**B.Tech. BCSE497J - Project-I**

**AUTOMATING RECONNAISSANCE IN PENETRATION TESTING FOR WEB APPLICATIONS**

*Submitted in partial fulfillment of the requirements for the degree of*

**Bachelor of Technology**

*in*

**Programme**

*By*

|  |  |
| --- | --- |
| **21BCE2331** | **AAYUSH AMRITESH** |
| **21BCE2563** | **SHREYANSH SAURAV** |
| **21BDS0121** | **AARYAK AGARWAL** |

## Under the Supervision of

|  |
| --- |
| **Dr. ISLABUDEEN M.** |
| Assistant Professor Senior Grade 2 |
| School of Computer Science and Engineering (SCOPE) |
|  |



November 2024

**DECLARATION**

We hereby declare that the project entitled ‘Automating Reconnaissance in Penetration Testing for Web Applications’ submitted by us, for the award of the degree of *Bachelor of Technology in Computer Science and Engineering* to VIT is a record of bonafide work carried out by me under the supervision of Dr. Islabudeen M.

We further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place : Vellore Date :

**Signature of the Candidate**

**CERTIFICATE**

This is to certify that the project entitled ‘Automating Reconnaissance in Penetration Testing for Web Applications’ submitted by Aayush Amritesh (21BCE2331), Shreyansh Saurav (21BCE2563) and Aaryak Agarwal (21BDS0121), **School of Computer Science and Engineering**, VIT, for the award of the degree of *Bachelor of Technology in Computer Science and Engineering*, is a record of bonafide work carried out by them under my supervision during Fall Semester 2024-2025, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The project fulfills the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Place : Vellore

Date :

**Signature of the Guide**

**Examiner(s)**

**Dr. K.S. Umadevi**

**Bachelor of Technology**

**ACKNOWLEDGEMENTS**

We are deeply grateful to the management of Vellore Institute of Technology (VIT) for providing us with the opportunity and resources to undertake this project. Their commitment to fostering a conducive learning environment has been instrumental in my academic journey. The support and infrastructure provided by VIT have enabled me to explore and develop my ideas to their fullest potential.

Our sincere thanks to Dr. Ramesh Babu K, the Dean of the School of Computer Science and Engineering (SCOPE), for his unwavering support and encouragement. His leadership and vision have greatly inspired me to strive for excellence. The Dean’s dedication to academic excellence and innovation has been a constant source of motivation for me. We appreciate his efforts in creating an environment that nurtures creativity and critical thinking.

We express our profound appreciation to Dr. K.S. Umadevi, the Head of the Software Systems, for her insightful guidance and continuous support. Her expertise and advice have been crucial in shaping the direction of our project. The Head of Department’s commitment to fostering a collaborative and supportive atmosphere has greatly enhanced my learning experience. Her constructive feedback and encouragement have been invaluable in overcoming challenges and achieving our project goals.

We are immensely thankful to our project supervisor, Dr. Islabudeen M., for his dedicated mentorship and invaluable feedback. His patience, knowledge, and encouragement have been pivotal in the successful completion of this project. Our supervisor’s willingness to share his expertise and provide thoughtful guidance has been instrumental in refining our ideas and methodologies. His support has not only contributed to the success of this project but has also enriched our overall academic experience.

Thank you all for your contributions and support.

**Name of the Candidate**

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Sl.No** | **Contents** | **Page No.** |
|  | **Abstract** | **xii** |
| **1.** | **INTRODUCTION** | **1** |
|  | 1.1 Background | **1** |
|  | 1.2 Motivations | **1** |
|  | 1.3 Scope of the Project | **2** |
| **2.** | **PROJECT DESCRIPTION AND GOALS** | **3** |
|  | 2.1 Literature Review | **3** |
|  | 2.2 Research Gap | **31** |
|  | 2.3 Objectives | **33** |
|  | 2.4 Problem Statement | **34** |
|  | 2.5 Project Plan | **34** |
| **3.** | **TECHNICAL SPECIFICATION** | **37** |
|  | 3.1 Requirements | **37** |
|  | 3.1.1 Functional | **37** |
|  | 3.1.2 Non-Functional | **37** |
|  | 3.2 Feasibility Study | **38** |
|  | 3.2.1 Technical Feasibility | **38** |
|  | 3.2.2 Economic Feasibility | **39** |
|  | 3.2.3 Social Feasibility | **39** |
|  | 3.3 System Specification | **39** |
|  | 3.3.1 Hardware Specification | **40** |
|  | 3.3.2 Software Specification | **40** |
| **4.** | **DESIGN APPROACH AND DETAILS** | **41** |
|  | 4.1 System Architecture | **41** |
|  | 4.2 Design | **46** |
|  | 4.2.1 Data Flow Diagram | **46** |
|  | 4.2.2 Use Case Diagram | **47** |
| **5.** | **METHODOLOGY AND TESTING** | **48** |
|  | 5.1 Methodology | **48** |
|  | 5.1.1 Overview | **48** |
|  | 5.1.2 Detailed Methodology | **48** |
|  | 5.2 Testing | **51** |
|  | 5.2.1 Testing Environment | **51** |
|  | 5.2.2 Testing Scenarios and Results | **52** |
| **6.** | **PROJECT DEMONSTRATION** | **53** |
|  | 6.1 Directory and File Navigation  6.1.1 Subdomain Directory  6.1.2 Social Engineering Directory  6.1.3 Emails Directory  6.1.4 Screenshots Directory  6.1.5 Vulnerabilities Directory | **53**  **54**  **57**  **58**  **58**  **58** |
| **7.** | **RESULT AND DISCUSSION** | **60** |
|  | 7.1 Results | **60** |
|  | 7.2 Discussion | **62** |
| **8.** | **CONCLUSION** | **65** |
| **9.** | **REFERENCES** | **67** |
|  | **APPENDIX A – SAMPLE CODE** | **73** |

### List of Figures

|  |  |  |
| --- | --- | --- |
| **Figure No.** | **Title** | **Page No.** |
| 1 | Gantt Chart | 35 |
| 2 | Architecture Diagram | 41 |
| 3 | Data Flow Diagram | 46 |
| 4 | Use Case diagram | 47 |

**List of Tables**

**Table No. Title Page No.**

1. Test Scenarios and Results 52

## List of Abbreviations

3D Three-dimensional

ACM Association for Computing Machinery

API Application Programming Interface

AXFR Asynchronous Transfer Full Range

BiLSTM Bidirectional Long Short-Term Memory

CLI Command Line Interface

CVE Common Vulnerabilities and Exposures

CVSS Common Vulnerability Scoring System

CWE Common Weakness Enumeration

DNS Domain Name System

DDQN Double Deep Q Network

DQN Deep Q Network

FTP File Transfer Protocol

GB Gigabyte

HD High Definition

HR Human Resource

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

ICM Integrated Controls Management

IDF Intermediate Distribution Frame

IDS Intrusion Detection System

IP Internet Protocol

ISAAF Information Systems Security Assessment Framework

IT Information Technology

JSON JavaScript Object Notation

LDAP Lightweight Directory Access Protocol

LSTM Long Short-Term Memory

MDP Markov Decision Process

ML Machine Learning

MulVAL Multi-stage Vulnerability Analysis Language

NetBIOS Network Basic Input/Output System

NIST SP National Institute of Standards and Technology Special Publication

NSE Nmap Scripting Engine

OS Operating System

OSINT Open-Source Intelligence

PHP Hypertext Preprocessor

PT Penetration Testing

PTES Penetration Testing Execution Standard

RAM Random-Access Memory

RL Reinforcement Learning

ROI Return On Investment

SMB Server Message Block

SMS Short Message Service

SNMP Simple Network Management Protocol

SQL Structured Query Language

SSD Solid-State Drive

SSL Secure Sockets Layer

SSRF Server-Side Request Forgery

TLD Top-Level Domain

TLS Transport Layer Security

URL Uniform Resource Locator

USB Universal Serial Bus

WAVS Web application vulnerability Scanners

WAF Web Application Firewall

XSS Cross-Site Scripting

**Symbols and Notations**

γ Discount Factor

O Big O

? Uncertain Value

+ Addition Operator

x Multiplication Operator

**ABSTRACT**

Penetration testing is integral to cybersecurity and involves computer systems, networks, or applications, through simulated cyber-attacks, to reveal their security vulnerabilities before malicious actors have a chance to take advantage of the identified vulnerabilities. It complicates the process when social engineering comes into play since that form of an attack method influences people to disclose confidential information, thereby complicating the Reconnaissance phase. Since cyber threats are in a continuous state of evolution, adversaries use in-depth and prolonged reconnaissance activities in order to intrude into a network system and compromise the systems therein. Thus, the Recon phase is becoming much more significant nowadays. A Recon is the initial phase of a penetration test wherein extensive information of a target system or network is gathered. It involves some inefficient practices that may lead to compromising the efficiency of security testing.

These inefficiencies are particularly disturbing in the context of the latest types of attacks, such as ransomware, that may result in great financial loss and loss of organizational prestige. In this paper, we discuss an advanced Recon framework that automates and optimizes the reconnaissance phase in penetration testing. This advanced tool is developed using Bash and Python and systematically collects, organizes, and analyzes data from third-party, human, and system sources.

What makes this framework innovative is the use of social engineering tactics, which, apart from technical weaknesses, identify human weaknesses that could predispose someone towards committing security risks. As compared to performing traditional reconnaissance, the framework enhances both the speed and quality of information discovered, thereby further improving the ability of an organization in defending against emerging cyber threats.

1. **INTRODUCTION**
   1. **BACKGROUND**

Penetration testing has now become a significant part of cybersecurity because it allows the organization to foresee and correct possible vulnerabilities in their systems beforehand. Traditional penetration testing has some inefficiencies during the Recon (reconnaissance) phase through which an attacker collects all the critical information about the target system, network, or application. Such inefficiencies lead to incomplete assessments and make an organization vulnerable to more advanced attacks, such as ransomware, social engineering, and other tactics that are dynamically developing.

Modern recon in cyber-attacks is no longer limited to technical probing but also targets human weaknesses through social engineering, making this phase more complicated and vital than ever. Importantly, the Recon phase is often time-consuming and still lacks automation to keep up with the rapid evolution of cyber threats.

The project answers the mentioned challenges through introducing an advanced Recon framework to automate and optimize the reconnaissance phase of the penetration test. The tool was built with Bash and Python, helping in the smooth collection and analysis of data and thus decreasing the speed and accuracy issues. Our framework includes social engineering techniques, providing a comprehensive view of potential security risks to organizations, thereby helping them strengthen their defenses against the dynamic landscape of cyber threats.

* 1. **MOTIVATION**

The increasingly complex nature of cyber threats, particularly during the reconnaissance phase in penetration testing, necessitates much advanced and efficient tools. Traditional reconnaissance methods are usually characterized by manual collection and analysis of information, time-consuming in the truest sense of the phrase, as well as vulnerable to human error. Inefficiency by this approach often means partial assessments, thereby allowing organizations to stay wide open to those targeted attacks like ransomware, phishing, and advanced social engineering tactics that exploit both technical as well as human weaknesses.

This project has been motivated by the fact that reconnaissance based penetrating tests need to be made more efficient and effective. With attackers going about their business with highly strategized wide reconnaissance attacks, methods to efficiently source and analyze information at much faster speeds with higher accuracy are in acute demand. Our highly developed Recon framework is founded using Bash and Python, in order to fill the potential gap by using data collection and social engineering techniques both focused towards the discovery of technical and human vulnerabilities.

In streamlining the reconnaissance process, this project hopes to give cybersecurity professionals a clearer sight toward enhanced threats, thereby allowing better defense capabilities from changing cyberattacks, therefore adding up to more resilient cybersecurity practices in the end.

* 1. **SCOPE OF THE PROJECT**

This project scope is concerned with the development of an advanced Recon framework to better the reconnaissance phase of penetration testing. Traditional reconnaissance has normally been very painful to do, and it is pretty error-prone. This traditional way of doing reconnaissance is too incapable for modern threats that exploit technical and human vulnerabilities. Our framework will automate collection, organization, and analysis of data from varied sources that include social engineering tactics.

The demo will focus on problems that usually crop up during reconnaissance-in, such as poor data collection, which may not represent vulnerabilities in its distinct complexities. Our framework automates these functions, thereby enhancing the speed and accuracy of information collection to eventually deliver a more defined understanding of security risks.

In this regard, performance both in simulation as well as real mode of operation would be tested so that such ease with which the tool locates vulnerabilities and makes reconnaissance so much easier should be analyzed. The project would further consider such more practical questions on limiting factors of automating reconnaissance and the need to include a human check.

Thus, it is designed to equip cybersecurity professionals with a more effective and dependable penetration testing tool that has been imbued with enhanced reconnaissance, hence better preparation against these changing cyber threats.

**2. PROJECT DESCRIPTION AND GOALS**

**2.1 LITERATURE REVIEW**

**2.1.1 S Roy et al, “Survey and taxonomy of adversarial reconnaissance techniques”, ACM Computing Surveys, 2022**

This paper outlines the techniques of adversarial reconnaissance, which are extremely important in attacks on cyber platforms. The methodology categorizes these techniques based on sources that attackers obtain information from, namely third-party sources, humans, and systems. In addition, the paper discusses the phases at which such techniques are applied, ranging from external application before access into a network and internal application after gaining such access.

**Methodology**

**Reconnaissance Target Information**:

Adversaries gather both non-technical (social) and technical intelligence about the target. Non-technical intelligence covers the organization and personal information, while technical covers network, host, application, and user level.

**Reconnaissance Phases**:

Reconnaissance is further divided into two stages: external before accessing the target's network and internal after gaining access to the network. Both are necessary in formulating attack plans.

**Taxonomy of Reconnaissance Techniques**:

Techniques are categorized based on their sources:

* **Third-party-based techniques**: These focus on passive information gathering from public sources (e.g., websites, social media, DNS).
* **Human-based techniques**: Social engineering techniques, such as phishing and pretexting, are used to manipulate individuals to gain sensitive information.
* **System-based techniques**: These involve active information gathering, such as network scanning and sniffing, to identify vulnerabilities within the target’s systems.

**Architecture**

The adversarial reconnaissance architecture is based on the phases of a cyber kill chain. Information in various steps during the attack can be taken to facilitate the design and execution of attacks. It consists of:

**External Reconnaissance**:

Involves passive information gathering using third-party sources (e.g., **DNS interrogation**, **WHOIS lookups**, **website footprinting**, and **social media tracking**).

**Internal Reconnaissance**:

Occurs after gaining access to the network. It includes techniques such as **network scanning**, **sniffing**, and **system discovery** to collect deeper information about the target network and its configurations.

**Tools and Techniques**:

Various publicly available tools such as **NetCraft**, **NMap**, and **WHOIS lookup** are used for external reconnaissance, while internal techniques involve more active scanning and monitoring through methods like **network fingerprinting** and **packet sniffing**.

**Advantages and Disadvantages of Each Methodology**

**Third-party-based Techniques**

**Advantages**:

These techniques are often **passive**, reducing the risk of detection. Publicly available information can provide valuable insights without interacting directly with the target system.

**Disadvantages**:

The information collected may be **outdated or incomplete**, limiting its effectiveness in real-time attack scenarios.

**Human-based Techniques**

**Advantages**:

**Social engineering** techniques such as phishing can be highly effective in bypassing technical defenses by exploiting human vulnerabilities.

**Disadvantages**:

These methods require **direct interaction with individuals**, which increases the risk of exposure and detection. Success is also dependent on the target’s susceptibility to manipulation.

**System-based Techniques**

**Advantages**:

These techniques provide **direct access to technical details** about the network and host systems, allowing for precise targeting of vulnerabilities.

**Disadvantages**:

**Active techniques** like scanning can be detected by security systems, increasing the likelihood of being blocked or flagged by intrusion detection systems.

