. “ , In 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), pages 2–10. IEEE, 2020. |