ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πτυχιακή Εργασία

Τίτλος Πτυχιακής Εργασίας	«Εργαλείο Διαχείρισης Ευπαθειών για Unix Συστήματα» «Vulnerability Assessment Tool for Unix Systems»
Ονοματεπώνυμο Φοιτητή	Ηρακλής Δούσης
Πατρώνυμο	Γεώργιος
Αριθμός Μητρώου	Π/ 16181
Επιβλέπων	Δέσποινα Πολέμη, Καθηγήτρια

Ημερομηνία Παράδοσης Σεπτέμβριος 2024

Copyright ©

Table of Contents

Abstract

In the evolving landscape of cybersecurity, the effective identification and assessment of vulnerabilities within IT infrastructures have become paramount. This bachelor thesis presents the development of a Vulnerability Assessment Tool, designed to streamline the process of detecting, categorizing, and prioritizing vulnerabilities in computer systems. While the tool's primary focus is on Unix-based systems, its adaptable framework allows for potential expansion to other operating systems, catering to a diverse range of IT environments and systems.

The tool aims to automate the process of vulnerability scanning, leveraging up-to-date information from established vulnerability databases and common areas of exploitation. It is designed to be user-friendly, enabling IT personnel to efficiently manage vulnerability assessments across multiple systems. By offering features such comprehensive and centralized reporting, real time risk scanning/handling, the tool serves as an asset for organizations seeking to bolster their cybersecurity approach and current procedures.

The core part of this project is the ability to manage and help secure a large-scale IT infrastructure with many systems through a management/web panel. The tool is developed with a keen eye on the requirements of modern IT infrastructures, ensuring compatibility, scalability, and ease of integration.

This thesis not only contributes a practical tool to the field of cybersecurity but also provides an in-depth analysis of the current landscape of vulnerability assessment. Through this work, it aims to enhance the understanding of vulnerability management processes and underscore the importance of proactive security measures in the digital age.

Part I: Theoretical Background and Market Analysis

1. Introduction

1.1 Evolution of vulnerability assessment

- The Birth of Network Security

The origins of vulnerability assessment are linked with the early development of computer networks in the 1960s and 1970s. During these initial stages, the emphasis was on establishing connectivity and functionality, with security being a secondary concern. However, as these networks began to expand beyond academic and military domains into the broader public sector, the need for security mechanisms became evident.

- The Arrival of Automated Tools

The 1980s and 1990s saw the spread of personal computers and the internet, marking the beginning of widespread network vulnerabilities. It was during this period that the concept of automated vulnerability assessment began to take shape. Tools like SATAN (Security Administrator Tool for Analyzing Networks), released in 1995, and its contemporary, Internet Security Scanner (ISS), provided administrators with the means to automate the process of scanning networks for known vulnerabilities. These tools were very basic by today's standards but laid the groundwork for the development of more sophisticated VA technologies.

- Standardization and Regulation

The late 1990s and early 2000s were characterized by efforts to standardize vulnerability assessment processes. The introduction of the Common Vulnerabilities and Exposures (CVE) system in 1999 offered a unified index of known vulnerabilities, helping the sharing of information and the comparison of security tools. This period also witnessed the implementation of regulatory frameworks, such as the Sarbanes-Oxley Act of 2002, which mandated regular security assessments for certain businesses, further embedding vulnerability assessment into the fabric of network security. [1]

- Integration with Software Development

The adoption of Agile and DevOps methodologies in software development during the 2000s and 2010s brought a model shift in vulnerability assessment. The concept of "shifting security left," or integrating security measures early in the development lifecycle, gained traction. Tools like OWASP ZAP and static application security testing (SAST) solutions became integral to the development process, enabling developers to identify and remediate vulnerabilities before deployment.

- The Era of Continuous Assessment

Today, the landscape of vulnerability assessment is dominated by continuous and integrated approaches. The rise of cloud computing and CI/CD (Continuous Integration/Continuous Deployment) pipelines has created the requirement of development of tools capable of continuous monitoring and assessment. Platforms like Qualys and Tenable.io offer cloud-based vulnerability management, allowing organizations to continuously track and address vulnerabilities across their digital infrastructure.

- The Future: AI and Predictive Analytics

The future of vulnerability assessment lies in the integration of artificial intelligence and machine learning technologies. Predictive analytics and AI-driven threat modeling are poised to transform vulnerability assessment, enabling organizations to anticipate potential vulnerabilities and proactively mitigate threats before they are exploited. [2] [3]

1.2 Overview of Unix security challenges

Figure 1 Showing different aspects of cybersecurity [4]

The Unix operating system, known for its reliability and flexibility, supports much of the world's computing infrastructure. However, despite its robust architecture, Unix systems face several security challenges, particularly in areas involving vulnerable packages, services, and outdated software. These vulnerabilities, combined with the complexities of user permissions and network exposure, make Unix systems prime targets for attackers. This chapter explores these challenges, focusing on vulnerable software, services, and critical system misconfigurations that can be exploited if left unchecked.

- Vulnerable Packages, Services, and Software

One of the most pressing challenges facing Unix systems is the presence of vulnerable packages and services. Software components such as outdated libraries, unpatched services, and poorly maintained applications can be exploited by attackers to gain unauthorized access or escalate privileges. For example, widely-used services like OpenSSH, Apache HTTP Server, and MySQL may contain known vulnerabilities if they are not regularly updated. Attackers often exploit these vulnerabilities using automated tools, making it critical for system administrators to keep packages up-to-date the manual way, or by use of an automated vulnerability scanning tool.

Legacy and outdated software present a similar challenge. Many Unix systems, especially those used in enterprise environments, rely on older software components that are no longer maintained or supported. This creates a significant risk, as attackers can exploit these outdated components to bypass security measures. Moreover, when vulnerable software is left unpatched, it provides a clear avenue for attackers to use publicly available exploits to gain control of the system. Therefore, ensuring that all packages are regularly updated and maintaining a strict patch management process are essential for maintaining system security. [5]

- Network Exposure and Services

Unix systems are often deployed in networked environments where they serve as the backbone for web servers, databases, and communication services. This network exposure makes Unix systems particularly vulnerable to network-based attacks. For instance, services such as SSH (Secure Shell), FTP (File Transfer Protocol), and SMTP (Simple Mail Transfer Protocol) can become attack vectors if they are misconfigured or outdated. Unpatched SSH services are often vulnerable to brute force attacks, while FTP services with improper configuration can allow unauthorized access to the system's file structure.

Public-facing services, especially those running on open ports, present significant risk. If these services are not secured or outdated, attackers can exploit them using known vulnerabilities or configuration flaws. Ensuring that only essential services are running, combined with proper firewall configuration and network segmentation, is crucial for minimizing the attack surface. [6]

- User and Group Permission Misconfigurations

Unix systems feature a detailed permissions model that manages access based on user, group, and world categories. While this provides detailed control over system resources, misconfigurations can introduce serious vulnerabilities. For example, files with improper permissions may allow unauthorized users to read, write, or execute sensitive files. Files with the setuid or setgid permission bits set allow programs to run with the privileges of the file owner or group, and if such files are exploitable, they can lead to privilege escalation attacks. [7]

Similarly, world-writable files can pose significant risks, as they allow any user to modify critical system files or scripts, potentially leading to privilege escalation or denial of service attacks. Ensuring that file permissions are regularly audited and properly configured is essential to reducing the risk of unauthorized access or privilege abuse. [8]

- Scripting and Command Injection Vulnerabilities

Unix systems often rely on shell scripts and command-line utilities to automate system tasks. However, this flexibility introduces risks related to command injection and scripting vulnerabilities. Insecure scripts that do not properly sanitize input can allow attackers to inject malicious commands, which are then executed by the system with elevated privileges. This can result in the unauthorized execution of commands, data leakage, or even full system compromise.

Applications that interface with shell commands, such as web applications or network services, are particularly vulnerable to command injection attacks. Proper input validation and the use of secure coding practices are critical to mitigating these risks. Additionally, system administrators should regularly audit scripts and utilities to ensure they do not introduce unnecessary security risks. [9]

- Misconfigured Sudoers File

The sudoers file, which controls the delegation of administrative privileges in Unix systems, is another common source of vulnerabilities. If misconfigured, the sudoers file can grant excessive privileges to users or allow users to run dangerous commands without proper authorization. For example, allowing a user to execute any command as root without requiring a password significantly weakens system security.