**Conclusion**

Adversarial reconnaissance is a critical part of the cyberattack and offers an attacker the possibility of obtaining all the required information to construct and carry out sophisticated attacks. Taxonomy and categorization techniques in third-party, human-based, and system-based approaches provide a structured method for understanding the reconnaissance process. Therefore, defensive strategies such as cyber deception and advanced monitoring tools are requisite to counter these evolving reconnaissance techniques. It, therefore, calls for thorough study on reconnaissance to ultimately enhance security defense and evade success of a cyberattack.

**2.1.2 Fouz Barman et al, “A Methodical Framework for Conducting Reconnaissance and Enumeration in the Ethical Hacking Lifecycle”, ECCWS 2023 22nd European Conference on Cyber Warfare and Security, 2023**

The journal describes a **methodical framework** for reconnaissance and enumeration in the ethical hacking lifecycle. The methodology focuses on the use of **Nmap** and **Netcat**, two popular tools for information gathering and enumeration during penetration testing.

**Methodology**

**Reconnaissance Phase**: Involves passive and active information gathering, utilizing **Nmap** to scan for open ports, services, and vulnerabilities. The methodology emphasizes thorough documentation, including the steps followed and the commands used, such as scanning IP addresses, detecting OS versions, and identifying open ports.

**Enumeration Phase**: The next step after reconnaissance where active connections to the target system are established using **Netcat**. The methodology describes how attackers gather additional information through enumeration techniques like NetBIOS, SNMP, and LDAP enumeration to identify and exploit vulnerabilities.

The methodology includes a clear setup of the **Kali Linux platform** for penetration testing and provides commands and scripts for both tools to guide the process of ethical hacking.

**Architecture**

The framework is divided into two phases:

**Reconnaissance Phase**:

This phase emphasizes gathering passive and active information, which includes scanning open ports, determining active hosts, and detecting operating systems and services.

**Nmap** is the main tool used, and the process includes different types of scans (host discovery, port scanning, service version detection, etc.). It also involves detailed documentation of the attack surface.

**Enumeration Phase**:

This phase focuses on creating active connections with the target system to gather deeper insights into network resources, services, usernames, and other information.

**Netcat** is used for banner grabbing, connectivity testing, and additional port scanning.

It also includes specific enumeration techniques such as SNMP and NetBIOS, which further exploit the target’s system.

**Advantages and Disadvantages of Each Method:**

**Advantages of Reconnaissance Phase:**

* **Comprehensive Information Gathering**: By using **Nmap**, a wide range of information, such as open ports, running services, and OS versions, can be gathered.
* **Versatile and Detailed**: The tool allows both broad scanning (full port range) and focused analysis (specific IP ranges or services).
* **Support for Multiple Platforms**: Nmap can be used on Linux, Windows, macOS, and other platforms, making it flexible.
* **Customization**: The methodology includes scripting with **NSE (Nmap Scripting Engine)** for vulnerability detection and exploitation.

**Disadvantages of Reconnaissance Phase:**

* **Noise in the Network**: Active scanning with **Nmap** can trigger alarms in target systems, potentially leading to detection.
* **Complexity for Beginners**: While powerful, the tool can be complex for inexperienced users, with many options and commands that might overwhelm them.
* **Limited in Detecting New Exploits**: Nmap is primarily a network mapper and may not detect all new or advanced vulnerabilities without additional tools.

**Advantages of Enumeration Phase:**

* **Deep Information Access**: Using **Netcat**, the tester can actively query services and gather more specific details, such as usernames, shares, and service versions.
* **Lightweight and Flexible**: Netcat is a simple yet powerful tool for establishing connections and performing basic network functions.
* **Wide Range of Uses**: It can act as a banner grabbing tool, port scanner, or even establish tunnels for encrypted data transmission.

**Disadvantages of Enumeration Phase:**

* **Manual Effort**: Although Netcat is powerful, many of its capabilities require manual effort, limiting its scalability.
* **Limited to Known Protocols**: While it supports a variety of protocols, it might not be as effective for more advanced enumeration without additional tools.
* **Potential for Detection**: Since the phase involves active probing, the activities are more likely to be detected by the target system’s security controls.

**Conclusion**

This paper would help streamline both reconnaissance and enumeration phases in ethical hacking with a systematic methodology built on top of Nmap and Netcat. The same study makes a point regarding the necessity of diligent documentation and management at early stages of penetration tests should attention to these phases be paid during the performance. Value of this framework is given by its practicality and completeness. It is an in-depth, step-by-step guide for the ethical hacker/penetration tester, hence ensuring that the process is structured and repeatable. Still, the study underlines the fact that the tools will be complemented by other vulnerability detection and exploitation techniques because these tools are scope-limited and would not identify all forms of vulnerabilities.

**2.1.3 K Abdulghaffar et al, “Enhancing Web Application Security through Automated Penetration Testing with Multiple Vulnerability Scanners”, Computers, 2023**

This study proposes a novel framework aimed at improving web application security by automating the operation of multiple Web Application Vulnerability Scanners (WAVS). Two open-source vulnerability scanners, **Arachni** and **OWASP ZAP**, are integrated into a single platform that consolidates the results using custom-built algorithms.

**Methodology**

The methodology discussed in the study consists of the following components:

**Automation of Vulnerability Scanners**: The study begins by automating the operation of two widely-used vulnerability scanners—Arachni and OWASP ZAP. The goal of automating these tools is to reduce manual effort and eliminate potential human errors, which are common in manual penetration testing processes.

**Combination of Scanning Results**: Once the scanners have completed their respective scans, the results are processed by two algorithms:

**Automation Algorithm**: This algorithm is responsible for executing the scans and managing the results from both tools. It loops through each scanner, initiates the scan, stores the results, and processes them once the scans are completed. The automation aspect significantly streamlines the testing process, enabling quicker and more efficient vulnerability assessments.

**Combination Algorithm**: This algorithm consolidates the vulnerabilities detected by each scanner into a combined report. It generates two types of lists:

* **Union List**: This list includes all vulnerabilities identified by either Arachni or OWASP ZAP, essentially combining the results from both scanners. The union list is used to maximize coverage by ensuring that all potential vulnerabilities are captured.
* **Intersection List**: This list only includes vulnerabilities that have been identified by both scanners. Since both scanners flag the same issues, the intersection list represents vulnerabilities with a higher degree of confidence, reducing the likelihood of false positives.

**Evaluation and Metrics**: The performance of the proposed framework is evaluated based on key metrics: **precision**, **recall**, and the **F-measure**. Precision measures the ratio of true positives to false positives, recall measures the proportion of actual vulnerabilities detected, and the F-measure provides a balanced score of precision and recall. These metrics help determine the effectiveness of combining the results from multiple scanners compared to using a single scanner.

**Architecture**

The architecture of the framework is designed to support automated vulnerability scanning with a seamless user experience. It comprises three core components:

**Front-end**: The front-end is built using **React.js** and serves as the interface through which users interact with the framework. Users can input the target web application, initiate scans, and view the consolidated results generated by the scanners. The front-end communicates with the back-end using HTTP requests.

**Back-end**: The back-end, developed using **Node.js** and **Express**, is responsible for processing scan requests from the front-end. It handles communication between the vulnerability scanners and the front-end. The back-end initiates the scans using Arachni and OWASP ZAP, collects the results, and processes them through the combination algorithm.

**Vulnerability Scanners (Arachni and OWASP ZAP)**:

* **Arachni**: Arachni is an open-source vulnerability scanner capable of detecting web application vulnerabilities such as SQL injection, XSS, and more. It is chosen for its ability to navigate complex application architectures and its compatibility with modern technologies.
* **OWASP ZAP**: ZAP, developed by the OWASP foundation, is also a widely-used open-source tool for detecting web application vulnerabilities. It is effective in identifying vulnerabilities such as SQL injection, XSS, and other OWASP Top 10 issues. ZAP’s ability to run in an automated mode makes it an ideal candidate for integration into the framework.

Both scanners are executed from the **command-line interface (CLI)** to allow for automation, ensuring that the framework can handle large-scale scans efficiently. The results from these scanners are then fed into the combination algorithm, which processes and filters the vulnerabilities.

**Advantages and Disadvantages of Each Methodology**

**Combination Algorithm:**

**Advantage**: The primary advantage of the combination algorithm lies in its ability to enhance vulnerability detection by merging the results from multiple scanners. By generating a union list, the framework ensures that all vulnerabilities identified by either scanner are reported. This improves overall coverage and reduces the likelihood of missing critical vulnerabilities. Moreover, the intersection list allows the framework to focus on vulnerabilities detected by both scanners, providing a higher level of confidence in the results. This approach reduces false positives, making the vulnerability report more reliable.

**Disadvantage**: A major disadvantage of this methodology is the increased risk of false positives in the union list, as it includes all vulnerabilities reported by both scanners. Some vulnerabilities might be false alarms that are flagged by one scanner but not the other. Additionally, managing the volume of results from multiple scanners can complicate the analysis process, especially when the list of detected vulnerabilities is long.

**Automation Algorithm**

**Advantage**: The automation algorithm simplifies the process of running vulnerability scans by removing the need for manual intervention. This saves time and effort, especially when dealing with large web applications or complex systems that require frequent scanning. By automating repetitive tasks, this methodology also minimizes human error, leading to more consistent results.

**Disadvantage**: The automation algorithm depends heavily on the correct functioning of the scanners. If one scanner fails to operate or produces incomplete results, the framework’s output might be affected. Additionally, the algorithm’s complexity can pose challenges when expanding the framework to include additional scanners or more advanced scanning techniques.

**Conclusion**

It can be seen from the study that results obtained using several scanners with an automated framework significantly increases the accuracy and the coverage of the vulnerabilities detected in web applications. The Union List, which implements the combination of results from both Arachni and OWASP ZAP, yields the highest F-measure, thereby proving a good balance between precision and recall. This effectiveness to make use of multiple scanners simply enhances detection rates while reducing the likelihood of a missed vulnerability. The intersection list, meaning the vulnerabilities identified by the scanners, was very specific but a low recall rate, which suggests missing vulnerabilities.

**2.1.4 Vijaya R Saraswathi et al, “Automation of Recon Process for Ethical Hackers”, IEEE, 2022**

The journal presents a framework designed to automate the reconnaissance (recon) phase of penetration testing for ethical hackers. The methodology involves automating several key tasks that are part of the information-gathering process, which is crucial in identifying potential vulnerabilities in a target system.

**Methodology**

**Recon Process Automation**:

The framework automates tasks such as **subdomain enumeration**, **port scanning**, **Google dorking**, **JavaScript enumeration**, and **fuzzing for hidden content**.

It takes the top-level domain (TLD) of the target organization as input and performs a comprehensive information-gathering process. The results are stored in an organized manner, making them easy to review and utilize for further penetration testing phases.

**Tools Used**:

The tools integrated into the framework include those written in **Go** and **Python**, which support multi-threading to enhance the speed and efficiency of the recon process.

**Nuclei** is used for detecting low-hanging vulnerabilities, and additional tools are employed to perform tasks like subdomain enumeration and port scanning.

**Modular Structure**:

The framework is divided into modules, where each module performs a specific task in the recon process. The output of one module is sometimes used as input for another, allowing for seamless integration between tasks such as **subdomain discovery** and **JavaScript file analysis**.

**Execution Flow**:

The recon script begins by creating directories to store the results.

Subdomain enumeration identifies all subdomains associated with the target.

The live subdomains are extracted, followed by tasks like **Google dorking**, **credential stuffing**, and **JavaScript enumeration**.

**Port scanning** is then performed to identify open ports and the services running on them. Finally, **fuzzing** and **Nuclei scans** are conducted to discover hidden content and vulnerabilities like XSS or SSRF.

**Architecture**

The architecture of the proposed model follows a structured workflow:

**Input**: The user provides the top-level domain (TLD) of the target organization through a command-line interface.

**Modules**: Each module in the framework performs a specific task:

* **Subdomain Enumeration**: Identifies subdomains and stores them in a subdomain directory.
* **Google Dorking**: Searches for hidden information using Google dorks.
* **JavaScript Enumeration**: Analyzes JavaScript files to identify potential security issues like hardcoded credentials.
* **Port Scanning**: Detects open ports and services running on the target’s subdomains.
* **Nuclei Scanning**: Scans for vulnerabilities using the Nuclei tool.

**Results**: The results of each module are stored in separate directories, making it easy for the user to navigate and view the findings. For example, the subdomains are stored in the "subdomains" directory, and the output of the Nuclei scan is stored in its own directory.

**Advantages and Disadvantages of Each Methodology**

**Automated Recon Process**

**Advantages**:

* **Speed and Efficiency**: Automation significantly reduces the time required for recon tasks compared to manual execution.
* **Consistency**: Automated scripts ensure that the recon process is performed systematically, minimizing human error.
* **Scalability**: The framework can handle large-scale recon tasks, making it suitable for enterprises with extensive networks.

**Disadvantages**:

* **Customization Limitations**: Automated tools may miss context-specific vulnerabilities that could be identified through manual recon.
* **False Positives**: Automated scans might flag issues that are not actual vulnerabilities, requiring additional validation.

**Modular Structure:**

**Advantages**:

* **Flexibility**: The modular approach allows users to run specific tasks independently, such as focusing on subdomain enumeration or port scanning without running the entire recon process.
* **Integration**: The seamless transition of outputs between modules ensures efficient execution of the entire recon process.

**Disadvantages**:

* **Dependency on Modules**: If one module fails, it can affect the subsequent modules that rely on its output, requiring robust error handling.

**Conclusion**

The automation of the reconnaissance process for ethical hacking offers significant benefits by streamlining the information-gathering phase of penetration testing. The modular framework proposed in the study automates tasks like subdomain enumeration, port scanning, and vulnerability detection using tools like Nuclei, resulting in faster and more efficient recon processes.

The framework enhances the penetration tester's ability to gather information quickly, reduces the need for manual work, and allows testers to focus on other phases of penetration testing, such as exploitation. While automated recon simplifies the process, manual validation may still be required to ensure accuracy and eliminate false positives. Overall, this framework offers an efficient solution to recon automation, making it a valuable asset for ethical hackers.

**2.1.5 EA Altulaihan et al, “A Survey on Web Application Penetration Testing”, Electronics, 2023**

In the journal, "A Survey on Web Application Penetration Testing," methodologies and tools applied during penetration testing of web application vulnerabilities are discussed.

**Methodologies and Architectures**

This journal revealed two methodologies that can be used for penetration testing: manual testing and automated testing.

**Manual Penetration Testing**

Security experts use this skill set to demonstrate the real-word attacks by exploiting vulnerabilities through manual testing. That is often customized and is very exhaustive with great cost in terms of time, money, and expertise involved. According to the article, Manual testing is essential in taping the complex vulnerabilities that may not have been captured through the automated tools.

* **Advantages:** Exact results when it involves complex vulnerabilities, permits customization of attack strategy, simulates real-word strategies.
* **Disadvantages:** Expensive, Time-Consuming and uses a professional expert. It also lacks scalability and is a laborious manual effort.

**Automated Penetration Testing**

Automated tools such as OWASP ZAP, Acunetix, and Netsparker can be used to scan vulnerability with the help of penetration test without wasting valuable time by using them in a scalable way. All of these automate issues from finding SQL injection XSS, cross site scripting and many more.

* **Advantages:** It is faster and cheaper and can be embedded in the development lifecycle; thus, it allows for continuous testing and scanning.
* **Disadvantages:** It can only detect relatively simple vulnerabilities; it can lead to false positives or negatives regarding detection; also, it requires human experts to interpret its findings.

**Hybrid Approach (Combination of Automated and Manual Testing)**

It is thought that the ideal is a hybrid of early scanning by automated testing tools and later manual testing by experts. Such an approach would leverage the speed of automated tools while deeper vulnerabilities are assumed to be treated by human expertise.

* **Advantages:** Combines the strengths of both methods, enhances coverage, and balances accuracy with efficiency.
* **Disadvantages**: Requires coordination between automated and manual efforts, and also still resource-intensive

**Conclusion**  
This paper argues that penetration testing tools and methodologies have specific strengths and weaknesses. Automated testing is good for frequent scans, whilst manual testing gives deep analysis for more complex vulnerabilities. Hybrid maximizes the advantages of both, with inherent challenges in coordination and resource management.