Ensuring that the sudoers file is correctly configured—granting minimal necessary privileges and restricting command execution to only trusted users—is essential. Regular audits of the sudoers file, combined with the use of tools such as sudo -l to verify user privileges, can help prevent unauthorized privilege escalation. [10]

- Firewall Misconfigurations and Network Services

Misconfigured firewalls represent another area where Unix systems can be exposed to external threats. If firewall rules are overly permissive or incorrectly configured, attackers may be able to bypass network protections and exploit vulnerabilities in publicly exposed services. For instance, improperly secured firewalls can leave sensitive services, such as databases or administrative interfaces, open to external access. Regularly reviewing firewall configurations, using tools like UFW or iptables, and ensuring that only necessary ports are open can mitigate these risks. [11]

Addressing Unix Security Challenges

To address these security challenges, it is important to use a combination of regular vulnerability assessments, good configuration practices, and following security best practices. Toolscan be used to scan Unix systems for known vulnerabilities, outdated software, and misconfigurations in services and file permissions. By dealing with these common vulnerabilities and prioritizing security, Unix administrators can lower the risk of system compromise and protect their systems from both internal and external threats.

2 Principles of vulnerability assessment

Vulnerability assessment is a critical process in managing the security of computer systems, particularly in Unix environments. It involves several key principles aimed at identifying, evaluating, and addressing potential vulnerabilities. While the process can be complex, understanding its core principles is not.

- Inventory and Understanding the Assets

The first step in vulnerability assessment is to have a clear picture of what assets are available. This means identifying all the components of the system, from the software applications to the physical devices. It's akin to knowing every entry point to a building before you can effectively secure it.

- Vulnerability Identification

Once we know what assets are available, the next step is to scan these assets for known vulnerabilities. This is where specialized tools come into play, scanning the systems for weaknesses just like a health check-up can reveal areas of concern that need attention.

- Risk Analysis

Not all vulnerabilities pose the same level of risk. Some may expose the systems to significant danger, while others might be less critical. Risk Analysis involves evaluating the potential impact of each vulnerability, helping prioritize which issues to address first based on their potential to harm the system or data.

- Remediation Planning

Identifying and analyzing vulnerabilities is part of the process. The next critical step is planning how to fix these issues. This might involve updating software, changing configurations, or applying security patches. The goal is to mitigate risks without disrupting system operations more than necessary.

- Continuous Monitoring and Reassessment

The digital landscape is constantly evolving, with new vulnerabilities emerging regularly. Continuous monitoring of the systems for new risks is crucial. This ensures that newly discovered vulnerabilities can be addressed promptly, keeping systems secure over time.

- Documentation and Reporting

Keeping detailed records of vulnerability assessments, including what vulnerabilities were found, how they were prioritized, and the steps taken to address them, is essential. This documentation is vital for tracking progress over time, proving compliance with security standards (i.e. ISO27001), and guiding future security decisions. [12]

3. Review of Existing Vulnerability Assessment Tools

In this section, some of the most commonly used vulnerability assessment tools for Unix systems are reviewed. These tools help organizations identify and address potential security issues, ranging from software vulnerabilities to misconfigurations. The following tools will be examined:

- OpenVAS (Open Vulnerability Assessment System)
- Nessus
- Nmap (Network Mapper)
- QualysGuard

3.1 Evaluation of in-place market solutions

- OpenVAS:

OpenVAS is an open-source vulnerability scanner primarily focused on network-based scanning. It offers a vast range of tests that cover various protocols and services, helping administrators detect vulnerabilities in Unix systems and beyond.

Use Case: It is ideal for enterprises looking for a comprehensive open-source tool for network scanning and system vulnerabilities. Its strength lies in network protocols and identifying misconfigurations in services like FTP, SSH, and more. [13]

- Nessus:

Nessus, developed by Tenable, is a widely used commercial vulnerability scanner. It offers extensive plugin support for scanning vulnerabilities across different system types, including Unix-based environments. The tool is also known for its ease of use and detailed reporting features.

Use Case: Nessus is particularly effective for large environments where detailed vulnerability scanning and in-depth reporting are essential. It scans operating systems, applications, and network services, identifying both common and obscure vulnerabilities. [14]

- Nmap:

Nmap, while primarily a network scanner, has grown to include a range of vulnerability detection capabilities through its powerful scripting engine (NSE). It's widely used for port scanning, network discovery, and detecting vulnerabilities in services that run on open ports.

Use Case: Nmap is often used in the initial stages of vulnerability assessments to detect open ports and discover running services. It's an invaluable tool for system administrators conducting network-level assessments. [15]

- QualysGuard:

QualysGuard is a cloud-based vulnerability management solution, allowing continuous vulnerability assessments through a web interface. It offers automated, cloud-based scanning for a wide range of security issues, from system vulnerabilities to compliance checks.

Use Case: Best suited for organizations that need a scalable and automated solution for continuous vulnerability monitoring across Unix and other environments. It is particularly valuable for organizations with compliance needs, such as GDPR or PCI DSS.

3.2 Strengths and weaknesses of current tools

Tool	Strengths	Weaknesses
Nmap	• Flexible and fast for network discovery and port scanning. • Extensive scripting engine (NSE) for custom vulnerability checks.	• Not a full-featured vulnerability assessment tool (focused more on network scanning). • Requires advanced scripting knowledge for more complex vulnerability assessments.
OpenVAS	• Open-source and regularly updated. • Comprehensive network and service vulnerability scanning.	• Resource-intensive; complex setup. • Not as user-friendly as commercial tools.
Nessus	• Easy to use with a large plugin library for detecting a wide range of vulnerabilities • Detailed reporting and integration with vulnerability management tools.	• Expensive for full-featured versions. • Free version (Nessus Essentials) has limited features.
QualysGuard	• Cloud-based, easy to deploy and manage without requiring on-premise infrastructure.	• Subscription-based, potentially costly for smaller organizations.

	• Continuous, automated scanning and compliance reporting.	• Relies on continuous internet connectivity

3.3 Current Trends and Practices in Vulnerability Assessment

One of the most significant trends in vulnerability assessment is the shift from periodic scanning to continuous, automated assessments. With cyberattacks becoming more sophisticated and frequent, organizations can no longer rely on occasional vulnerability scans. Automated tools continuously scan systems, identifying vulnerabilities in real-time and alerting administrators to critical issues as they arise. Continuous vulnerability assessment enables proactive detection of new threats, allowing for rapid remediation before vulnerabilities can be exploited.

Moreover, automation has made it easier to scale vulnerability assessments across large, complex networks, ensuring that all assets—including those in cloud and hybrid environments—are monitored effectively. [16] [17]

3.4 Latest developments and the use of AI in vulnerability assessment

In recent years, the field of vulnerability assessment has undergone significant transformations, driven primarily by advancements in artificial intelligence (AI) and machine learning (ML). These technologies have enabled faster and more accurate identification, prioritization, and remediation of vulnerabilities in increasingly complex IT environments. Below are some of the latest developments and the role AI plays in enhancing vulnerability assessment:

- AI-Driven Vulnerability Detection

AI has improved the process of detecting vulnerabilities by automating and enhancing traditional methods. AI algorithms analyze large amounts of data in real-time to detect patterns that signify potential vulnerabilities. For example, tools like Google's Sec-PaLM use generative AI to automatically scan and analyze code, providing detailed insights into vulnerabilities that may have been missed by manual inspections. [18] AI can also detect zero-day vulnerabilities by identifying new patterns that don't match known signatures, a capability that traditional tools struggle with.

This AI-driven approach enhances the speed and accuracy of vulnerability detection, reducing the chances of human error and helping organizations react quickly to new threats [19]

- AI-Powered Remediation

Once vulnerabilities are detected, AI also plays a crucial role in guiding the remediation process. AI-based tools provide step-by-step remediation suggestions tailored to the specific environment, helping security teams patch vulnerabilities more efficiently. For instance, solutions like Tenable's AI capabilities automate risk prioritization and offer context-aware remediation, allowing teams to focus on fixing the most critical issues first, rather than being overwhelmed by a long list of vulnerabilities. [20]

Part II: Development and Analysis of simpleVAP

4. System Architecture and Design

4.1 Design Principles and Requirements

The design of simpleVAP (Simple Vulnerability Assessment Platform) is based on core principles that ensure scalability, security, and user-friendliness. Given the critical role of vulnerability assessments in modern Unix environments, simpleVAP was designed to meet the needs of system administrators by offering automated, scalable, and reliable vulnerability detection, coupled with a flexible user interface and management system. The following principles and requirements formed the development of simpleVAP:

1. Security by Design

Security is very important in simpleVAP. Given that the tool deals with sensitive data such as system configurations, CVEs, user credentials, and machine information, it was crucial to design the platform with security-first principles.

Encrypted Communication: All communication between the central server and systems is handled via secure protocols such as SSH (using paramiko for Python [21]) to ensure confidentiality and integrity.

Token-Based Authentication: Each machine added to the platform is assigned a unique token, which ensures that only authorized systems can communicate with the central server (securely generated with python secrets library [22]).

Data Encryption: Sensitive data, including system credentials and tokens, are securely stored in an encrypted format within the SQLite database to prevent unauthorized access.

2. User-Focused Interface

The platform's front-end design, developed using Flask [23] with Bootstrap 5.3 [24], was built to provide an intuitive and user-friendly interface for system administrators. Some key design choices include:

Simple Navigation: The user management system and dashboard allow administrators to easily add systems, view machine statuses, and monitor vulnerabilities.

Customizable Scan Options: Administrators can perform vulnerability scans on specific systems or all systems at once, with the ability to schedule scans or run them manually based on real-time needs.

Detailed Reporting: Scan results are presented clearly, with high-priority vulnerabilities prominently displayed to help prioritize remediation efforts.

3. Scalability

simpleVAP was designed with scalability in mind, enabling administrators to manage a large pool of machines without experiencing performance bottlenecks.

Cron-Based Status Updates: Each machine regularly "pings" the server using a cron job, reporting its current status and internal IP address. This method of decentralized updates ensures that the central server isn’t overloaded by frequent communication from multiple machines.

Efficient Storage with Pickled Data: To optimize lookup times for vulnerability information, simpleVAP stores the NVD/NIST database in a serialized format (Pickle), allowing for faster queries when matching packages to CPEs (Common Platform Enumerations).

4. Extensibility

simpleVAP was built with flexibility in mind, allowing for future feature expansion and integrations. For example:

Modular Tools: The platform includes built-in tools like Nmap, Nikto, and SQLMap, which can be run against selected systems. These tools are modular, so additional tools or scanning engines can be easily integrated into the platform in the future.

Custom Scan Scripts: The two main scanning scripts (checks.sh and remote_script.py) can be modified or expanded as new vulnerabilities or security concerns arise, ensuring that the system remains up-to-date with evolving security standards.

6. Automation and Efficiency

Automated Dependency Installation: The platform's checks.sh script automatically verifies whether Python and necessary libraries are installed on target machines, reducing the burden on system administrators to prepare each machine for vulnerability scans.

- Core Requirements

1. Usability

The platform must be easy to use, even for administrators who may not be familiar with advanced vulnerability scanning tools. As such, all features—including adding systems, initiating scans, and reviewing results—are accessed through a clean web interface.

2. Performance

Performance was another key requirement. The platform must be able to handle multiple scans simultaneously without excessive latency or server load. To achieve this, simpleVAP efficiently manages communication with systems using SSH, and threading, and relies on asynchronous methods to perform vulnerability lookups against the NVD/NIST database.

3. Accuracy

The platform must provide accurate vulnerability assessments by using the latest security data from the NVD/NIST database. This is achieved through regular updates to the CPE dictionary and CVE matching algorithms, ensuring that system administrators receive up-to-date information on vulnerabilities affecting their systems.

4.2 Methods of identifying Vulnerabilities

Identifying vulnerabilities is a core function that helps system administrators maintain a secure and updated environment. The process of identifying vulnerabilities in Unix systems can be broken down into two main methods: detecting known vulnerabilities through Common Vulnerabilities and Exposures (CVEs) and identifying security risks related to system misconfigurations or other potential threats.

4.2.1 CVEs

CVEs are standardized identifiers for known security vulnerabilities [25] in software and systems. Each CVE is assigned a unique identifier, followed by detailed information on the nature of the vulnerability, including its impact and how it can be exploited (CVSS – Common Vulnerability Scoring System). The CVE system is maintained by the MITRE Corporation and is used worldwide as a reference for identifying and managing security issues. CVEs are a fundamental aspect of vulnerability management in simpleVAP, as they provide a reliable and standardized way to track vulnerabilities in software packages and services.

In simpleVAP, after a vulnerability scan is performed on an systen, the gathered package names and versions are converted into CPE (Common Platform Enumeration) format, which allows for accurate matching with known CVEs. The tool then queries the National Vulnerability Database (NVD) using the CPE identifiers to retrieve relevant CVEs associated with each package [26].

This automated process enables administrators to quickly identify if their systems are vulnerable to any known security issues.

For example, if a system is running an outdated version of OpenSSH, simpleVAP will identify the associated CVE (e.g., CVE-2020-15778 [27]) and provide the details of the vulnerability, including the severity and potential impact. The platform then compiles this information into a report, allowing system administrators to prioritize remediation efforts based on the severity of the CVE, using its CVSS.

4.2.2 Misconfigurations and various other threats

Beyond identifying vulnerabilities through CVEs, simpleVAP also performs checks for misconfigurations and other potential threats that may expose a system to risk. Misconfigurations are a common cause of security breaches, as they often leave systems unintentionally exposed to attackers. simpleVAP uses a combination of shell scripts and Python to detect several types of misconfigurations, focusing on key areas like file permissions, running services, network settings, and software configurations.

1. File and Directory Permissions:

Unix systems use a detailed permissions model to manage access to files and directories. However, improper permissions can lead to privilege escalation or unauthorized access to sensitive files. For example, files with world-writable permissions (777) can be modified by any user on the system, which is a significant security risk. Similarly, files with the setuid or setgid bit set can be exploited to execute commands with elevated privileges. simpleVAP scans for these insecure configurations and highlights them in the report, allowing administrators to correct them promptly [28].

2. SUID/SGID Files:

Files with SUID (Set User ID) or SGID (Set Group ID) permissions allow users to execute files with the privileges of the file's owner or group. If these files are not properly secured, attackers can exploit them to gain unauthorized access or escalate privileges. simpleVAP checks for any SUID or SGID files and flags them if they pose a risk to the system. [29]

3. Outdated Software and Packages:

Another area of concern is outdated software packages. Many vulnerabilities exist simply because software is not kept up to date with the latest security patches. For instance, old versions of OpenSSH or Apache may have known vulnerabilities that can be easily exploited [30]. simpleVAP compares installed package versions against the latest known versions and flags any that are outdated, encouraging administrators to apply the necessary updates. [31] [32]

4. Network Services and Open Ports:

simpleVAP also checks for unnecessary or vulnerable services running on open ports. Open ports expose services to external access, and if these services are not properly secured, attackers can exploit them to gain unauthorized entry to the system. For instance, services like FTP and Telnet are known for their lack of encryption and should be replaced with more secure alternatives like SFTP or SSH [33]. simpleVAP scans the network configuration and reports on potentially insecure services running on open ports.

5. Insecure SSH Configurations:

SSH is a common protocol used for remote access to Unix systems, but if it is misconfigured, it can be an easy target for attackers. For example, allowing root login via SSH or using weak encryption algorithms can expose a system to brute force or man-in-the-middle attacks. simpleVAP checks the SSH configuration for insecure settings, such as allowing PermitRootLogin or using deprecated ciphers, and recommends changes to improve security. [34] [35]

6. World-Writable Directories:

World-writable directories allow any user to create or modify files within them, which could lead to malicious scripts or files being executed with elevated privileges. simpleVAP identifies such directories and flags them as high-risk areas, enabling system administrators to tighten permissions where necessary. [36]

7. Misconfigured Firewalls:

Firewalls play a critical role in protecting Unix systems from external threats. However, a misconfigured firewall can leave sensitive services exposed to external attacks. simpleVAP reviews firewall configurations and checks for overly permissive rules that may allow unauthorized access to important services like databases, administrative interfaces, or web servers. [37] [38]