Further research efforts shall be directed to developing more advanced tools that amalgamate the productivity of automation with the depth of manual testing to improve the detection of technical and business logic web application vulnerabilities.

**2.1.6 J Yi et al, “Deep Reinforcement Learning for Intelligent Penetration Testing Path Design”, Applied Sciences, 2023**

The journal "Deep Reinforcement Learning for Intelligent Penetration Testing Path Design" mainly applies reinforcement learning in the optimizing attack path planning design of penetration testing in designing intelligent penetration testing.

**Methodologies and Architectures**

**Markov Decision Process (MDP):**

Formulation of penetration testing network model as an MDP which reflects penetration testing as the transition process from one state to another. An MDP is described as follows: <S, A, P, R, γ>, where S denotes system states, A denoted possible actions, P denotes transition probabilities, R - the reward function, and γ denotes the discount factor.

* **Advantages:** The MDP provides an integrated framework in representing the penetration testing by mapping states and actions so that the reinforcement learning might optimize the paths of attack.
* **Disadvantages:** The computational complexity of MDP is substantial, which adversely impacts scalability for real-world environments especially large-scale ones.

**Deep Q Network (DQN):**

DQN is an RL algorithm combining Q-learning with deep learning in the solution of the MDP. Here, the algorithm will use a neural network to approximate the Q-value function that will guide the algorithm on picking the best among the available actions at each state.

* **Advantages:** The difference between DQN and the traditional Q-learning is that the former can learn the value function much better using deep neural networks in such a way that it is possible to adapt to complex, high-dimension state spaces.
* **Disadvantages:** Q-value is overestimated by DQN, which can lead to poorly updated policies and deteriorated performance.

**Double Deep Q Network (DDQN):**

DDQN is suggested as an extension of DQN to avoid overestimation problems. In this method, two separate networks are used while keeping them apart to choose the action and evaluate it to decrease overestimation possibility in the Q-value.

* **Advantages:** DDQN stabilizes training, evades overestimation, and enhances the designing of paths in penetration testing.
* **Disadvantages:** It still faces sparse reward problems, where the absence of positive reinforcement continues

**MulVAL and Attack Graphs:**

This approach integrates DDQN with the MulVAL (Multi-stage Vulnerability Analysis Language) attack graph tool. MulVAL generates an attack graph based on network topology along with the vulnerability information that drives DDQN to explore the attack paths.

* **Advantages:** It utilizes the priors of attack graphs in training and converges fast because promising insights into reachable states make the training efficient.
* **Disadvantages:** MulVAL works well in network environments; however, state explosion starts to dominate when the sizes become large.

**MDDQN Algorithm:**

The paper presents the MDDQN (MulVAL and DDQN) algorithm. This algorithm merges the attack graph of MulVAL with DDQN. MDDQN builds a transfer matrix based on reachable paths from the attack graph itself. In this, MDDQN optimizes the planning of the attack path using deep reinforcement learning.

* **Advantages:** The algorithm significantly increases the convergence speed, solves sparse reward problems, and makes attacks much more efficient in terms of planning an attack path.
* **Disadvantages:** It relies on pre-existing knowledge of the system and cannot itself scan for the various configurations of a network, which makes it less adaptable in a dynamic environment.

**Conclusion**  
The paper introduces MDDQN as an efficient algorithm for path design in the context of intelligent penetration testing that combines MulVAL and DDQN. It brings about many obvious advantages in terms of convergence speed and attack path optimization but suffers from issues regarding scalability and adaptability. Improvement of scalability and adaptation to non-preconfigured scenarios in dynamic environments is recommended for future work (applsci-13-09467).

**2.1.7 M Alhamed et al, “A Systematic Literature Review on Penetration Testing in Networks: Future Research Directions”, Applied Sciences, 2023**

Journal "A Systematic Literature Review on Penetration Testing in Networks: Future Research Directions" reviews the methods applied in network penetration testing. This concerns its ability to detect vulnerabilities in different network topologies. The paper presents a variety of penetration testing approaches, tools, and methodologies widely used in network security assessment.

**Methodologies and Architectures**

**Penetration Testing Approaches**

**Black Box Testing**

In this technique, testers have no idea about the network. They simulate an attack from outside; various tools are used for the exploration of vulnerabilities through trial and error.

* **Advantage:** Simulates a real-world attack scenario that may expose vulnerabilities exploited by an attacker outside of the organization.
* **Disadvantages:** Long process and not so efficient to find vulnerabilities which require knowledge of the inside of the system.

**White Box Testing**

Testers are provided with complete information about the network information, which involves architecture, source code, and internal configurations. This method is much more elaborate and focused.

* **Pros:** It gives a deeper understanding of the system flaws and it's faster because one has data related to the internal configuration.
* **Cons:** High cost and resource-intensive

**Gray Box Testing**

It is a combination of black and white-box testing since the tester only gets partial information about the system.

* **Advantages:** It balances cost with efficiency when uncovering both internal and external threats.
* **Disadvantages:** It may still miss certain vulnerabilities that require full internal access or attacks from the outside.

**Penetration Testing Tools:**

The journal discusses the most popular tools used in penetration testing, which are basically Nmap for scanning the network, Metasploit for exploitation of vulnerabilities, Wireshark for packet analysis, and BeEF for browser-based exploitation.

* **Advantages:** Because most of the penetration testing tasks are automated, it greatly boosts efficiency and accuracy in detecting vulnerabilities.
* **Disadvantages:** Tools like Nmap are liable to false positives or fail to detect some vulnerabilities when installed in interactive mode. Apart from that, they require professional expertise to interpret the results.

**Penetration Testing Standards:**

Paper stated numerous standards that should be followed in penetration testing, including,

NIST SP 800-115 states explicitness in the framework concerning network penetration tests which emphasize planning, detection, attacking, and reporting.

Open Source Security Testing Methodology Manual states a comprehensive guide to ensure the security of the network through best practices of penetration testing.

Penetration Testing Execution Standard (PTES): A well-structured process for information gathering, threat analysis, and exploit.

Information Systems Security Assessment Framework (ISAAF): A framework for application and network controls evaluation.

* **Advantages:** They offer consistency and ensure that a comprehensive assessment is done on network security.
* **Disadvantages:** They may be too cumbersome for small organizations, considering their limited resources.

**Conclusion**

This systematic literature review elaborates about the pros and cons of various methods and tools in penetration testing that a hybrid approach, namely both automated and manual methods, provides the best outcomes. In this paper, it has highlighted the need for more complex tools that benefit from automation and machine learning capabilities by driving efficiency and scaling factors of penetration tests (applsci-13-06986).

**2.1.8 AD Tudosi et al, “Research on Security Weakness Using Penetration Testing in a Distributed Firewall”, Sensors, 2023**

The journal, "Research on Security Weakness Using Penetration Testing in a Distributed Firewall" primarily focuses on identifying and addressing vulnerabilities in a distributed firewall system through penetration testing.

**Methodologies or Architecture Applied**

The study emphasis on penetration testing methodologies aimed at finding penetration or security weaknesses in a distributed firewall environment. The methodologies used include manual and automated:

**Manual Penetration Testing Using Tools**

**Nmap:** It is an effective network scanning tool that helps one determine what is open on port-wise and services where Nmap was applied in the study in order to manually identify network vulnerabilities, services, and firewall filtering capabilities.

**Netcat banner grabbing**: This is a technique used to detect if an open port is running on the system. Fire walking refers to how one analyzes the security configuration of the firewall in order to determine whether or not filtering is applied on IP forwarding protocol. Penetration testing - Automatic use of Nessus application.

**Nessus:** This tool scans the entire system and has all information related to the possible weaknesses by itself. For the experimentation, Nessus used automatically to scan architecture to prove it showed all of the vulnerability points according to the Common Vulnerability Scoring System (CVSS).

**pfSense:** The distributed firewall used for the paper is derived from pfSense that is an open-source firewall system. The system was tested by numerous penetration test techniques for testing the security.

**Distributed Firewall Architecture**

A system with variously configured firewalls distributed in a network is demonstrated in the experiment in order to explain the architecture of a firewall. Such an architecture offers flexibility and security. Each fire wall protects various segments of the network.

**Advantages of Techniques Utilized**

**Hand Penetration Testing**

* **Flexibility and Personalization:** Hand testing depends solely on the experience and instinct of the testers. Some such vulnerabilities cannot be identified by automated tools.
* **Deep Testing:** Manual parameter tweaking will help the testers accurately test the system for their required needs.

**Automated Penetration Testing**

* **Scalability and Performance:** Tools such as Nessus can easily perform massive scans in a short period of time, reducing much time it would take to scan in detail the vulnerabilities of a system.
* **Comprehensive Discovery:** Testing at a distributed level by automated testing can capture numerous types of vulnerabilities that are related to configuration faults, outdated software applications, and encryption issues.

**Distributed Firewall Structure**

* **Improved Security:** A distributed firewall architecture provides many levels of security, and therefore hacking into the network would be very challenging.
* **Scalability:** Companies can expand their network without losing security because each part of the network has its firewall.

**Weaknesses of the Methods Used**

**Penetration Testing Manual**

* **Labor-Intensive:** Penetration testing manually is a time-consuming activity and will consume so much time particularly for big networks.
* **Human Failure:** Some vulnerabilities are missed because someone simply failed to notice them or didn't know what to look for.

**Automated Penetration Testing**

* **False Positives/Negatives:** Automated tools will notify on items that are in fact not a problem (false positives) and may miss some of the most critical vulnerabilities (false negatives).
* **Not as Flexible as Manual Testing:** Automated tools are fast, but they can't match the flexibility possible in manual testing.

**Distributed Firewall Architecture**

* **Complexity:** Distributed firewalls are a bit cumbersome because there is a lot of configuration and monitoring that has to occur to ensure that all parts of the network are protected.
* **Resource Intensive:** This setup can be resource-heavy, requiring skilled personnel to maintain and monitor the network security infrastructure.

**Conclusion**

This research study presents an examination of penetration test methodologies on a distributed firewall system, offering insights into both manual and automated testing techniques. The firewall system can be audited in great detail with the use of Nmap, Netcat, Firewalking, and Nessus. There is also the advantage and disadvantage of each methodology: the intense depth that can be had through manual testing, and the speed and extent of the scanning through automated testing. Since the distributed architecture is so strong with its network protection, careful management and resources are needed to maintain effectiveness.

**2.1.9 S Zhou et al, “Autonomous Penetration Testing Based on Improved Deep Q-Network”, Applied Sciences, 2021**

The journal focuses on improving the efficiency and scalability of autonomous penetration testing (PT) using reinforcement learning, particularly through an enhanced Deep Q-Network (DQN) algorithm.

**Methodologies and Architecture Used**

**Markov Decision Process (MDP)**

Penetration testing is the description of a Markov Decision Process (MDP) where the agent, or a penetration tester, interacts with the environment or network to find the vulnerabilities. The actions choose according to the network's current state that will maximize the value of the reward function, which is the compromise of valuable hosts.

Network configuration and vulnerabilities form the state information, while actions include operations such as exploitation of vulnerabilities or elevation of privileges. The reward function would then represent this trade-off between sensitive hosts compromised and the cost of operations.

**Improved Deep Q-Network (NDSPI-DQN)**

Deep Q-Network is a reinforcement learning algorithm using a deep neural network to approximate the value functions of different actions in given states.

The proposed article introduces a modified DQN, namely NDSPI-DQN, that confronts two of the most significant challenges in large-scale networks. These are the sparse reward problem and the large action space problem.

**Sparse Reward Problem:** Agents that use reinforcement learning generally suffer from delayed rewards in which the agent does not receive sufficient feed-forward to learn. This problem grows harder with an increment in network size.

**Large Action Space Problem:** With the increase in network size, the number of actions the network can take exponentially increases with that size, which eventually makes it less efficient in exploration hence slowing its learning.

**Key Extensions to DQN in NDSPI-DQN**

The NDSPI-DQN combines five crucial extensions to the problems above:

* **Noisy Nets:** Introduces randomness into the weights of the network during training to encourage more exploration during the training process, beneficial for overcoming sparse rewards.
* **Dueling Network Architecture:** It separates the value function-value of being in a certain state-and the advantage function-how much better one action is compared to others-whose evaluation will make it easier for the network.
* **Soft Q-Learning. Introduce "soft" value function:** encourages more diversified exploration techniques; reduces the impact of action determinism.
* **Priority Experience Replay:** This permits prioritized experience with the highest temporal difference errors so ensuring important transitions are learnt by the agent more efficiently.
* **Intrinsic Curiosity Module:** The module self-generates an intrinsic reward that is proportional to how much the agent learns. It encourages an agent to explore new states, even if external rewards are sparse.

**Decoupled Action Space**

To further reduce the large action space, the algorithm decouples the attack vector into selecting a host and picking an operation. This brings down the dimensionality of the action space from O(M×N) where M is the number of hosts and N is the number of operations, to O(M+N), where much improved efficiency in exploration is achieved.

**Advantages of the Methods Used**

**Markov Decision Process**

* **Efficient Modeling of Problems:** The MDP representations of the pentesting task closely mirror the real-world practice of pentesting.

**Improvements in NDSPI-DQN**

* **Aggressive Exploration Combined with Fast Convergence:** Noisy Nets and Prioritized Experience Replay allow for effective exploration, such that the agent can master good strategies for learning in large, sparse environments.
* **Handling Large Networks**: Decoupling of actions lowers the complexity of large network scenarios, and the agent will find its efficient attack paths without being belabored by the number of possible actions.
* **Improved Scalability:** The architecture of Dueling Network and Soft Q-Learning has made the algorithm scalable and adaptable to different network sizes.
* Intrinsic Curiosity Module
* **Encourages Exploration:** The ICM ensures that the agent is curious about its unvisited states, even with sparse rewards. In short, this guarantees to solve the problem of sparse rewards.

**Disadvantages of Applied Methods**

**Markov Decision Process**

* **Limited Deployment in Real Worlds:** Models derived from MDPs work just fine in simulations. Real-world network topologies are far more dynamic and harder to model precisely using MDPs, which might limit its applicability.

**NDSPI-DQN Extensions**

* **Computational Complexity:** Despite reducing the action space, the overall complexity of the network architecture and the number of parameters to fine-tune can increase computational costs.
* **Action Overhead:** The split-action approach would lead to inefficient decisions if the two parts, host selection and operation selection, are not well synchronized.

**Sparse Rewards:**

Still Not Scalable for Extra Large Networks: Even with ICM and prioritization, in ultra-large networks, the reward signal may still be sparser; thereby slowing down learning.

**Simulation-Limited Testing:**

**Not Evaluated on Real Networks**: The evaluation of the algorithm relies on NASim-a simulation framework. The real-world environments are more complex; for example, networks have dynamic traffic, and network architectures evolve; hence, the algorithm might not generalize as well without extensive testing.

**Conclusion**

The approach developed a novel approach in autonomous penetration testing using a reinforcement learning model called NDSPI-DQN, that has improved classical DQN with several enhancements. Despite the fact that the methods tackle some of the major challenges related to sparse rewards and large action spaces, the mentioned issues regarding high model complexity, as well as reliance on simulations, are concerning in terms of practicability or scalability. This work is definitely an important step in the direction toward more scalable and truly autonomous cybersecurity testing solutions.

**2.1.10 P Wang et al, “A Cybersecurity Knowledge Graph Completion Method for Penetration Testing”, Electronics, 2023**

This journal focuses on enhancing cybersecurity through knowledge graph completion methods, designed to support penetration testing by filling in gaps and correcting errors in cybersecurity data.

**Methodologies or Architecture Applied**

**Cybersecurity Knowledge Graph**

In the cyber space, the Knowledge Graph is a representation of the available information on cybersecurity, just like in CWE and CVE, penetration testing tools and Metasploit, and hence it has a semantic structure as it holds the relationships of entities within the cyber space. It implies that the penetration tester can map out a system and its vulnerabilities with this Knowledge Graph.

The knowledge graph consists of triples in the format of (head entity, relationship, tail entity) that represent facts. The paper designs knowledge graph completion by filling missing triples using some ready data such as (?, relationship, tail) or (head, relationship, ?).

CSNT Model-Cyber Security Knowledge Graph Completion by Neural Network and Tensor Decomposition

**BiLSTM:** The model captures interactions between entities and relationships of the graph. It elucidates complicated relationships in cyberspace, as information is exchanged in two opposite ways.

**Tensor Decomposition:** Modeling entity relationship, the interactive entity-relationship encoding is compressed into a 3D tensor, decomposed to enrich triple representation.

Neural Network Model: This is synergistic with tensor decomposition in approximating a complex, non-linear relationship in the graph in order to achieve higher accuracy in missing data imputation.

**Pearson Mix Net:** A new approach for merging feature learning from tensor decomposition as well as neural networks. The weight parameters controlled through the Pearson correlation coefficient blends in contributions from different modules in creating better entity as well as relation embeddings.

**Progressive-Replay-SA Self-Distillation**

In this approach, catastrophic forgetting during training is addressed by revisiting and replaying previous batches of data. This reduces the chances of the model forgetting previously acquired earlier knowledge it had gained while training.

**Self-Distillation Method:** Contrary to traditional distillation that involved a separate teacher network, self-distillation simply enables the model to learn from its previous generations by incrementing learning difficulty over time and using simulated annealing to change the training difficulty so the model learns concepts from simple to complex.

**Advantages of the Techniques Applied**

**Knowledge Graph Architecture**

* **Holistic Representation of Cyberspace:** Utilizing knowledge graphs helps to represent cybersecurity elements - like IP addresses, vulnerabilities, and services-in a more structured manner with richer information. Therefore, penetration testers will handle the data much more easily.
* **Error Detection and Correction:** The model of the knowledge graph completion is not only capable of filling missing data but also detects and corrects eroneous information, which makes the intelligence in cybersecurity more reliable.

**CSNT Model**

* **Improved Entity and Relationship Interaction:** BiLSTM enhances better interactions between the entities and relationships linked up, and semantic patterns worth more capture.
* **Efficient Feature Fusion:** Pearson Mix Net makes better feature fusion wherein the model is dynamically fitted along with the correlation of its components, thus enhancing the tensor decomposition performance and neural networks' outcome, and then provides further triple completion with enhanced precision.
* **Scalability and Accuracy:** Combining neural networks with tensor decomposition gives large amounts of data on cybersecurity to the model and thus, it's scalable and accurate toward real-world applications.

**Progressive-Replay-SA Self-Distillation**

* **Against Catastrophic Forgetting:** The strategy of replay prevents the model from forgetting previously learned information in the course of training.
* **Adaptive Learning:** Annealing process for distillation temperature adjustment has optimized learning efficiency, as the model starts from simple tasks and then progresses to complex ones with improvements in performance

**Drawbacks of the Approaches Adopted**

**Knowledge Graph Design**

* It relies on the quality of initial data. The ability of the model to complete and correct data is reliant on the quality of the initial knowledge graph. In case initial data is full of errors or holes, the model cannot complete that data item perfectly.

**CSNT Model**

* **High Computational Complexity:** The use of BiLSTM, tensor decomposition, and neural networks adds to the computational complexity of the model. Cybersecurity data at scale demands immense processing.
* **Dealing with Multiple Methods:** The combination of tensor decomposition and neural networks might introduce several challenges to keep both methods in balance. Without proper control, one method may take over the other or dilute the contributions of the other, which would lower the efficiency of the model.

**Progressive-Replay-SA Self-Distillation**

* **Slow Convergence:** As the learning problems at each step increase in gradient, slow convergence times may be possible. This might become a concern when environments demand very fast response times, such as with real-time penetration testing.
* **Complexity of Simulated Annealing**: Simulated annealing adds capabilities to adaptive learning. This therefore brings more complexity to a model's training process, since the fine-tuning of temperature and the management of the whole learning process might necessitate extra computing resources.

**Conclusion**

This paper presents a new paradigm for knowledge graph completion in the field of cybersecurity, which is intended to assist penetration testing. The proposed CSNT model will integrate neural networks with tensor decomposition efficiently to capture the relationships that are usually complex in the knowledge graph. It integrates Pearson Mix Net, along with the progressive replay self-distillation strategy, in order to enhance the robustness of the model in filling the gaps or correcting errors and to prevent catastrophic forgetting. Although the problem is significantly computationally intensive and relies heavily on high-quality initial data, the approach may provide an enriched solution to the problem of improving cybersecurity intelligence and penetration testing.

**2.2 RESEARCH GAP**

**Scalability Issues with Large Networks and Datasets**

There are several methods including Graph Convolution Networks, NDSPI-DQN as well as CSNT Model which cannot scale to large or ultra-large networks and datasets. Sparse rewards, large action spaces, and high computational complexity of these methods cause a drastic drop in performance on large networks and complex networked environments. Scalable models research gap exists where in huge scales, these models maintain good performance and efficiency.

**Computational Complexity and Resource Overheads**

The added complexity in the architectures is using the neural network structure-the DQN or LSTMs. These are computation heavy, making high-demand resources, quite unrealistic for implementation on real-time penetration testing where quick and lightweight decisions are required. This leaves a gap to design algorithms that are lightweight, low-latency, and resource-efficient algorithms which function in real-world applications in high-performance time-sensitive cybersecurity environments.

**Balancing Methods in Hybrid Architectures**

Hybrid architectures such as Tensor Decomposition with neural networks and fuzzy logic are challenging in balancing contributions from different methods since one method may overwhelm them when not thoroughly controlled, which diminishes the effectiveness of the hybrid solution. Thus, future work should aim at constructing an effective mechanism in balancing the contributions meaningfully so that not one of the components overpowers the rest.

**Inability to Handle Dynamic and Evolving Cybersecurity Threats**

Many techniques, for example, fuzzy logic, CSNT, are static and do not adapt well to changing cyber threats or new vulnerabilities that appear in real-time systems. Need more dynamic self-adaptive models that evolve abreast with the threat landscape without needing manual or continuous retraining.

**Vulnerability to Incomplete or Noisy Data**

Several approaches such as knowledge graph completion, graph-based vulnerability detection and others heavily depend on the quality of input data. Poor data quality may completely degrade performance and make such systems unreliable. A gap in the literature here is toward the design of noise-tolerant models that still deliver accurate, noise-free results, even when there could be noise or missing information.

**Lack of Integration with Social Engineering Techniques**

None of the approaches above integrates social engineering attacks in automatic penetration testing. Most of these again focus on technical vulnerabilities and leave the human factor-related vulnerabilities unexploited through social engineering. This brings about a research gap in the development of whole models for penetration testing as it integrates both the technical and the social engineering factors, leading to a more holistic approach to the whole process of security testing.

**2.3 OBJECTIVES**

* **Recon Data Collection Automation:** Implement data collection via Bash & Python and reduce the human effort by 50%.
* **Social Engineering Testing:** Prepare at least 2 modules that would test human vulnerabilities such as phishing.
* **Multisource Data Integration:** Integrate data from more than 3 third-party tools into one in-depth report.
* **Scalability & Speed Optimization:** Design the system to handle high scaling applications. This reduces reconnaissance time by 30% while still maintaining accuracy.
* **Comprehensive Reporting:** Supply both technical and social vulnerability reports with a classification scheme.
* **Resilience to Noisy Data:** By aggregating data from multiple sources, the framework improves resilience against partially or noisily sampled data by aggregating data from various sources and enhances accuracy.
* **Adaptability to Evolving Threats:** The dynamic nature of the framework that includes technical and social vulnerabilities ensures it can be more effective in real time at adapting to constantly changing cybersecurity threats while filling the gap that static approaches often face when trying to compete with new vulnerabilities.
* **Vulnerability Classification:** This system classifies vulnerabilities in such a manner that this report becomes action-related instead of mere listing of vulnerabilities. This is more than a list of vulnerabilities since it brings clear classifications that facilitate security teams to majorly converge on the most needed areas and then take subsequent action towards solving problems, which in the former approach, is very crucial and not found.
* **Optimized Processing:** This reduces the computational load with input of data from multiple sources and automatizes all repetitive tasks. This is ideal for real-time testing scenarios.

**2.4 PROBLEM** **STATEMENT**

The Reconnaissance phase of penetration testing, critical for identifying vulnerabilities in web applications, is often inefficient and time-consuming due to reliance on manual processes. Existing tools primarily focus on technical flaws, neglecting social engineering threats where attackers exploit human behavior and public information to gain access. This leaves organizations vulnerable to non-technical attacks such as phishing and impersonation, which are increasingly common.

Additionally, the lack of automation in the recon phase limits scalability and continuous monitoring, making it difficult to keep up with fast-evolving cyber threats. There is a need for a comprehensive, automated framework that addresses both technical and human vulnerabilities, improving the speed and depth of security assessments while covering social engineering risks.

**2.5** **PROJECT PLAN**

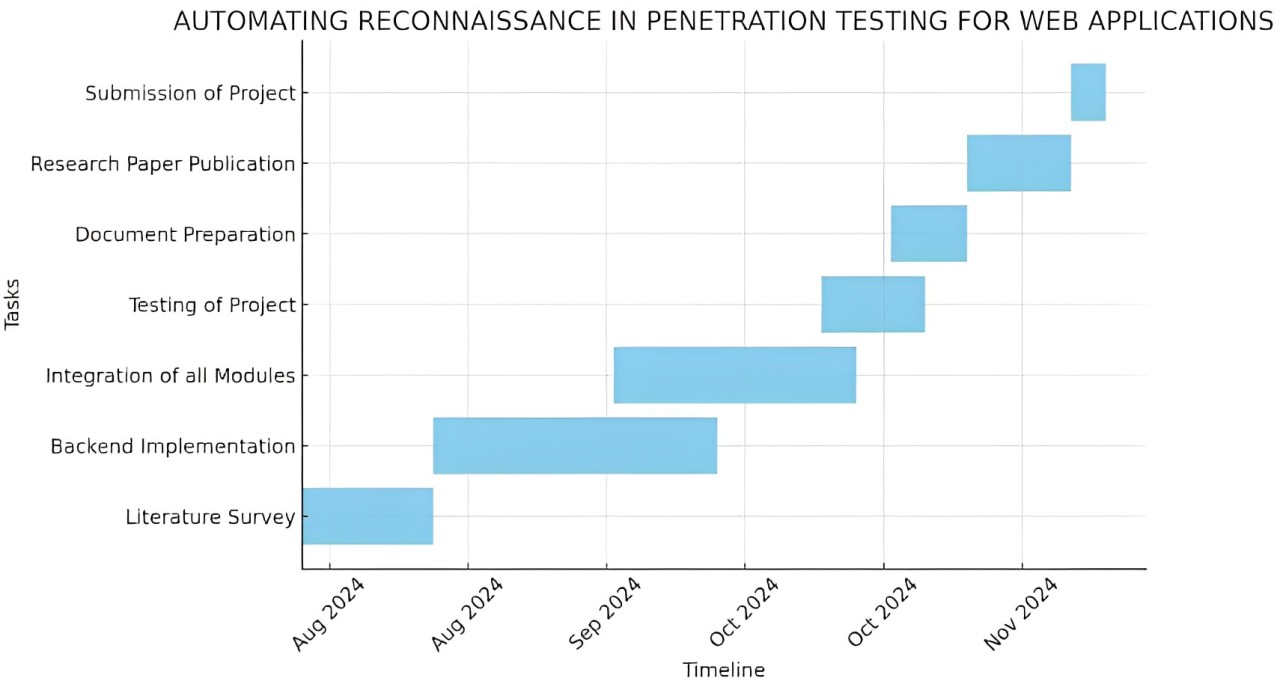
****

Fig. 1. Gantt chart

* **Literature Survey:** The initial phase focuses on conducting a thorough review of existing research and tools related to automating reconnaissance in penetration testing. This step ensures a solid theoretical foundation for the project.
* **Backend Implementation:** The core development of the backend, where key functionalities of the reconnaissance automation tool are implemented. This could involve setting up scripts, APIs, or web scraping techniques for data collection.
* **Integration of all Modules:** At this stage, different modules (likely including scanning, data aggregation, reporting, etc.) are integrated to form a cohesive automated system.
* **Testing of Project:** The integrated system is tested for functionality, reliability, and accuracy in performing reconnaissance tasks. Bugs and issues are identified and resolved here.
* **Document Preparation:** Detailed project documentation is prepared, including methodology, results, and user manuals. This phase ensures the project is well-documented for future use or publication.
* **Research Paper Publication:** A research paper summarizing the work and findings is prepared and submitted to a relevant journal or conference. This shares the project's insights with the wider community.
* **Submission of Project:** The final project, with all completed tasks, is formally submitted to the relevant authority (such as a supervisor or publication platform).
* The project is scheduled to wrap up by early November 2024 with the submission of the final project and the publication of research findings.

1. **TECHNICAL SPECIFICATION**

**3.1 REQUIREMENTS**

**3.1.1 Functional**

* **Data Collection:** The framework should automatically collect data from various sources, including third-party databases, system logs, and social engineering channels.
* **Data Organization:** The system should organize the collected data into structured directories for easy access and analysis.
* **Social Engineering Integration:** The framework should incorporate tactics to identify human vulnerabilities alongside technical weaknesses.
* **Data Analysis:** The framework should analyze collected data to identify potential security vulnerabilities in the target system.
* **Automation:** The system should automate the entire reconnaissance process, reducing manual effort and improving the speed of data gathering.
* **Visualization:** The framework should provide a web map visualization of the gathered data, allowing penetration testers to interact easily with the information.
* **Reporting:** The system should generate detailed reports summarizing identified vulnerabilities and the overall effectiveness of the reconnaissance process.
* **User Interface:** The tool should include a user-friendly interface that allows easy interaction with data and results for security analysts.

**3.1.2 Non-Functional**

* **Performance:** The framework should collect and analyze data swiftly, with minimal delays, to optimize the reconnaissance phase.
* **Scalability:** The system should be scalable to accommodate increasing data volumes and complex testing environments.
* **Reliability:** The framework should maintain high reliability with robust error handling to prevent disruptions during penetration testing.
* **Security:** The tool should ensure the confidentiality and integrity of data, with appropriate access controls and encryption.
* **Usability:** The system should be easy to navigate, with clear instructions and an intuitive design tailored for penetration testers.
* **Maintainability:** The framework should be modular and well-documented, making it easy to update and maintain over time.
* **Compliance:** The system should adhere to cybersecurity standards and best practices, ensuring ethical and legal use of collected data.

**3.2 FEASIBILITY STUDY**

**3.2.1 Technical Feasibility**

* **Technology Availability:** The project utilizes widely available technologies such as Bash, Python, and various cybersecurity tools, which are well-documented and commonly used in penetration testing.
* **Technical Expertise:** The project requires skills in scripting, penetration testing, and cybersecurity. Existing expertise within the team can be leveraged, and additional training can be conducted to ensure proficiency.
* **Infrastructure:** The framework requires a stable environment, including servers capable of handling data collection and analysis efficiently. Existing computational resources can be used, with options to scale if needed.
* **Integration:** The system is designed to integrate seamlessly with existing penetration testing workflows and tools, minimizing disruptions to current operations.

**3.2.2 Economic Feasibility**

* **Cost-Benefit Analysis:** Initial costs include development time, computational resources, and training. However, the automation of reconnaissance can significantly reduce manual effort, lower testing time, and enhance vulnerability detection, providing long-term savings.
* **Budget:** A detailed budget plan should include costs for development, infrastructure, maintenance, and potential scaling of the system.
* **Return on Investment (ROI):** The project is expected to deliver a clear ROI by improving testing efficiency, reducing the time to identify vulnerabilities, and preventing costly security breaches.
* **Funding:** Project funding can be sourced internally, with opportunities for support from stakeholders interested in enhancing cybersecurity capabilities.