4.3 Overview of the tools architecture

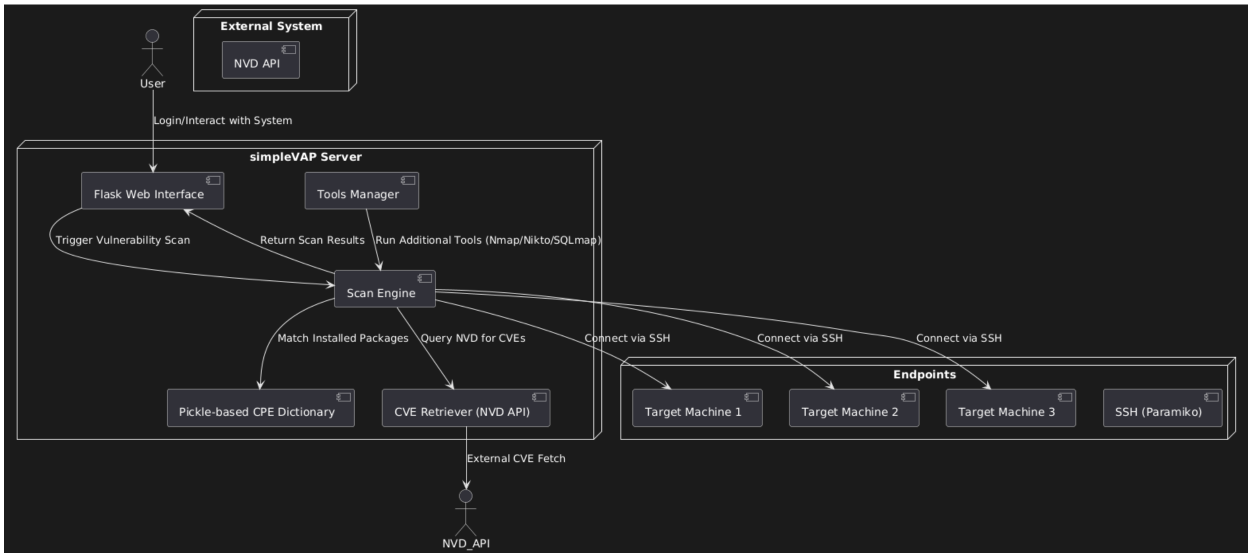


Figure 2 Deployment Diagram

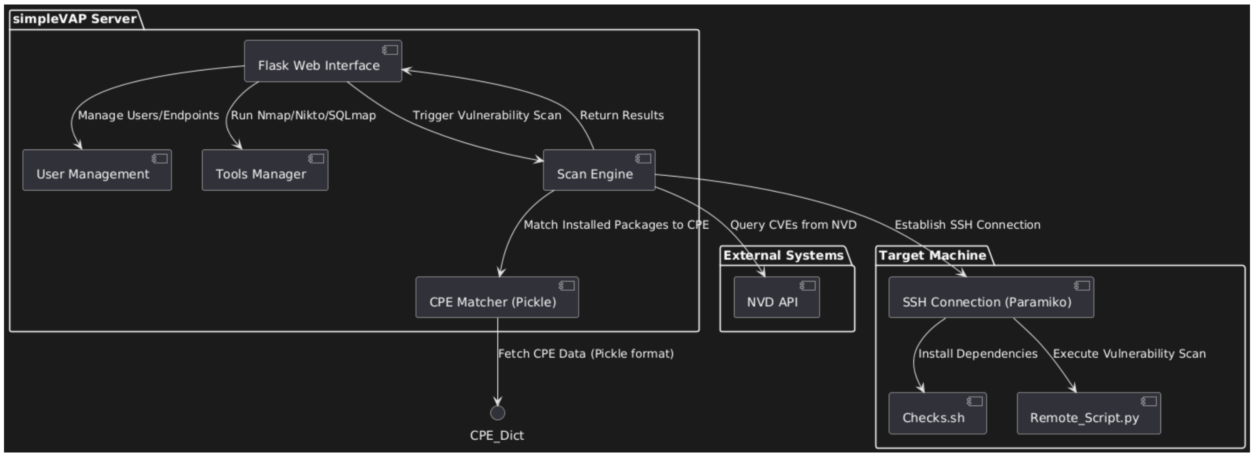


Figure 3 Component Diagram

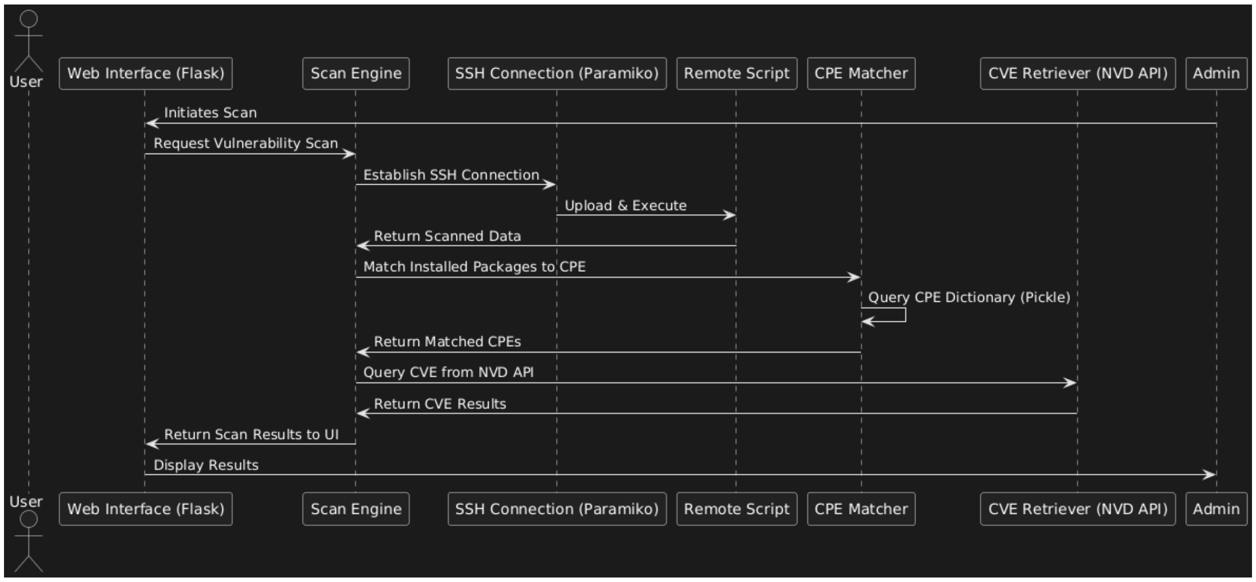


Figure 4 Sequence Diagram

4.4 Services of simpleVAP

The core services of simpleVAP provide system administrators with a wide array of tools to monitor, scan, and assess the security posture of Unix-based systems. These services, accessible through a web-based interface, allow administrators to manage systems, execute scans, retrieve reports, and perform additional security tasks. The platform's modularity allows for future extensibility through a plugin system where users can add their custom scripts to the scanning process.

4.4.1 WebPanel

The WebPanel is the main interface for interacting with simpleVAP, built using Flask as the backend and Bootstrap 5.3 for the front-end design. It provides an intuitive user experience with simple navigation, allowing system administrators to easily perform the following tasks:

- User Authentication and Management: Users log in with credentials, and the platform supports multiple administrator accounts. Role-based access can be implemented to assign permissions to different users based on their responsibilities.

- Dashboard: The dashboard provides a quick overview of all the systems, their statuses, and the most critical vulnerabilities found during the last scan. Administrators can easily see which machines need urgent attention.

- System Management: Administrators can add new machines to the platform through the WebPanel by specifying their details (name, IP, port, username, password, or SSH certificate). The WebPanel handles storing this information securely in the database and allows administrators to assign machines to specific users for better management.

- Scan Management: Administrators can initiate manual scans, schedule scans, or trigger a scan across multiple systems through the WebPanel.

- Scan Results Reporting: Results of scans, including detailed CVEs, system misconfigurations, and open ports, are displayed in organized containers and tables. Administrators can dive deeper into each machine's details to view scan results and download the complete report.

- Security and Permissions: All interactions with the WebPanel are secured through HTTPS, ensuring secure communication between users and the server. Additionally, sessions are used to limit access to unauthorized users.

4.4.2 Scanning Service

The Scanning Service is the core functionality of simpleVAP, allowing for a detailed vulnerability assessment of each systen added to the platform. The scanning service is responsible for the following tasks:

- SSH-Based Scanning: For each system, the scanning service establishes an SSH connection (via the paramiko library [21]) to remotely upload and execute the scanning scripts. This includes the bash script checks.sh for dependency checks and the Python script remote_script.py to collect system information and detect vulnerabilities.

- Package and Service Enumeration: The scanning script gathers a comprehensive list of installed packages, running services, and system configurations. The service identifies potential vulnerabilities related to outdated packages, misconfigured services, and security risks associated with open ports and weak user configurations.

- CPE and CVE Matching: Collected package names and versions are converted to CPE (Common Platform Enumeration) format using a locally stored dictionary (Pickle-based for fast lookup). The matched CPEs are then used to query the NVD (National Vulnerability Database) for relevant CVEs, which are returned to the server and organized in the scan report.