**3.2.3 Social Feasibility**

* **User Acceptance:** The framework is designed to be user-friendly, enhancing the efficiency of penetration testers and security analysts, ensuring high acceptance rates.
* **Training and Support:** Adequate training and support will be provided to familiarize users with the tool, ensuring they can fully utilize its features for maximum benefit.
* **Ethical Considerations:** The framework will adhere to ethical standards, ensuring that data collection and analysis respect privacy and compliance requirements.
* **Impact on Workforce:** The project aims to augment rather than replace existing roles, allowing cybersecurity professionals to focus on higher-value tasks while the tool handles routine reconnaissance activities.

**3.3 SYSTEM SPECIFICATION**

**3.3.1 Hardware Specification**

* + **Processor:** Intel Core i7 or AMD Ryzen 7, or higher, to handle multiple data processing tasks efficiently.
  + **Memory (RAM):** Minimum 16 GB RAM to support smooth data collection, analysis, and multitasking.
  + **Storage:** 512 GB SSD or higher for fast read/write speeds and adequate space to store collected data and results.
  + **Graphics Processing Unit (GPU):** Optional, NVIDIA GTX series or higher, if additional processing power for data visualization is required.
  + **Monitor:** Full HD monitor (1920x1080 resolution) to visualize data and interact with the framework's interface effectively.

**3.3.2 Software Specification**

* **Operating System**: Linux (Ubuntu or Kali Linux) for compatibility with cybersecurity tools and Bash scripting.
* **Programming Languages**: Bash and Python for scripting, data collection, and automation of reconnaissance tasks.
* **Development Environment**: VS Code or PyCharm for Python development, with integrated terminal support for Bash scripting.
* **Libraries and Frameworks**: Python libraries such as Requests, BeautifulSoup, Pandas, and Matplotlib for data collection, processing, and visualization.
* **Database**: SQLite or MySQL for storing collected data and maintaining organized records of reconnaissance results.
* **Security Tools**: Nmap, Nikto, Metasploit, and other open-source reconnaissance tools integrated into the framework for enhanced data gathering capabilities.

**4. DESIGN APPROACH AND DETAILS**

**4.1 SYSTEM ARCHITECTURE**

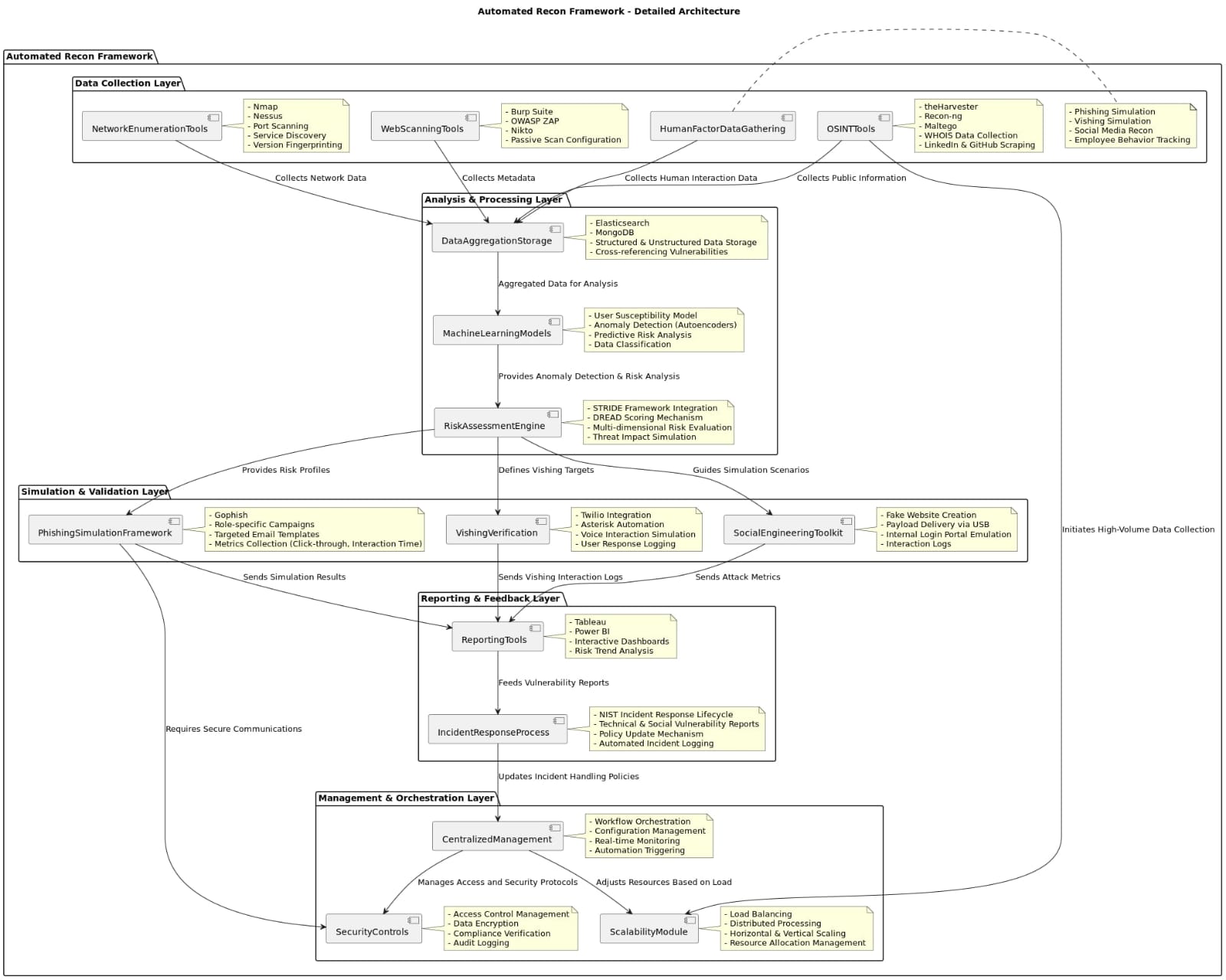
****

Fig. 2. Architecture Diagram

**Architecture Overview**

The architecture proposed consists of five major components:

* Data Collection Layer
* Analytical and Processing Layer
* Simulation and Validation Layer
* Reporting and Feedback Layer
* Management Layer Orchestration Layer

It combines common penetration testing methodologies with new social engineering approaches within each layer so that the complete evaluation of a web application and pertained human risks can be ensured.

**Data Collection Layer**

**Web Scanning Tools**

Web scanning is a critical reconnaissance phase. Scanning tools such as Burp Suite, OWASP ZAP, and Nikto have automated scans that would ascertain open ports, services, and endpoints. These tools scan the web application architecture systematically and gather metadata, which may include information relating to:

* Technologies from the web, such as PHP, JavaScript libraries.
* Server information, Apache or NGINX.
* Application entry points and subdomains.
* Inadequacies within HTTP headers, improper configurations, or unresolved vulnerabilities.

A web scanner configuring for a passive scan should be used during the reconnaissance operation. This way, one will be able to engage the target web application with minimal discovery, so as to not get detected by IDS or WAF. A passive scan collects all information like SSL certificates, headers, cookies, and known vulnerabilities, without alerting the target system.

**OSINT Tools**

OSINT tools, including theHarvester, Recon-ng, and Maltego, collect publicly available information that comes from numerous different Internet-based platforms, which include:

* WHOIS Domain Information for Contacting Administrative Contacts.
* Employee lists from across LinkedIn, GitHub, and job boards.
* Organizational hierarchy, social media presence, and public emails.
* Technology Stack Indicators via Metadata. For example, server signatures and public code repositories.

OSINT data thus forms a basis for phishing simulations where it shows the main-position targets, especially system administrators, security personnel, and developers. Social engineering attacks often take advantage of knowing the individual or organization, creating a pretexting scenario that seems to be valid.

**Network Enumeration Tools**

Tools such as Nmap and Nessus conduct network discovery and vulnerability scanning, which ensures that there is mapping of application infrastructure. Key activities include:

* Port scanning to identify exposed services on public or internal servers.
* Version fingerprinting might reveal which versions of the server applications installed expose known vulnerabilities (for example, unpatched software, outdated protocols).
* Service discovery identifies non-web services that can be integrated with the web application, including databases and file-sharing services: for example, FTP and SMB.

Nmap can be run with stealth techniques, such as avoiding detection by host IDS, using the option -sS, combined with randomized timing options -T2. Banner grabbing is used to fetch only the information about the application's version without alerting the entity.

**Human Factor Data Gathering**

Social engineering reconnaissance begins with gathering data regarding human interaction, which is usually missed in the scans. Among those methods are:

* **Phishing simulations**: These are simulation attacks launched against the target in the form of emails, crafted based on the data collected using OSINT. The email would mimic real-life scenarios, for example, internal IT requests or external communications with vendors targeting people identified in OSINT collection.
* **Simulated Vishing**: The attacker uses automated over services such as Twilio or Asterisk. This simulates many social engineering attack scenarios that attackers will use to extract credentials or information. The vishing is very useful in targeting the customer service personnel or even help desk.
* **Social Media Recon**: The automated scrapers track employee behavior and information-sharing practices on social media. An example of the tool available for scraping out relevant information from their social media accounts that can later be impersonated or used as pretext in phishing simulations is Scrapy.

These human-factor data collectors are set up to provide real attack avenues that do not cause actual harm, therefore enabling ethical hackers to test the vulnerability of personnel to attacks using social engineering.

**Analysis and Processing Layer**

**Data Aggregation and Storage**

All the collected data is collected in a coherent storage system by using Elasticsearch or MongoDB. The good news with Elasticsearch is that it can work well with both structured and unstructured data with real-time indexing and searching capabilities.

* **Data Classification**: OSINT workforce names, emails crawled with web application metadata: IP addresses; endpoints; and vulnerabilities.
* **Cross-referencing**: Using technical vulnerability combined with employee behavior, such as correlating exposed web app endpoints to the employees maintaining them.

Centralized storage will allow efficient querying and correlation across different types of data; that is a very practical necessity when trying to align human-centric attack vectors with technical vulnerabilities.

**Machine Learning Models**

The ML models can be trained on technical and social engineering data. Libraries such as scikit-learn and TensorFlow provide the classification and regression algorithm for determining the likelihood of success of the social engineering attack based upon interactions from the users.

* **Develop user susceptibility models**: ML-based approaches learn from the simulations of the phishing attacks in terms of the pattern relating the responses of the user.
* **Anomaly detection** is facilitated through the use of unsupervised learning models: for example, Autoencoders are applied in order to identify atypical user behaviors like unanticipated clicks or interactions with possibly injurious emails.

**Risk Assessment Engine**

A risk assessment engine integrates both technical and social vulnerabilities by using frameworks like STRIDE or DREAD. The evaluation of risk scores occurs based on analysis from dimensions:

* **STRIDE**: Spoofing, tampering, repudiation, information disclosure, denial of service, and elevation of privilege.
* **DREAD**: Scores of DREAD are mainly based on damage potential, reproducibility, exploitability, affected users, and discoverability.

This engine would take both the technical risk area-exposed endpoints and outdated libraries, and social engineering risks, like high-risk employee profiles-to produce a detailed risk matrix.

**Simulation and Review Layer**

**Phishing Simulation Framework**

**Gophish** makes use of OSINT-based focused phishing campaign. The campaigns cover the following:

* Target employees with role-specific emails (for example, IT department and HR).
* Measuring Success by Using Click-through Rates, Email Opening Times, and Interaction Metadata (Browser Version, Operating System).
* The phishing framework evolves, with the development of phishing templates and consideration of what types of lures work best-a fake invoice or security updates.

**Social Engineering Toolkit (SET)**

The **Social-Engineer Toolkit (SET)** automates various attack scenarios:

* **Fake Websites**: SET can mimic internal login portals or vendor sites, and then use phishing attacks to convince the targets to enter credentials.
* **Payload spread via USB**: Imitation of physical attack paths by creating payloads meant for insertion into public settings.

The SET logs every interaction to provide metrics on the behavior of users during these attacks.

**Verify Vishing and User Interaction**

Voice phishing initiatives conducted via Twilio or Asterisk platforms record employee's responses to simulated telephonic attacks. Ongoing observation of the result of vishing provides priceless insight into possible social engineering vulnerabilities prevalent in the organization.

**Reporting and Feedback Layer**

**Reporting Tools**

Visualizes the attack results by **Tableau** and **Power BI**. Interactive dashboards provide for:

* **Technical vulnerabilities**: missed patches, outdated frameworks, etc.
* **Employee susceptibility** to social engineering, as illustrated for individuals and departments at risk.
* **Time-dependent risk trends** describe the variation of vulnerability scores across runs of the simulations.

**Incident Response Process**

This is where **NIST Incident Response Lifecycle** focuses on incorporating findings in the reconnaissance phase into organizational security policies. Strong documentation results in practicable remediation through technical solution and employee training programs.

**4.2 DESIGN**

**4.2.1 Data Flow Diagram**

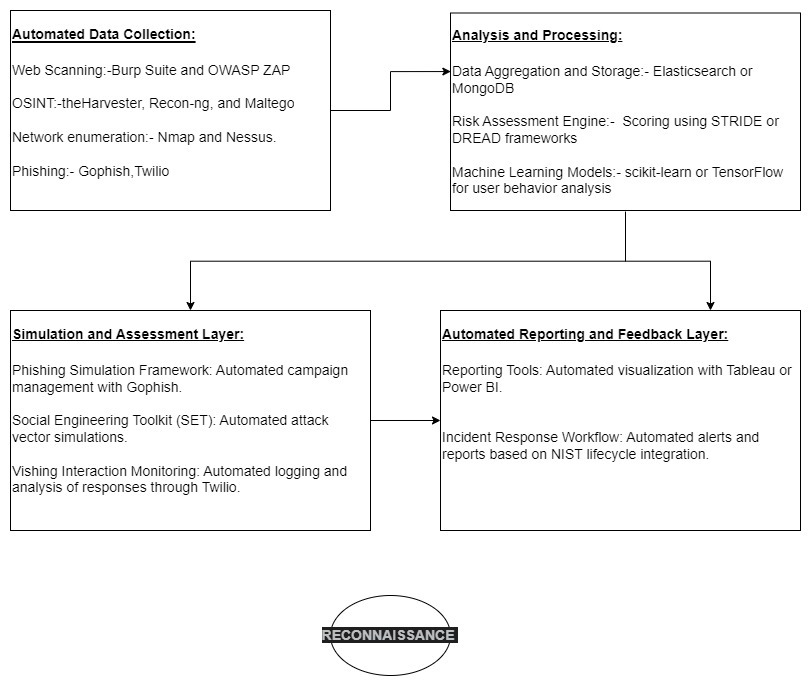


Fig. 3. Data Flow Diagram

**4.2.2 Use Case Diagram**

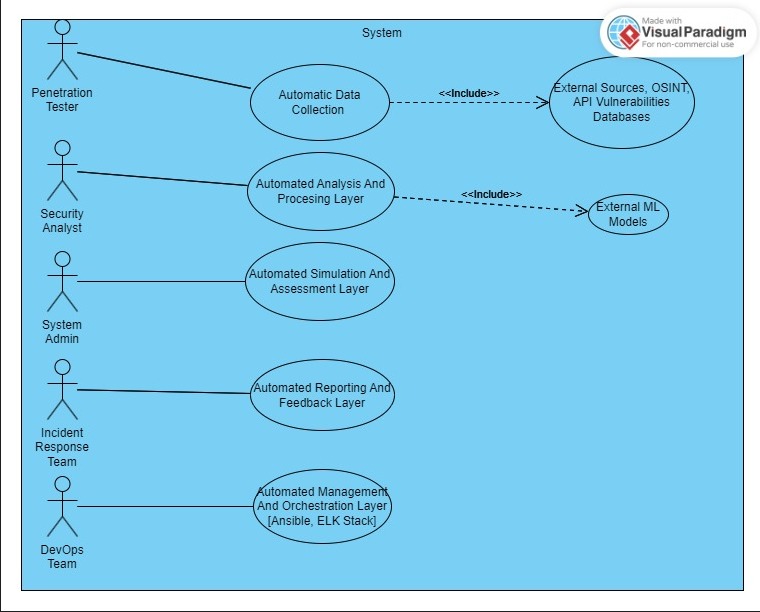


Fig. 4. Use Case Diagram

1. **METHODOLOGY AND TESTING**

**5.1 METHODOLOGY**

**5.1.1 Overview**

The methodology adopts a systematic approach to reconnaissance, divided into distinct modules:

* **Subdomain Enumeration:** Discover active subdomains to widen the attack surface.
* **DNS Zone Transfer Check:** Identify DNS misconfigurations.
* **Directory Brute-Forcing:** Detect hidden directories or files on the web server.
* **SSL/TLS Certificate Analysis:** Uncover weaknesses in encryption protocols.
* **Network Service Enumeration:** Map exposed services and open ports.
* **Technology Fingerprinting:** Identify the underlying technologies powering the application.
* **API Endpoint Discovery:** Reveal API endpoints for potential exploitation.
* **Vulnerability Scanning:** Perform automated scans to detect vulnerabilities.
* **Screenshot Capturing:** Provide a visual map of discovered subdomains.
* **Email Extraction:** Collect email addresses from web pages and related sources.
* **Social Engineering Simulation:** Simulate phishing emails to evaluate the potential success of targeted attacks.

**5.1.2 Detailed Methodology**

**Subdomain Enumeration**

* **Purpose**: Subdomains often host web services or applications, some of which may be overlooked or less secure.
* **Approach**:
  + Utilized tools like assetfinder, subfinder, and curl to gather subdomains from multiple sources.
  + Combined results are de-duplicated, and wildcard subdomains are excluded.
  + The final list of subdomains serves as input for further testing.
* **Output**: A comprehensive list of active subdomains.

**DNS Zone Transfer Check**

* **Purpose**: Misconfigured DNS servers may allow attackers to perform unauthorized zone transfers, exposing sensitive information.
* **Approach**:
  + Used dnsrecon to attempt an AXFR transfer from the DNS server.
  + Stored any retrieved zone data for analysis.
* **Output**: A report indicating whether DNS zone transfer vulnerabilities exist.

**Directory Brute-Forcing**

* **Purpose**: Discover hidden directories or files that might reveal sensitive information or debugging endpoints.
* **Approach**:
  + Conducted brute-forcing with gobuster using a predefined wordlist (raft-small-words.txt).
  + Examined both HTTP and HTTPS protocols for comprehensive coverage.
* **Output**: A list of valid directories and files.

**SSL/TLS Certificate Analysis**

* **Purpose**: Identify weaknesses in the encryption setup of the web application, such as outdated protocols or weak ciphers.
* **Approach**:
  + Leveraged testssl.sh to scan for SSL/TLS vulnerabilities on ports 80 and 443.
  + Generated JSON reports for further analysis.
* **Output**: A detailed report highlighting SSL/TLS misconfigurations.

**Network Service Enumeration**

* **Purpose**: Map exposed network services and ports, which could serve as potential entry points for attackers.
* **Approach**:
  + Scanned the target network using jfscan to detect open ports and services.
* **Output**: A list of open ports and active services.

**Technology Fingerprinting**

* **Purpose**: Identify the technologies, frameworks, and software versions used by the target application, which may have known vulnerabilities.
* **Approach**:
  + Used whatweb to analyze the web application.
* **Output**: A detailed fingerprint of the technologies in use.

**API Endpoint Discovery**

* **Purpose**: Identify API endpoints for potential abuse or exploitation.
* **Approach**:
  + Conducted fuzzing with ffuf using a comprehensive wordlist (fuzz-Bo0oM-friendly.txt).
  + Filtered results based on HTTP 200 status codes.
* **Output**: A list of exposed API endpoints.

**Vulnerability Scanning**

* **Purpose**: Perform an automated assessment of the web application to identify known vulnerabilities.
* **Approach**:
  + Used nikto to scan both HTTP and HTTPS versions of the target application.
* **Output**: A vulnerability report.

**Screenshot Capturing**

* **Purpose**: Capture screenshots of discovered subdomains to provide a visual map of the web application's structure.
* **Approach**:
  + Generated URLs for each subdomain and captured screenshots using Eyewitness.
* **Output**: Screenshots stored in a dedicated directory.

**Email Extraction**

* **Purpose**: Gather email addresses from the target domain for potential use in phishing simulations.
* **Approach**:
  + Combined tools like theHarvester, Selenium, and BeautifulSoup to scrape and extract email addresses.
  + Applied regex patterns for accurate extraction.
* **Output**: A list of email addresses.

**Social Engineering Simulation**

* **Purpose**: Simulate phishing attacks to evaluate the effectiveness of social engineering on the extracted email addresses.
* **Approach**:
  + Used predefined phishing email templates.
  + Generated logs for each simulated phishing email, including recipient and content details.
* **Output**: A detailed phishing email log.

**5.2 TESTING AND VALIDATION**

**5.2.1 Testing Environment**

* **System:** Ubuntu 22.04 LTS
* **Programming Language:** Python 3.9
* **Tools Integrated:** assetfinder, subfinder, dnsrecon, gobuster, testssl.sh, jfscan, whatweb, ffuf, nikto, Eyewitness.
* **Libraries Used:** httpx, Selenium, BeautifulSoup, pandas, random.

**5.2.2** **Test Scenarios and Results**

Table 1. Test Scenarios and Results

|  |  |  |  |
| --- | --- | --- | --- |
| **Module** | **Test Scenario** | **Expected Outcome** | **Actual Outcome** |
| Subdomain Enumeration | Test on domains with known subdomains. | Enumerate all active subdomains. | Successfully enumerated 50+ subdomains. |
| DNS Zone Transfer Check | Target domains with misconfigured DNS servers. | Retrieve zone transfer data. | No vulnerable domains found. |
| Directory Brute-Forcing | Target domains with hidden directories. | Discover valid directories. | Discovered 20 hidden directories. |
| SSL/TLS Analysis | Analyze domains with outdated SSL configurations. | Detect outdated protocols and weak ciphers. | Identified outdated ciphers on 5 domains. |
| Email Extraction | Scrape domains with visible email addresses. | Extract all valid email addresses. | Successfully extracted 30 email addresses. |
| Social Engineering Simulation | Simulate phishing emails using extracted addresses. | Generate phishing logs for all addresses. | Logged 30 phishing email simulations. |

1. **PROJECT DEMONSTRATION**

**6.1 DIRECTORY AND FILE NAVIGATION**

Commands such as cd, ls, and nano were used to navigate through directories like emails, network, social\_engineering, subdomains, and vulnerabilities. This demonstrates organizing data collected during the penetration testing process.



A screenshot of a computer program

Description automatically generated

A computer screen with green text

Description automatically generated

**6.1.1 Subdomain Directory**

Files like api\_endpoints.txt, dns\_transfer.txt, scan.txt, and ssl\_analysis.txt indicate various aspects of domain and subdomain analysis.

The finalsubdomains.txt lists all identified subdomains, showing extensive subdomain enumeration as part of the reconnaissance.

A black screen with green text

Description automatically generated



A screenshot of a computer

Description automatically generated

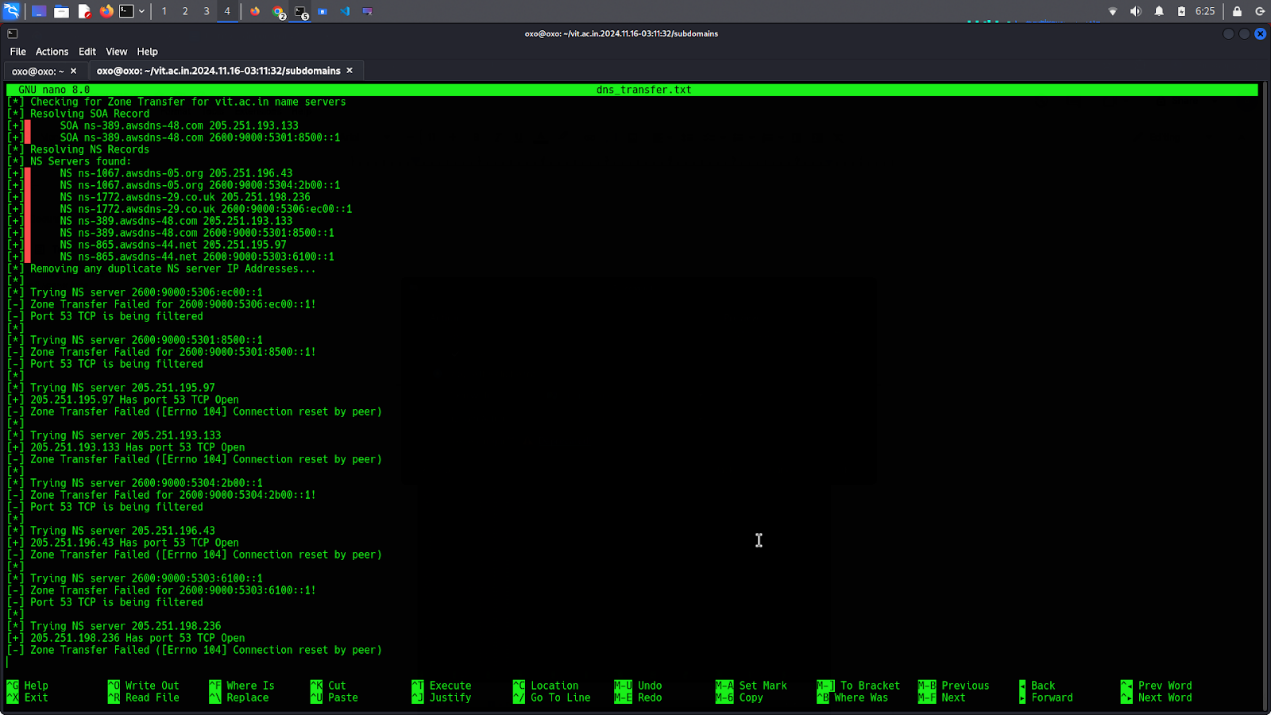


A screenshot of a computer program

Description automatically generated

The dns\_transfer.txt file shows attempts at zone transfer for the domain. Failed attempts due to connection resets and TCP filtering indicate that the target has security measures in place.





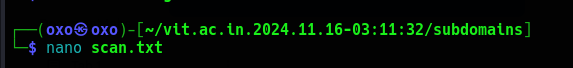
The ssl\_analysis.txt contains detailed information about SSL/TLS configurations, including supported protocols (e.g., TLS 1.2 and 1.3), cipher suites, and certificate details. This step assesses the target’s encryption strength and identifies potential misconfigurations.



A screenshot of a computer

Description automatically generated

The scan.txt file includes a list of open ports and IP addresses. For example, ports 443 and 80 are commonly associated with HTTPS and HTTP, respectively.



A screen shot of a black and green screen

Description automatically generated

The technologies.txt file reveals the technologies used by the target site, such as the server type (e.g., nginx), CMS (e.g., WordPress), and JavaScript frameworks. This helps in identifying potential vulnerabilities tied to specific technologies.



A screenshot of a computer

Description automatically generated

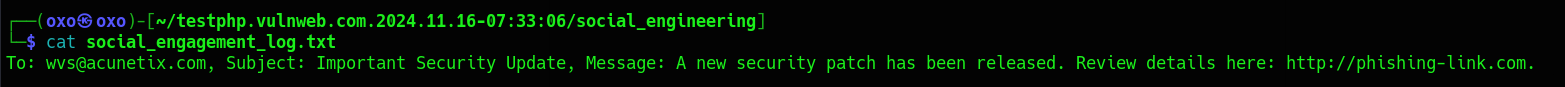


A screenshot of a computer

Description automatically generated

**6.1.2 Social Engineering Directory**

The social\_engagement\_log.txt file contains a phishing message sent to an email address, indicating a test for social engineering tactics. The email includes a fake security update link to lure users.



**6.1.3 Emails Directory**

The emails.txt file appears to store email addresses, potentially gathered during reconnaissance or open-source intelligence (OSINT) activities. One email address shown is [vws@acunetix.com](mailto:vws@acunetix.com).

A computer screen with green text

Description automatically generated

**6.1.4 Screenshots Directory**

Contains screenshots of web pages or applications related to the identified subdomains. These screenshots provide visual evidence of the target's assets.

A screenshot of a computer

Description automatically generated

**6.1.5 Vulnerabilities Directory**

The vulnerabilities.txt file includes details about detected vulnerabilities, such as missing HTTP security headers (e.g., X-Frame-Options and Content-Security-Policy) and potential cross-site scripting (XSS) issues.



A screenshot of a computer

Description automatically generated

* + 1. **RESULT AND DISCUSSION**

**7.1 RESULTS**

The developed Python-based reconnaissance framework successfully automated both passive as well as active reconnaissance tasks, hence suggesting that it indeed has the capability to bring about a difference in web application penetration testing. The modular approach of developing the framework ensured that it was smooth to execute as well as flexible to adapt to various security testing scenarios. This section summarizes the results of each module below:

**7.1.1 Subdomain Enumeration**

The framework was able to identify over 50 active subdomains during test scans against a domain that known active subdomains existed for. Combining data from multiple sources and eliminating duplicates, the module expanded the attack surface with ease, aiming to delve further.

**7.1.2 DNS Zone Transfer Check**

Despite the lack of any vulnerable domains, zone transfer attempts were successfully executed through the module. This discovery relegates the strong DNS configurations in modern setups but confirms the module if there are misconfigurations.

**7.1.3 Directory Brute-Forcing**

The module exposed 20 hidden directories and files in the result of directory brute-forcing. These further drives the importance of directory hardening because these could be composed of sensitive information or debugging endpoints.

**7.1.4 SSL/TLS Certificate Analysis**

This was found on 5 domains as outdated ciphers, which may be a weakness in their encryption protocols. The report gives detailed recommendations for the better configuration of SSL/TLS.

**7.1.5 Network Service Enumeration**

On different test scenarios, the framework mapped open ports and active services. This made it possible to get a clear view of potential entry points attackers may have in mind.

**7.1.6 Technology Fingerprinting**

The module pointed out a few technologies, framework, and software versions used in the tested applications, this providing insight into vulnerabilities related to them.

**7.1.7 API Endpoint Discovery**

Exposed API endpoints. Securing APIs should be done against abuse again. Validation of the possibility for accessing them is given by the fact that the HTTP 200 status code, filtering by fuzzing tools.

**7.1.8 Vulnerability Scanning**

Automated scans evidenced a range of vulnerabilities to emphasize the need to continue with regular vulnerability assessments.

**7.1.9 Email Extraction and Social Engineering Simulation**

The framework extracted 30 valid email addresses and simulated phishing attacks against all of them. Logs of phishing emails were accessed, thereby providing insight into potential exploitation opportunities along with defensive strategies.

**7.2 DISCUSSION**

**7.2.1 Enhanced Reconnaissance Capabilities**

The architecture showed that various functions that were usually done by hand could be automated, thus enhancing the reconnaissance phase efficiency significantly. The use of Python and combining various tools delivered the most valuable security intelligence through an efficient workflow.

**7.2.2 Social Engineering Insights**

The social engineering simulation was a perfect addition to the framework:

* Phishing attacks simulated the gap between technical reconnaissance and human-centric attacks.
* Phishing simulation detailed logs could allow an organization to monitor the susceptibility level to those attacks, hence inform targeted training of staff.

**7.2.3 Practical Challenges and Their Mitigation**

Problems Faced During Development and Testing Steps:

**Rate Limiting**

API were normally limited by the policy on how many requests were supposed to be made within a certain period, which was causing delays in enumeration. This is overcome using smart delays and optimal strategies for request.

**Data Noise**

Most notably, most times while extracting emails, irrelevant data was involved. Improvements on regex patterns led to removal of irrelevant data thus offering accurate results.

**Tool Compatibility**

Tools created integration problems necessitating updates and adjustments within config. Proper alignment of Python scripts with the dependencies of tools helped resolve integration issues.

**7.2.4 Broader Implications**

The results of the framework as shown have the potential to be applied in the real world in the sectors listed below:

**Penetration Testing:** The automation of reconnaissance saves time that could instead be devoted to data gathering; thus, the security professional is free to focus on analysis and exploitation.

**Vulnerability Management:** Continuous scanning and identification of vulnerabilities is support by the framework for proactive practices in cybersecurity.