4.4.3 Nmap, Nikto and SQLMap Tools Integration

simpleVAP includes built-in tools like Nmap, Nikto, and SQLMap to perform additional network and vulnerability scans. These tools are integrated into the platform and can be configured through the WebPanel:

- Nmap Integration: Administrators can run Nmap scans to detect open ports and services and perform basic network mapping. The results are displayed in the WebPanel, providing insights into which services are exposed to potential attackers. Administrators can also configure specific Nmap scan options, such as performing a stealth scan or aggressive scanning mode. [15]

- Nikto Integration: Nikto is a web server vulnerability scanner that checks for common issues such as outdated server software, insecure configurations, and the presence of default files that could be exploited by attackers. Administrators can target specific systems and run Nikto scans to assess the security of web servers running on their Unix systems. [39]

- SQLMap Integration: SQLMap is a tool designed to detect and exploit SQL injection vulnerabilities in web applications. In simpleVAP, administrators can use the WebPanel to run SQLMap against web-facing services on their systems. The results include any detected SQL injection vulnerabilities and options to further probe for possible database compromise. [40]

4.4.4 Schedules Scanning and Automation

To reduce manual intervention, simpleVAP allows administrators to schedule scans at regular intervals. This is done using the python library Flask-APScheduler [41].This service ensures that the system remains vigilant and performs assessments at predefined times:

- Scheduled Scans: Administrators can set up scheduled scans for specific machines or groups of machines. This ensures that regular vulnerability assessments are conducted without manual triggering.

- Automation of Scan Reporting: Once a scheduled scan completes, the system generates a report automatically and updates the json file that contains the report of each system in the WebPanel.

4.4.5 Reporting and Analytics

simpleVAP provides detailed reporting and analytics for every scan that is performed. These reports offer insights into the vulnerabilities discovered, potential misconfigurations, and recommended fixes. Features of the reporting system include:

- Detailed JSON Reports: After each scan, a report is generated in JSON format, containing all the gathered information from the system. The report lists the vulnerabilities, services, installed packages, and CVEs detected during the scan.

- Visual Analytics: The WebPanel includes visualizations such as charts that allow administrators to quickly identify the most critical issues across their systems. High-priority vulnerabilities are highlighted in the dashboard, helping administrators prioritize their remediation efforts.

- Downloadable Reports: Administrators can download reports in multiple formats, including JSON and Excek, making it easier to share findings with managers or keep records for auditing purposes.

4.4.6 Modular Plugin System

The Modular Plugin System is an integral feature of simpleVAP, allowing administrators and advanced users to extend the functionality of the platform by uploading custom Python scripts.

This system empowers users to tailor vulnerability assessments to their specific needs by adding new security checks, scanning for proprietary software vulnerabilities, or automating particular security procedures that aren't covered by the core scanning scripts.

- Plugin Upload and Management:
    - Administrators can easily upload Python scripts via the WebPanel.
    - The WebPanel provides a management interface for enabling, disabling, or removing plugins.

- Integration with Core Scanning Process:
    - When a vulnerability scan is initiated on an system, any active plugins are executed in sequence after the core scripts (checks.sh and remote_script.py). These plugins receive the same environment variables and system access, allowing them to run specific commands or checks on the target machine.
    - The results of each plugin's execution are collected and merged with the main scan results, ensuring that all custom checks are included in the final report.

- Customizable Checks:
    - Custom Configuration Checks: Users can create Python scripts to check for specific system configurations that are unique to their environment. For example, checking for compliance with internal security policies or verifying proprietary configurations.
    - Proprietary Software Vulnerability Scanning: For enterprises using custom-built or proprietary software, plugins can be used to detect vulnerabilities or weaknesses that are not covered by public vulnerability databases like NVD/NIST.
    - Automation of Unique Tasks: Plugins can automate repetitive security tasks, such as cleaning up temporary files, auditing log files for specific patterns, or running custom security scripts regularly.

- Security and Execution Control:
    - The execution of plugins is controlled through a permissions system, ensuring that only authorized administrators can upload, modify, or execute plugins.

- Reporting and Feedback:
    - The results of each plugin's execution are returned to the central server, and the findings are integrated into the final scan report. Administrators can review the output of each plugin separately.
    - If a plugin detects a new vulnerability or configuration issue, it is flagged in the scan results alongside standard CVEs and misconfigurations.

- Example Use Cases
    - Proprietary Software Checks: A company using a custom web application may develop a plugin that checks for known security issues specific to their software stack. This plugin can be uploaded to simpleVAP and executed during every scan to ensure the custom application remains secure.

    - Compliance Audits: A plugin can be developed to verify compliance with internal security policies, such as checking if all machines have encrypted disks or if sensitive configuration files have proper access control settings.

    - Log File Audits: Administrators can upload a plugin that searches system log files for patterns indicating potential attacks or misconfigurations. For example, a plugin can scan for repeated failed login attempts and report suspicious activity directly into the scan results.

4.5 Technologies to be used

The development of simpleVAP integrates several modern technologies, each serving a distinct purpose in the platform's architecture. These technologies are selected based on their reliability, flexibility, and scalability to ensure a secure and efficient vulnerability assessment system for Unix environments.

1. Python
    a. Role:
        i. Python is the core programming language used for building the backend of simpleVAP. Its extensive free-to-use and open source libraries, easy syntax, and community support make it a great choice for implementing security-related tasks and handling complex data structures.
    b. Use Cases:
        i. Core scanning logic, handling SSH connections, file transfers, and vulnerability detection.
        ii. Interaction with third-party APIs (NVD/NIST) for CVE retrieval.
        iii. Building custom plugins for additional scanning functionality.

2. Flask
    a. Role:
        i. Flask is a lightweight web framework used for developing the WebPanel. Its simplicity and flexibility make it suitable for creating small to medium-sized web applications, while still providing the necessary components for handling user requests, managing sessions, and interacting with databases. [23]
    b. Use Cases:
        i. Managing user sessions, handling authentication, and presenting scan results.
        ii. Providing an interface for administrators to manage systems, trigger scans, and view vulnerability reports.
        iii. Handling plugin uploads and integrations via the web interface.

3. SQLite3
    a. Role:
        i. SQLite3 is a lightweight, serverless database engine used for storing critical information about systems, scan results, users, and tokens. It is highly suitable for the project due to its simplicity and low overhead, making it ideal for environments where a full-fledged database system may not be necessary. [42]
    b. Use Cases:
        i. Storing system information, user credentials, authentication tokens, and scan schedules.
        ii. Recording scan results and providing fast access to data for the WebPanel.

4. Paramiko (SSH):
    a. Role:
        i. Paramiko is a Python library for handling SSH connections, which are used to securely communicate with Unix systems. It allows simpleVAP to establish connections to remote machines, execute commands, and transfer files securely. [21]
    b. Use Cases:
        i. Establishing SSH connections to remote systems.
        ii. Executing scripts (e.g., checks.sh, remote_script.py) on the target machines for system scanning.
        iii. Handling secure file transfers via SFTP for scanning scripts and scan results.

5. Bootstrap 5.3
    a. Role:
        i. Bootstrap is a front-end framework used to design the user interface for the WebPanel. It provides a responsive and mobile-friendly layout with pre-built components, making the UI visually appealing and easy to navigate. [43]
    b. Use Cases:

        i. Creating a responsive, user-friendly interface that works across different devices.
        ii. Providing customizable UI components like tables, forms, and modals for managing scans and displaying results.

6. Pickle (for CPE Matching)
    a. Role:
        i. The Pickle module in Python is used to serialize and store the CPE dictionary. This enables faster lookups of Common Platform Enumerations (CPE) during the vulnerability assessment process, improving the performance of the CPE-to-CVE matching workflow.
    b. Use Cases:
        i. Storing a preprocessed CPE dictionary in a serialized format for quick access during scans.
        ii. Loading the dictionary into memory when needed for fast CPE matching and minimizing disk read times.

7. NVD API (National Vulnerability Database)
    a. Role:
        i. The NVD API is used to query the National Vulnerability Database for the latest CVE data based on the CPE identifiers obtained during the scan. This integration allows simpleVAP to provide administrators with up-to-date vulnerability information for each package or service found on the system.
    b. Use Cases:
        i. Fetching relevant CVEs for matched CPEs during scans.
        ii. Regularly querying the NVD with found CPEs to ensure that the platform is aware of newly published vulnerabilities.

8. Nmap, Nikto, and SQLMap

    a. Role:
        i. These open-source security tools are integrated into simpleVAP to provide additional scanning capabilities. Each tool targets a specific aspect of system or network security, giving administrators the ability to run more specialized tests alongside the core vulnerability assessments.
    b. Nmap: Used for network discovery and identifying open ports, services, and OS detection. [15]
    c. Nikto: A web server scanner that checks for outdated software, default files, and other web-based vulnerabilities. [39]
    d. SQLMap: Detects and exploits SQL injection vulnerabilities in web applications. [40]

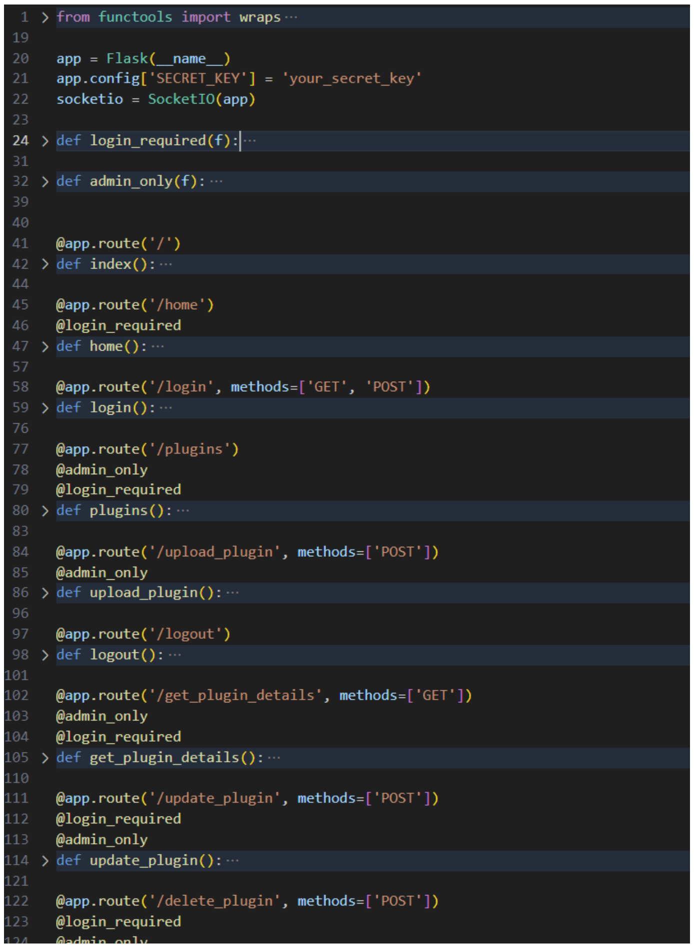
9. Cron (Linux Scheduling)
    a. Role:
        i. Cron is a Unix-based job scheduler used to automate regular tasks. In simpleVAP, cron is used to schedule scans and ensure that machines regularly ping the server to report their status and IP addresses. [44]
    b. Use Cases:
        i. Running regular system pings to maintain an updated machine list.
        ii. Scheduling vulnerability scans at predefined intervals without requiring manual intervention.

10. JavaScript and jQuery (for Interactive UI)
    a. Role:
        i. JavaScript and jQuery are used to enhance the interactivity of the WebPanel, allowing users to interact with the UI seamlessly. jQuery simplifies DOM manipulation and event handling, making the front-end more dynamic and responsive. [45] [46]
    b. Use Cases:
        i. Handling asynchronous actions in the WebPanel, such as starting a scan without reloading the page.

11. Shell Scripting (.sh Scripts)
    a. Role:
        i. The use of bash scripts in simpleVAP allows for platform-independent execution of various system checks and dependency installation tasks. [47]
    b. Use Cases:
        i. checks.sh: This script is uploaded to remote machines to verify whether Python and other required dependencies are installed. If dependencies are missing, the script installs them automatically, ensuring the system is prepared for the main vulnerability scan.
        ii. Automation of basic tasks: Automates initial system setup and preparation, reducing the need for manual configuration on each machine before a scan can be performed.

5. Implementation and Testing

5.1 Code Structure Explained



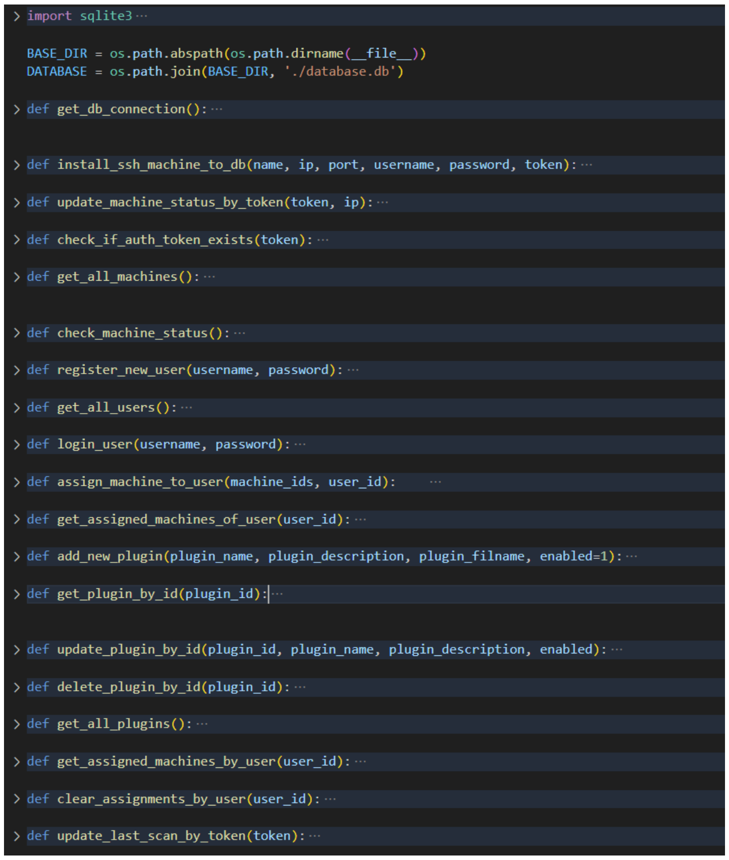
```python
from functools import wraps···

app = Flask(__name__)
app.config['SECRET_KEY'] = 'your_secret_key'
socketio = SocketIO(app)

def login_required(f):···

def admin_only(f):···


@app.route('/')
def index():···

@app.route('/home')
@login_required
def home():···

@app.route('/login', methods=['GET', 'POST'])
def login():···

@app.route('/plugins')
@admin_only
@login_required
def plugins():···

@app.route('/upload_plugin', methods=['POST'])
@admin_only
def upload_plugin():···

@app.route('/logout')
def logout():···

@app.route('/get_plugin_details', methods=['GET'])
@admin_only
@login_required
def get_plugin_details():···

@app.route('/update_plugin', methods=['POST'])
@login_required
@admin_only
def update_plugin():···

@app.route('/delete_plugin', methods=['POST'])
@login_required
@admin_only
```

Figure 5 Snippet of app.py code

- **app.py**

The app.py file is the central controller of the simpleVAP platform, built with Flask for managing web routes, user authentication, and interactions between the web interface and backend. It orchestrates various functionalities:

1. Routing: Defines endpoints like /login, /home, and /upload_plugin, directing requests to appropriate functions and ensuring access control through decorators (login_required, admin_only).
2. User Management: Handles session-based login and admin authentication, ensuring secure access to the platform's functions.
3. Real-Time Communication: Uses SocketIO to provide live updates during vulnerability scans, pushing data to the client without page refresh.
4. Plugin Management: Allows for the dynamic uploading and execution of custom Python plugins during scans, giving the platform extensibility.
5. Vulnerability Scanning: Coordinates the initiation of remote scans on systems using various tools and modules (e.g., scan.py, nmap_scan.py) and integrates results into the system.



```python
> import sqlite3···

  BASE_DIR = os.path.abspath(os.path.dirname(__file__))
  DATABASE = os.path.join(BASE_DIR, './database.db')

> def get_db_connection():···


> def install_ssh_machine_to_db(name, ip, port, username, password, token):···

> def update_machine_status_by_token(token, ip):···

> def check_if_auth_token_exists(token):···

> def get_all_machines():···


> def check_machine_status():···

> def register_new_user(username, password):···

> def get_all_users():···

> def login_user(username, password):···

> def assign_machine_to_user(machine_ids, user_id):    ···

> def get_assigned_machines_of_user(user_id):···

> def add_new_plugin(plugin_name, plugin_description, plugin_filname, enabled=1):···

> def get_plugin_by_id(plugin_id):···


> def update_plugin_by_id(plugin_id, plugin_name, plugin_description, enabled):···

> def delete_plugin_by_id(plugin_id):···

> def get_all_plugins():···

> def get_assigned_machines_by_user(user_id):···

> def clear_assignments_by_user(user_id):···

> def update_last_scan_by_token(token):···
```