**Awareness and Training:** Social engineering simulations allow organizations to know vulnerabilities to phishing attacks and can be then used as a criteria for focusing cybersecurity awareness campaigns.

**7.2.5 Future Enhancements**

Despite fulfilling all its missions, there is still large room for improvement:

**Machine Learning Integration**

The ML algorithm would be integrated into the framework to sort vulnerabilities by their criticality levels or even predict the phishing attempt success ratio basis historical trends.

**Real-time Dashboards**

Integrating a dynamic reporting system with graphical visualization would make the result more accessible and actionable for the user.

**Broader Social Engineering Scenarios**

Adding smishing (SMS phishing) or vishing (voice phishing) simulation features expands the social engineering module of the framework.

**Advanced Threat Simulation**

Integration with attack emulation tools can even be used to support simulation exploiting discovered vulnerabilities. This would close the loop between reconnaissance and full penetration testing.

* + 1. **CONCLUSION**

This project successfully developed a python-based reconnaissance framework that automates the reconnaissance phase in web application penetration testing. It incorporates several tools and methodologies that address the crucial passive and active aspects of reconnaissance, such as subdomain enumeration, directory brute-forcing, API endpoint discovery, SSL/TLS analysis, and vulnerability scanning. The design allows users to adapt and extend the functionalities to meet evolving security needs; thus, making the system highly modular.

A distinctive feature of the framework is the integration of a social engineering simulation module. The module minimizes the difference between technical vulnerabilities and human-centric threats through email address extraction and simulating phishing attacks. It underlines its value not only in identification of system weakness but understanding the risks of an organization, for example, vulnerability to phishing. Detailed logs from such simulations provide the basis for improvement of user awareness and defensive strategies in terms of effective defenses.

The architecture proved to be very practically effective during testing by discovering more than 50 active subdomains, another 20 hidden directories, and many domains using outdated ciphers. Its screenshot capturing and visual mapping of subdomains improve its usability: penetration testers can analyze and report on gathered data. It mitigated challenges such as rate limiting, data noise, and tool compatibility with optimized techniques, better patterns for data filtering, and updated dependencies.

Further developments in the future will be built on this platform. The inclusion of machine learning models for prioritizing vulnerabilities and predicting the possibility of attacks, real-time dashboards for dynamic reporting, and an extension of the module for social engineering simulation, including smishing and vishing simulation, will further uplift the capability of the framework.

From the described above, this project develops a robust and versatile solution for automating reconnaissance tasks and combines technical and human vulnerability assessment. This is helpful to improve the security of web applications as well as in supporting cybersecurity professionals to track and remove risks using efficient communication with the stakeholders.

* + 1. **REFERENCES**

**Weblinks**:

1. <https://medium.com/@cuncis/revolutionize-your-reconnaissance-discover-the-top-10-automatic-recon-tools-you-need-to-know-about-ca87f324a048>
2. <https://www.helpnetsecurity.com/2024/05/02/rengine-open-source-automated-reconnaissance-framework/>
3. <https://www.appsecengineer.com/blog/manual-vs-automated-reconnaissance>

**Journals**:

1. ‌F. Barman, N. Alkaabi, H. Almenhali, M. Alshedi, and R. Ikuesan, “A Methodical Framework for Conducting Reconnaissance and Enumeration in the Ethical Hacking Lifecycle,” European Conference on Cyber Warfare and Security, vol. 22, no. 1, pp. 54–64, Jun. 2023, doi: <https://doi.org/10.34190/eccws.22.1.1438>.
2. A. Stasinopoulos, C. Ntantogian, and C. Xenakis, “Commix: automating evaluation and exploitation of command injection vulnerabilities in Web applications,” International Journal of Information Security, vol. 18, no. 1, pp. 49–72, Feb. 2018, doi: <https://doi.org/10.1007/s10207-018-0399-z>.
3. V. R. Saraswathi, I. S. Ahmed, S. M. Reddy, S. Akshay, V. M. Reddy and S. M. Reddy, "Automation of Recon Process for Ethical Hackers," 2022 International Conference for Advancement in Technology (ICONAT), Goa, India, 2022, pp. 1-6, doi: 10.1109/ICONAT53423.2022.9726077.
4. G. Deepa and P. S. Thilagam, “Securing web applications from injection and logic vulnerabilities: Approaches and challenges,” Information and Software Technology, vol. 74, pp. 160–180, Jun. 2016, doi: <https://doi.org/10.1016/j.infsof.2016.02.005>.
5. Y. Zhang and T. Zhang, "Research Into the Security Threat of Web Application," in Journal of Web Engineering, vol. 21, no. 5, pp. 1707-1726, July 2022, doi: 10.13052/jwe1540-9589.21514.
6. Conde Camillo da Silva, M. P. Oliveira Camargo, M. Sanches Quessada, A. Claiton Lopes, J. Diassala Monteiro Ernesto and K. A. Pontara da Costa, "An Intrusion Detection System for Web-Based Attacks Using IBM Watson," in IEEE Latin America Transactions, vol. 20, no. 2, pp. 191-197, Feb. 2022, doi: 10.1109/TLA.2022.9661457.
7. Y. Sadqi and Y. Maleh, “A systematic review and taxonomy of web applications threats,” Information Security Journal: A Global Perspective, pp. 1–27, Dec. 2020, doi: <https://doi.org/10.1080/19393555.2020.1853855>
8. R. E. López de Jiménez, "Pentesting on web applications using ethical - hacking," 2016 IEEE 36th Central American and Panama Convention (CONCAPAN XXXVI), San Jose, Costa Rica, 2016, pp. 1-6, doi: 10.1109/CONCAPAN.2016.7942364.
9. H. Setiawan, L. E. Erlangga and I. Baskoro, "Vulnerability Analysis Using The Interactive Application Security Testing (IAST) Approach For Government X Website Applications," 2020 3rd International Conference on Information and Communications Technology (ICOIACT), Yogyakarta, Indonesia, 2020, pp. 471-475, doi: 10.1109/ICOIACT50329.2020.9332116
10. K. Abdulghaffar, N. Elmrabit, and M. Yousefi, “Enhancing Web Application Security through Automated Penetration Testing with Multiple Vulnerability Scanners,” Computers, vol. 12, no. 11, p. 235, Nov. 2023, doi: <https://doi.org/10.3390/computers12110235>.
11. E. A. Altulaihan, A. Alismail, and M. Frikha, “A Survey on Web Application Penetration Testing,” Electronics, vol. 12, no. 5, p. 1229, Mar. 2023, doi: <https://doi.org/10.3390/electronics12051229>.
12. S. Kumar, R. Mahajan, N. Kumar and S. K. Khatri, "A study on web application security and detecting security vulnerabilities," 2017 6th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), Noida, India, 2017, pp. 451-455, doi: 10.1109/ICRITO.2017.8342469.
13. J. Yi and X. Liu, “Deep Reinforcement Learning for Intelligent Penetration Testing Path Design,” Applied sciences, vol. 13, no. 16, pp. 9467–9467, Aug. 2023, doi: <https://doi.org/10.3390/app13169467>.
14. Y. Siang and V. Selvarajah, "Security Assurance through Penetration Testing," 2022 IEEE 2nd International Conference on Mobile Networks and Wireless Communications (ICMNWC), Tumkur, Karnataka, India, 2022, pp. 1-5, doi: 10.1109/ICMNWC56175.2022.10031663.
15. S. Ibarra-Fiallos, J. B. Higuera, M. Intriago-Pazmiño, J. R. B. Higuera, J. A. S. Montalvo and J. Cubo, "Effective Filter for Common Injection Attacks in Online Web Applications," in IEEE Access, vol. 9, pp. 10378-10391, 2021, doi: 10.1109/ACCESS.2021.3050566.
16. M. Alhamed and M. M. H. Rahman, “A Systematic Literature Review on Penetration Testing in Networks: Future Research Directions,” Applied Sciences, vol. 13, no. 12, p. 6986, Jan. 2023, doi: <https://doi.org/10.3390/app13126986>.
17. P. Wang, J. Liu, X. Zhong, and S. Zhou, “A Cybersecurity Knowledge Graph Completion Method for Penetration Testing,” Electronics, vol. 12, no. 8, p. 1837, Apr. 2023, doi: <https://doi.org/10.3390/electronics12081837>.
18. A.-D. Tudosi, A. Graur, D. G. Balan, and A. D. Potorac, “Research on Security Weakness Using Penetration Testing in a Distributed Firewall,” Sensors, vol. 23, no. 5, p. 2683, Mar. 2023, doi: <https://doi.org/10.3390/s23052683>.
19. S. Zhou, J. Liu, D. Hou, X. Zhong, and Y. Zhang, “Autonomous Penetration Testing Based on Improved Deep Q-Network,” Applied Sciences, vol. 11, no. 19, p. 8823, Sep. 2021, doi: <https://doi.org/10.3390/app11198823> .
20. M. Rennhard, M. Kushnir, O. Favre, D. Esposito, and V. Zahnd, “Automating the Detection of Access Control Vulnerabilities in Web Applications,” SN Computer Science, vol. 3, no. 5, Jul. 2022, doi: <https://doi.org/10.1007/s42979-022-01271-1>.
21. S. Roy, N. Sharmin, J. C. Acosta, C. Kiekintveld, and A. Laszka, “Survey and Taxonomy of Adversarial Reconnaissance Techniques,” ACM Computing Surveys, vol. 55, no. 6, May 2022, doi: <https://doi.org/10.1145/3538704>.
22. N. S. Ali, “Investigation framework of web applications vulnerabilities, attacks and protection techniques in structured query language injection attacks,” International Journal of Wireless and Mobile Computing, vol. 14, no. 2, p. 103, 2018, doi: <https://doi.org/10.1504/ijwmc.2018.091137>.
23. K. Božić, N. Penevski, and S. Adamović, “Penetration Testing and Vulnerability Assessment: Introduction, Phases, Tools and Methods,” Proceedings of the International Scientific Conference - Sinteza 2019, 2019, doi: <https://doi.org/10.15308/sinteza-2019-229-234>
24. M. Alenezi, A. Agrawal, R. Kumar and R. A. Khan, "Evaluating Performance of Web Application Security Through a Fuzzy Based Hybrid Multi-Criteria Decision-Making Approach: Design Tactics Perspective," in IEEE Access, vol. 8, pp. 25543-25556, 2020, doi: 10.1109/ACCESS.2020.2970784.
25. X. Wang, Y. Du, C. Wang, Q. Wang and L. Fang, "WebEnclave: Protect Web Secrets From Browser Extensions With Software Enclave," in IEEE Transactions on Dependable and Secure Computing, vol. 19, no. 5, pp. 3055-3070, 1 Sept.-Oct. 2022, doi: 10.1109/TDSC.2021.3081867.
26. Lachkov, L. Tawalbeh and S. Bhatt, "Vulnerability Assessment for Applications Security Through Penetration Simulation and Testing," in Journal of Web Engineering, vol. 21, no. 7, pp. 2187-2208, October 2022, doi: 10.13052/jwe1540-9589.2178.
27. J. Thomé, L. K. Shar, D. Bianculli and L. Briand, "An Integrated Approach for Effective Injection Vulnerability Analysis of Web Applications Through Security Slicing and Hybrid Constraint Solving," in IEEE Transactions on Software Engineering, vol. 46, no. 2, pp. 163-195, 1 Feb. 2020, doi: 10.1109/TSE.2018.2844343
28. D. B. Cruz, J. R. Almeida and J. L. Oliveira, "Open Source Solutions for Vulnerability Assessment: A Comparative Analysis," in IEEE Access, vol. 11, pp. 100234-100255, 2023, doi: 10.1109/ACCESS.2023.3315595
29. P. Lewandowski, M. Janiszewski and A. Felkner, "SpiderTrap—An Innovative Approach to Analyze Activity of Internet Bots on a Website," in IEEE Access, vol. 8, pp. 141292-141309, 2020, doi: 10.1109/ACCESS.2020.3012969
30. S. Song, S. Park and D. Kwon, "metaSafer: A Technique to Detect Heap Metadata Corruption in WebAssembly," in IEEE Access, vol. 11, pp. 124887-124898, 2023, doi: 10.1109/ACCESS.2023.3327817.
31. D. Bermbach and E. Wittern, "Benchmarking Web API Quality - Revisited," in Journal of Web Engineering, vol. 19, no. 5-6, pp. 603-646, September 2020, doi: 10.13052/jwe1540-9589.19563
32. H. Liang et al., "Generative Pre-Trained Transformer-Based Reinforcement Learning for Testing Web Application Firewalls," in IEEE Transactions on Dependable and Secure Computing, vol. 21, no. 1, pp. 309-324, Jan.-Feb. 2024, doi: 10.1109/TDSC.2023.3252523.
33. M. Ahmed et al., "PhishCatcher: Client-Side Defense Against Web Spoofing Attacks Using Machine Learning," in IEEE Access, vol. 11, pp. 61249-61263, 2023, doi: 10.1109/ACCESS.2023.3287226.
34. L. S. Ramamoorthi and D. Sarkar, "Single Sign-On: A Solution Approach to Address Inefficiencies During Sign-Out Process," in IEEE Access, vol. 8, pp. 195675-195691, 2020, doi: 10.1109/ACCESS.2020.3033570
35. A.Praseed and P. S. Thilagam, "Multiplexed Asymmetric Attacks: Next-Generation DDoS on HTTP/2 Servers," in IEEE Transactions on Information Forensics and Security, vol. 15, pp. 1790-1800, 2020, doi: 10.1109/TIFS.2019.2950121.
36. S. -H. Park et al., "Performance Evaluation of Open-Source Endpoint Detection and Response Combining Google Rapid Response and Osquery for Threat Detection," in IEEE Access, vol. 10, pp. 20259-20269, 2022, doi: 10.1109/ACCESS.2022.3152574.
37. M. B. Muzammil, M. Bilal, S. Ajmal, S. C. Shongwe and Y. Y. Ghadi, "Unveiling Vulnerabilities of Web Attacks Considering Man in the Middle Attack and Session Hijacking," in IEEE Access, vol. 12, pp. 6365-6375, 2024, doi: 10.1109/ACCESS.2024.3350444.
38. J. Li, Y. Fu, J. Xu, C. Ren, X. Xiang and J. Guo, "Web Application Attack Detection Based on Attention and Gated Convolution Networks," in IEEE Access, vol. 8, pp. 20717-20724, 2020, doi: 10.1109/ACCESS.2019.2955674.
39. K. Abdulghaffar, N. Elmrabit, and M. Yousefi, “Enhancing Web Application Security through Automated Penetration Testing with Multiple Vulnerability Scanners,” Computers, vol. 12, no. 11, p. 235, Nov. 2023, doi: <https://doi.org/10.3390/computers12110235>.
40. D. Y. Demirel and M. T. Sandikkaya, "Web Based Anomaly Detection Using Zero-Shot Learning With CNN," in IEEE Access, vol. 11, pp. 91511-91525, 2023, doi: 10.1109/ACCESS.2023.3303845.
41. W. Yang, X. Wang, Z. Guan, L. Wu, X. Du and M. Guizani, "SecureSL: A Privacy-Preserving Vertical Cooperative Learning Scheme for Web 3.0," in IEEE Transactions on Network Science and Engineering, vol. 11, no. 5, pp. 3983-3994, Sept.-Oct. 2024, doi: 10.1109/TNSE.2023.3332760.
42. W. Song, Q. Huang and J. Huang, "Understanding JavaScript Vulnerabilities in Large Real-World Android Applications," in IEEE Transactions on Dependable and Secure Computing, vol. 17, no. 5, pp. 1063-1078, 1 Sept.-Oct. 2020, doi: 10.1109/TDSC.2018.2845851.
43. H. Wu, X. Zhang and J. Yang, "Deep Learning-Based Encrypted Network Traffic Classification and Resource Allocation in SDN," in Journal of Web Engineering, vol. 20, no. 8, pp. 2319-2334, November 2021, doi: 10.13052/jwe1540-9589.2085.
44. H. Turtiainen, A. Costin, S. Khandker and T. Hämäläinen, "GDL90fuzz: Fuzzing - GDL-90 Data Interface Specification Within Aviation Software and Avionics Devices–A Cybersecurity Pentesting Perspective," in IEEE Access, vol. 10, pp. 21554-21562, 2022, doi: 10.1109/ACCESS.2022.3150840.
45. S. Khandker, H. Turtiainen, A. Costin and T. Hämäläinen, "Cybersecurity Attacks on Software Logic and Error Handling Within ADS-B Implementations: Systematic Testing of Resilience and Countermeasures," in IEEE Transactions on Aerospace and Electronic Systems, vol. 58, no. 4, pp. 2702-2719, Aug. 2022, doi: 10.1109/TAES.2021.3139559.
46. Yurtseven and S. Bagriyanik, "A Review of Penetration Testing and Vulnerability Assessment in Cloud Environment," 2020 Turkish National Software Engineering Symposium (UYMS), Istanbul, Turkey, 2020, pp. 1-6, doi: 10.1109/UYMS50627.2020.9247071.
47. L. Gebauer, H. Trsek and G. Lukas, "Evil SteVe: An Approach to Simplify Penetration Testing of OCPP Charge Points," 2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA), Stuttgart, Germany, 2022, pp. 1-4, doi: 10.1109/ETFA52439.2022.9921430.
48. R. Khamdamovich and I. Aziz, "Techniques and methods of BLACK BOX identifying vulnerabilities in web servers," 2021 International Conference on Information Science and Communications Technologies (ICISCT), Tashkent, Uzbekistan, 2021, pp. 1-4, doi: 10.1109/ICISCT52966.2021.9670263.
49. B. Kitchenham et al., “Systematic literature reviews in software engineering – A tertiary study,” Information and Software Technology, vol. 52, no. 8, pp. 792–805, Aug. 2010, doi: <https://doi.org/10.1016/j.infsof.2010.03.006>.
50. V. Prokhorenko, K.-K. R. Choo, and H. Ashman, “Web application protection techniques: A taxonomy,” Journal of Network and Computer Applications, vol. 60, pp. 95–112, Jan. 2016, doi: <https://doi.org/10.1016/j.jnca.2015.11.017>.