Figure 6 Snippet of db_handler.py code

- **db_handler.py**

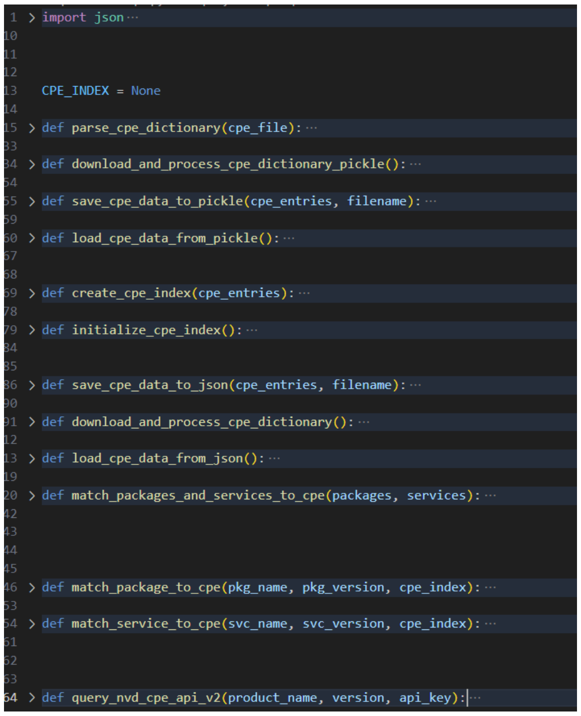
The db_handler.py file manages all database interactions in the simpleVAP platform. It uses SQLite to store and retrieve essential information such as user credentials, machine details, plugin metadata, and scan history. It establishes connections to the database and executes SQL queries to perform various operations.

This file handles:

- User Management: Registering and authenticating users, storing their credentials securely, and managing user access to machines.
- Machine and Scan Management: Storing information about the machines being scanned, their SSH credentials, and updating the status of each machine after scans are completed.
- Plugin Management: Adding, updating, and deleting plugins uploaded by administrators, as well as retrieving plugin details for use during scans.

- **cpe.py and cve.py**

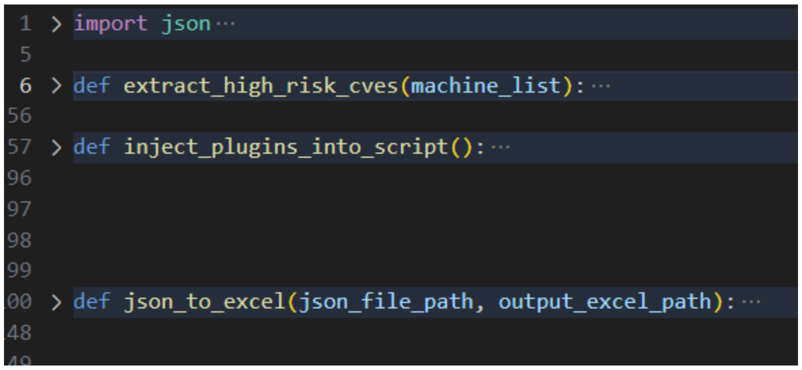
The cpe.py and cve.py files work together to identify vulnerabilities on a system by leveraging the Common Platform Enumeration (CPE) and Common Vulnerabilities and Exposures (CVE) standards.



```python
import json⋯


CPE_INDEX = None

def parse_cpe_dictionary(cpe_file):⋯

def download_and_process_cpe_dictionary_pickle():⋯

def save_cpe_data_to_pickle(cpe_entries, filename):⋯

def load_cpe_data_from_pickle():⋯


def create_cpe_index(cpe_entries):⋯

def initialize_cpe_index():⋯


def save_cpe_data_to_json(cpe_entries, filename):⋯

def download_and_process_cpe_dictionary():⋯

def load_cpe_data_from_json():⋯

def match_packages_and_services_to_cpe(packages, services):⋯




def match_package_to_cpe(pkg_name, pkg_version, cpe_index):⋯

def match_service_to_cpe(svc_name, svc_version, cpe_index):⋯



def query_nvd_cpe_api_v2(product_name, version, api_key):⋯
```

Figure 7 Snippet of cpe.py code

- cpe.py: This file processes the CPE dictionary and matches software packages and services on a target machine to their respective CPE entries. It handles downloading, parsing, and storing CPE data for quick lookup, enabling the system to link installed packages to known vulnerabilities.
- cve.py: Once CPEs are matched to the installed software, cve.py uses the NVD API to retrieve CVEs related to the matched CPEs. It fetches detailed vulnerability information, such as CVE IDs and severity scores, which help in assessing the risk level of each software component.

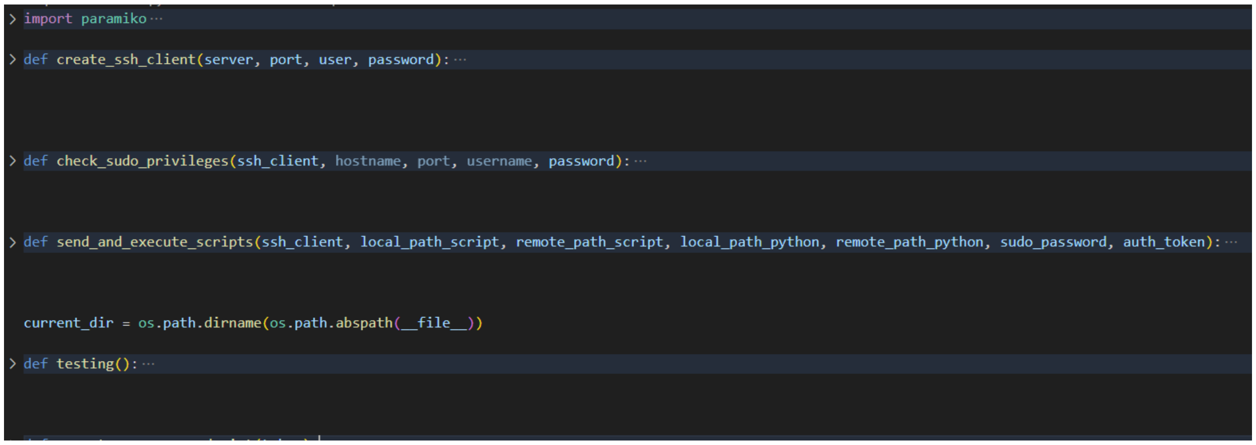


```python
import json⋯

def extract_high_risk_cves(machine_list):⋯

def inject_plugins_into_script():⋯




def json_to_excel(json_file_path, output_excel_path):⋯
```

Figure 8 Snippet of helper.py code

- **helper.py**

The helper.py file serves as a utility module for the simpleVAP platform, assisting with critical tasks such as vulnerability extraction, plugin management, and report generation.



```
> import paramiko···

> def create_ssh_client(server, port, user, password):···



> def check_sudo_privileges(ssh_client, hostname, port, username, password):···


> def send_and_execute_scripts(ssh_client, local_path_script, remote_path_script, local_path_python, remote_path_python, sudo_password, auth_token):···


  current_dir = os.path.dirname(os.path.abspath(__file__))

> def testing():···
```