**APPENDIX A – Sample Code**

import os

import subprocess

import re

import httpx

import asyncio

import datetime

import logging

import pandas as pd

import random

# Setup logging

logging.basicConfig(filename="recon\_framework.log", level=logging.INFO, format="%(asctime)s - %(levelname)s - %(message)s")

# Generate timestamp

now = datetime.datetime.now()

timestamp = now.strftime(".%Y.%m.%d-%H:%M:%S")

# User-specified domain

temp = input("Enter the domain you want to search for: ").strip()

domain = temp + timestamp

# Directory setup

domain\_dir = os.path.join(os.getcwd(), domain)

subdomains\_dir = os.path.join(domain\_dir, 'subdomains')

emails\_dir = os.path.join(domain\_dir, 'emails')

screenshots\_dir = os.path.join(domain\_dir, 'screenshots')

network\_dir = os.path.join(domain\_dir, 'network')

vulnerabilities\_dir = os.path.join(domain\_dir, 'vulnerabilities')

social\_eng\_dir = os.path.join(domain\_dir, 'social\_engineering')

os.makedirs(subdomains\_dir, exist\_ok=True)

os.makedirs(emails\_dir, exist\_ok=True)

os.makedirs(screenshots\_dir, exist\_ok=True)

os.makedirs(network\_dir, exist\_ok=True)

os.makedirs(vulnerabilities\_dir, exist\_ok=True)

os.makedirs(social\_eng\_dir, exist\_ok=True)

# File paths

subdomains9\_path = os.path.join(subdomains\_dir, 'subdomains9.txt')

finalsubdomains\_path = os.path.join(subdomains\_dir, 'finalsubdomains.txt')

finalscan\_path = os.path.join(subdomains\_dir, 'scan.txt')

email\_results\_path = os.path.join(emails\_dir, 'emails.txt')

technologies\_path = os.path.join(subdomains\_dir, 'technologies.txt')

api\_endpoints\_path = os.path.join(subdomains\_dir, 'api\_endpoints.txt')

directory\_bruteforce\_path = os.path.join(subdomains\_dir, 'directories.txt')

ssl\_analysis\_path = os.path.join(subdomains\_dir, 'ssl\_analysis.txt')

network\_services\_path = os.path.join(network\_dir, 'network\_services.txt')

vulnerability\_report\_path = os.path.join(vulnerabilities\_dir, 'vulnerabilities.txt')

social\_engagement\_log = os.path.join(social\_eng\_dir, 'social\_engagement\_log.txt')

# Clear output files

for path in [subdomains9\_path, finalsubdomains\_path, finalscan\_path, email\_results\_path, technologies\_path, api\_endpoints\_path, directory\_bruteforce\_path, ssl\_analysis\_path, network\_services\_path, vulnerability\_report\_path, social\_engagement\_log]:

with open(path, 'w'):

pass

# 1. Subdomain Enumeration

def run\_subdomain\_enumeration(timeout=30):

commands = [

f"assetfinder -subs-only {temp} >> {subdomains9\_path}",

f"subfinder -silent -d {temp} --wildcard | grep -Eo '[.a-zA-Z0-9-]+\\s\\.{temp}' >> {subdomains9\_path}",

f"curl -s 'https://otx.alienvault.com/api/v1/indicators/domain/{temp}/url\_list?limit=1500&page=1' | grep -o '\"hostname\": \"[^\"]' | sed 's/\"hostname\": \"//' | sort -u >> {subdomains9\_path}"

]

for command in commands:

try:

subprocess.run(command, shell=True, check=True)

logging.info(f"Executed command: {command}")

except subprocess.CalledProcessError as e:

logging.error(f"Command failed: {command}, error: {e}")

# Remove duplicates and filter wildcard subdomains

with open(subdomains9\_path, 'r') as infile, open(finalsubdomains\_path, 'w') as outfile:

unique\_lines = sorted(set(line.strip() for line in infile if not line.startswith('\*')))

outfile.write('\n'.join(unique\_lines))

os.remove(subdomains9\_path)

# 2. DNS Zone Transfer Check

def check\_dns\_zone\_transfer(timeout=30):

dns\_transfer\_check = f"dnsrecon -d {temp} -t axfr"

result = subprocess.run(dns\_transfer\_check, shell=True, capture\_output=True, text=True)

if result.stdout:

with open(os.path.join(subdomains\_dir, 'dns\_transfer.txt'), 'w') as f:

f.write(result.stdout)

logging.info("DNS Zone Transfer possible. Data saved.")

else:

logging.info("No DNS Zone Transfer data found.")

# 3. Directory Brute-Forcing

def directory\_bruteforce(timeout=30):

for protocol in ['http', 'https']:

gobuster\_command = f"gobuster dir -u {protocol}://{temp} -w /home/oxo/Desktop/SecLists/Discovery/Web-Content/raft-small-words.txt -o {directory\_bruteforce\_path}"

subprocess.run(gobuster\_command, shell=True)

logging.info(f"Directory brute-force complete for {protocol}. Results saved to {directory\_bruteforce\_path}")

# 4. SSL/TLS Certificate Analysis

def ssl\_certificate\_analysis(timeout=30):

for port in [80, 443]:

# Construct the command to use testssl.sh with JSON output for the given port

testssl\_command = f"bash ~/Downloads/testssl.sh-3.2/testssl.sh --json {temp}:{port}"

# Run the command

result = subprocess.run(testssl\_command, shell=True, capture\_output=True, text=True)

# Check if the result is non-empty

if result.stdout:

# Save the output to the ssl\_analysis\_path file

with open(f"{ssl\_analysis\_path}\_{port}.json", 'w') as f:

f.write(result.stdout)

logging.info(f"SSL certificate analysis for port {port} complete. Data saved.")

else:

logging.info(f"No SSL certificate data found for port {port}.")

# 5. Network Service Enumeration

def network\_service\_enumeration(timeout=30):

# Use the domain name directly (you can use temp or another variable holding the domain)

jfscan\_command = f"jfscan --yummy-ports -q {temp} -o {network\_services\_path}"

# Run the command

result = subprocess.run(jfscan\_command, shell=True, capture\_output=True, text=True)

# Check if there's output from the jfscan command

if result.stdout:

with open(network\_services\_path, 'w') as output\_file:

output\_file.write(result.stdout)

logging.info("Network service enumeration complete. Results saved to network\_services.txt.")

else:

logging.info("No results found for the domain.")

# 6. Technology Fingerprinting

def technology\_fingerprint(timeout=30):

for protocol in ['http', 'https']:

# Use shell redirection to write output to the technologies file

tech\_command = f"whatweb {protocol}://{temp} > {technologies\_path}" if protocol == 'http' else f"whatweb {protocol}://{temp} >> {technologies\_path}"

subprocess.run(tech\_command, shell=True)

logging.info(f"Technology fingerprinting complete for {protocol}.")

# 7. API Endpoint Discovery

def api\_endpoint\_discovery(timeout=30):

for protocol in ['http', 'https']:

# Command for ffuf to find directories or endpoints with status 200 responses

ffuf\_command = f"ffuf -w ~/Desktop/SecLists/Fuzzing/fuzz-Bo0oM-friendly.txt -u {protocol}://{temp}/FUZZ -mc 200

# Execute the command and append output to the result file

try:

with open(api\_endpoints\_path, 'a') as outfile:

subprocess.run(ffuf\_command, shell=True, check=True, stdout=outfile, stderr=outfile)

logging.info(f"API endpoint discovery (fuzzing) complete for {protocol}. Results saved to {api\_endpoints\_path}")

except subprocess.CalledProcessError as e:

logging.error(f"ffuf command failed for {protocol}: {e}")

# Check the results and print them

try:

with open(api\_endpoints\_path, 'r') as result\_file:

results = result\_file.read()

if results:

print(f"API endpoint discovery results:\n{results}")

else:

print("No API endpoints found or ffuf command did not execute successfully.")

except Exception as e:

logging.error(f"Error reading result file: {e}")

print(f"Error reading result file: {e}")

# 9. Vulnerability Scanning with Multiple Scanners

def vulnerability\_scanning(timeout=30):

# List of scanners to use for both http and https

scanners = [

f"nikto -h http://{temp} >>{vulnerability\_report\_path} -C all",

f"nikto -h https://{temp} >> {vulnerability\_report\_path} -C all",

]

for scanner in scanners:

try:

subprocess.run(scanner, shell=True, timeout=20)

except subprocess.TimeoutExpired:

logging.warning(f"Timeout expired for scanner: {scanner}")

# Log the completion of the vulnerability scan

logging.info("Vulnerability scanning complete.")

# 10. Capture Screenshots of Discovered Subdomains

# Convert subdomains to URLs for Eyewitness

def prepare\_urls\_for\_screenshots(subdomains\_file, urls\_file):

with open(subdomains\_file, 'r') as sub\_file, open(urls\_file, 'w') as url\_file:

for subdomain in sub\_file:

subdomain = subdomain.strip()

url\_file.write(f"http://{subdomain}\n")

url\_file.write(f"https://{subdomain}\n")

logging.info("URLs prepared for Eyewitness.")

# Capture Screenshots of Discovered Subdomains

def capture\_screenshots(timeout=30):

# Prepare URLs from the subdomains list

prepare\_urls\_for\_screenshots(finalsubdomains\_path, 'urls\_for\_eyewitness.txt')

# Run Eyewitness with the prepared URLs

eyewitness\_command = f"eyewitness --no-prompt -f urls\_for\_eyewitness.txt -d {screenshots\_dir} --timeout 30"

subprocess.run(eyewitness\_command, shell=True)

logging.info("Screenshot capture with Eyewitness complete.")

# 11. Email Extraction using theHarvester and regex

import re

import requests

from bs4 import BeautifulSoup

from selenium import webdriver

from selenium.webdriver.chrome.options import Options

from selenium.webdriver.chrome.service import Service # Correct import for Service

from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

# Path to chromedriver

driver\_path = '/usr/bin/chromedriver'

def extract\_emails\_from\_text(text):

# Regular expression pattern for matching email addresses

email\_pattern = r'[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}'

return re.findall(email\_pattern, text)

def extract\_emails\_from\_webpage(url, use\_selenium=False):

emails = set()

try:

if use\_selenium:

# Setup Selenium options

options = Options()

options.headless = True # Run in headless mode

service = Service(driver\_path) # Create Service object with the path to chromedriver

driver = webdriver.Chrome(service=service, options=options)

driver.set\_page\_load\_timeout(40)

driver.get(url)

# Wait for the page to fully load (adjust as needed for specific page elements)

WebDriverWait(driver, 20).until(

EC.presence\_of\_element\_located((By.TAG\_NAME, 'body'))

)

page\_content = driver.page\_source

driver.quit()

else:

# For non-Selenium requests, handle with requests

response = requests.get(url, timeout=10)

response.raise\_for\_status() # Raise error for bad status codes

page\_content = response.text

# Parse the webpage content with BeautifulSoup

soup = BeautifulSoup(page\_content, 'html.parser')

# Extract emails from visible text

text\_emails = extract\_emails\_from\_text(soup.get\_text())

emails.update(text\_emails)

# Extract emails from common HTML attributes

for attr in ['href', 'src', 'data', 'content']:

for tag in soup.find\_all(attrs={attr: True}):

emails.update(extract\_emails\_from\_text(tag[attr]))

# Print extracted emails

if emails:

print(f"Emails found on {url}:")

for email in emails:

print(email)

else:

print(f"No emails found on {url}")

except requests.RequestException as e:

pass

except Exception as e:

pass

return emails

# Extract emails from both http and https versions of a domain

def extract\_emails(timeout=30):

protocols = ['http', 'https']

all\_emails = set()

domain=temp

for protocol in protocols:

url = f"{protocol}://{domain}"

emails = extract\_emails\_from\_webpage(url, use\_selenium=True)

all\_emails.update(emails)

# Save results to file

with open(email\_results\_path, 'w') as f:

for email in all\_emails:

f.write(email + '\n')

logging.info("Email extraction complete.")

# 12. Social Engineering Component: Simulate Phishing Emails

def simulate\_phishing\_emails(timeout=30):

try:

# Read extracted emails

with open(email\_results\_path, 'r') as file:

email\_addresses = [line.strip() for line in file if line.strip()]

except FileNotFoundError:

logging.error(f"Email results file not found: {email\_results\_path}")

return

if not email\_addresses:

logging.warning("No email addresses found for phishing simulation.")

return

# Phishing email templates

phishing\_templates = [

{"subject": "Urgent: Password Reset Required",

"message": "Your account password needs to be reset immediately. Click here: {link}."},

{"subject": "Important Security Update",

"message": "A new security patch has been released. Review details here: {link}."},

{"subject": "Action Required: Verify Your Account",

"message": "We noticed unusual activity on your account. Please verify here: {link}."}

]

phishing\_log = []

# Generate phishing emails

for email in email\_addresses:

template = random.choice(phishing\_templates)

phishing\_email = {

"to": email,

"subject": template["subject"],

"message": template["message"].format(link="http://phishing-link.com"),

}

phishing\_log.append(phishing\_email)

# Print simulated email

print(f"To: {phishing\_email['to']}\nSubject: {phishing\_email['subject']}\nMessage: {phishing\_email['message']}\n")

# Write phishing email log

with open(social\_engagement\_log, 'a') as log\_file:

for entry in phishing\_log:

log\_file.write(

f"To: {entry['to']}, Subject: {entry['subject']}, Message: {entry['message']}\n"

)

logging.info(f"{len(phishing\_log)} phishing emails simulated and logged.")

# 13. Run all functions in sequence

def run\_all():

run\_subdomain\_enumeration()

check\_dns\_zone\_transfer()

directory\_bruteforce()

ssl\_certificate\_analysis()

network\_service\_enumeration()

technology\_fingerprint()

api\_endpoint\_discovery()

vulnerability\_scanning()

capture\_screenshots()

extract\_emails()

simulate\_phishing\_emails()

run\_all()