Figure 9 Snippet of scan.py code

- **scan.py**

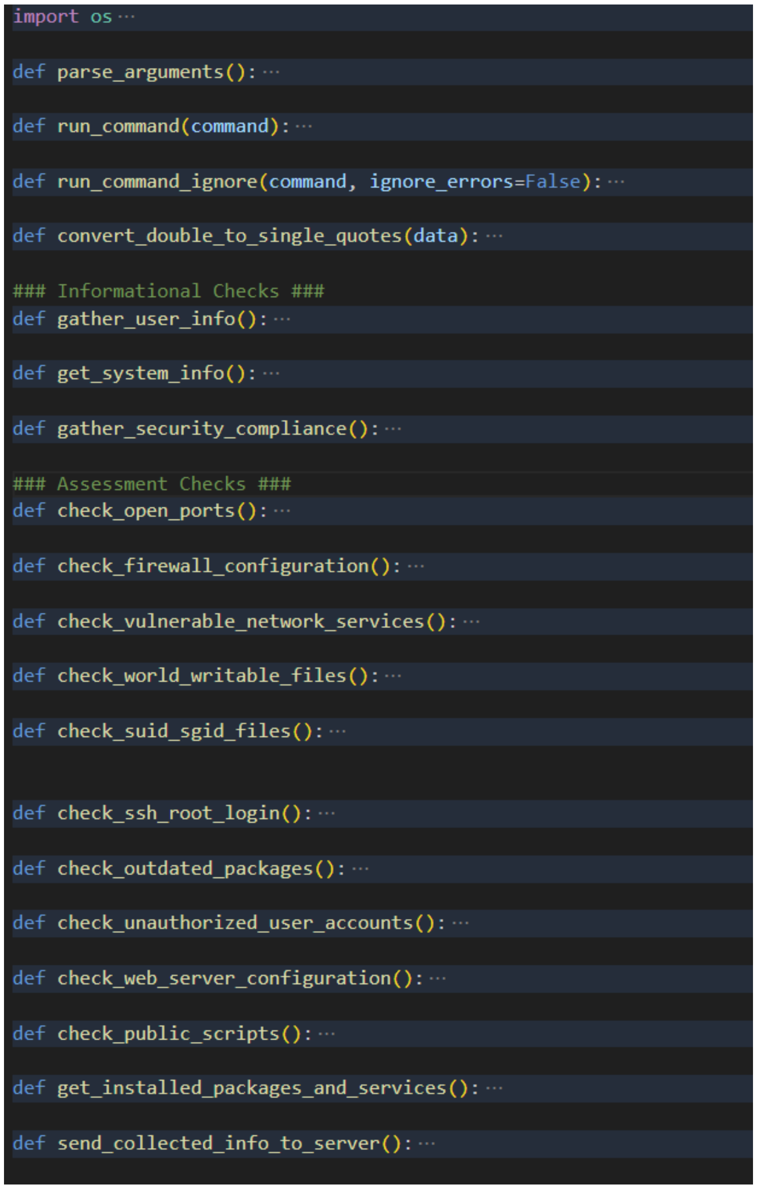
The scan.py file is the core component responsible for managing remote script execution on systems in simpleVAP. It uses Paramiko to establish SSH connections, securely transfer scanning scripts to remote machines, and execute them. The file also checks for sudo privileges on the remote system, ensuring that the necessary commands run with appropriate access levels.

By automating the deployment and execution of scripts across multiple systems, scan.py organizes the scanning process. It also integrates plugins into the scanning scripts, allowing for customizable vulnerability checks. This file plays an important role in gathering system data and executing the vulnerability assessment remotely on each machine.

- **remote_script.py**

The remote_script.py file is designed to be executed on the remote system, gathering system information, checking for security vulnerabilities, and sending the results back to the central server. It performs various checks on system configurations, installed packages, network services, and user accounts, providing both informational and security-related data.

- Collects Information: It retrieves system details like the hostname, IP addresses (both internal and external), users, and security configurations (e.g., SELinux, AppArmor).
- Performs Security Checks: It checks for open ports, firewall configurations, vulnerable network services, world-writable files, SUID/SGID files, and root login permissions.
- Runs Vulnerability Assessments: The script checks for outdated packages, unauthorized users, insecure web server configurations, and public scripts.
- Sends Data to Server: After collecting all relevant information and performing assessments, the script compiles the data into a structured format and sends it back to the server for further analysis and reporting.



```
import os ···

def parse_arguments(): ···

def run_command(command): ···

def run_command_ignore(command, ignore_errors=False): ···

def convert_double_to_single_quotes(data): ···

### Informational Checks ###
def gather_user_info(): ···

def get_system_info(): ···

def gather_security_compliance(): ···

### Assessment Checks ###
def check_open_ports(): ···

def check_firewall_configuration(): ···

def check_vulnerable_network_services(): ···

def check_world_writable_files(): ···

def check_suid_sgid_files(): ···


def check_ssh_root_login(): ···

def check_outdated_packages(): ···

def check_unauthorized_user_accounts(): ···

def check_web_server_configuration(): ···

def check_public_scripts(): ···

def get_installed_packages_and_services(): ···

def send_collected_info_to_server(): ···
```

Figure 10 Snippet of remote_script.py code

5.2 Implementation Aspects

The implementation of simpleVAP involved making a series of decisions that ensured the platform's performance in achieving its goals of providing a simple, flexible, and secure vulnerability assessment tool. Each aspect of the development process was considered, from choosing the right technologies to making the platform extensible and efficient. The key aspects of the implementation and the reasoning behind choices are as follows.

- NVD as the CPE and CVE Database Source

The National Vulnerability Database (NVD) [25] was selected as the primary source for both CPE (Common Platform Enumeration) matching and CVE (Common Vulnerabilities and Exposures) retrieval. Several factors made NVD the ideal choice:

Comprehensive and Widely Recognized: The NVD is the most comprehensive and widely accepted database for tracking security vulnerabilities. It is continuously updated with the latest vulnerabilities, ensuring that administrators receive the most up-to-date information when scanning their systems. [48]

Standardization of CPE: NVD's adherence to CPE standards ensures that package names and versions are mapped to known software vulnerabilities. This made it easier for simpleVAP to automate the process of matching installed software/services to CPE entries.

API Accessibility: The NVD provides a public API for querying CVEs based on CPE data, which allowed for seamless integration into simpleVAP's scanning process. By leveraging this API, simpleVAP ensures that administrators can always retrieve the most relevant CVEs for the software running on their systems. [26]

Alternative databases were considered, but none offered the same level of comprehensive coverage, consistent updates, and ease of access as NVD.

- Using Pickle for the CPE Dictionary

When it came to storing and accessing the CPE dictionary for package matching, Pickle was chosen over other formats like JSON for its speed and efficiency in handling large data sets.

Serialization Speed: Pickle is a Python-native serialization format, meaning it is much faster when it comes to loading and saving data in Python-based applications. Since the CPE dictionary contains large numbers of entries, speed was important to ensure that scan times remained low even when scanning systems with many installed packages.

Efficient Lookups: Unlike JSON, which is more suited for human-readable data, Pickle is optimized for machine-readability. By storing the CPE dictionary in Pickle format, simpleVAP could quickly deserialize the data into Python objects, significantly speeding up CPE lookups during scans. [49]

Lower Overhead: Pickle avoids the need for string parsing and conversion (which is required when working with JSON), reducing overhead and improving the platform's performance, especially during the CPE matching phase.

- SSH with Paramiko vs. Manual Upload of Scripts

One of the major implementation decisions was using Paramiko for SSH connections rather than having users manually upload scanning scripts to each system.

Automation and Efficiency: Automating the script upload and execution process via SSH with Paramiko ensures that administrators do not need to manually interact with each system. This significantly reduces human effort, especially in environments with a large number of machines. Automating these interactions not only saves time but also reduces the chance of errors in configuration or execution.

Security: Using Paramiko's secure SSH communication ensures that all file transfers and command executions are encrypted, minimizing the risk of eavesdropping or tampering. In contrast, manual uploading could introduce security risks, as it would rely on non-automated, potentially less secure methods of file transfer.

Error Handling: Paramiko allows simpleVAP to handle errors gracefully. If a connection fails or the script encounters an issue, the system can log the error, alert the administrator, and retry if necessary. This level of automation and control is difficult to achieve with manual uploads.

Consistency Across Machines: With SSH and Paramiko, the same scripts are automatically deployed and executed on each machine in a consistent manner. This consistency ensures that each system is scanned under the same conditions, reducing the risk of configuration drift or human error. [50]

- Developing the Plugin System

The plugin system was one of the more challenging aspects of simpleVAP's implementation, as it poses certain security risks.

Plugin Management via WebPanel: The WebPanel was extended to allow Administrators to upload Python scripts that could be executed during the scanning process. Plugins are integrated into the scanning pipeline and can be managed (enabled, disabled, or removed) through the web interface.

Customizability: The plugin system provides an open-ended way for administrators to extend simpleVAP's functionality without altering the core scanning scripts. This allows for custom security checks and vulnerability detection tailored to the specific environment in which simpleVAP is deployed.

- Error Handling and Logging

Throughout the implementation, error handling was integrated at multiple levels of the system to ensure smooth operation and easier debugging.

Logging: Detailed logs are generated for SSH connections, script executions, plugin activity, and API calls to NVD. These logs allow administrators to troubleshoot any issues that arise during scans or interactions with remote systems.

Retry Mechanisms: In case of errors, such as a loss of SSH connectivity or an API timeout when querying NVD, the system is designed to retry operations, ensuring that temporary issues don't result in failed scans.

5.3 Test Cases

During the implementation and development phase of simpleVAP, various test cases were executed to ensure the robustness and useability of the platform in performing vulnerability assessments. These tests were done across multiple Linux distributions, including both consumer-grade and server-grade systems, to cover a wide spectrum of environments:

Linux Distributions Used for Testing:

- Debian: Tested for stability and long-term support capabilities, focusing on server environments. [51]
- Ubuntu Desktop: Used to test consumer-grade systems, particularly focusing on systems with graphical user interfaces. [52]
- Ubuntu Server: Tested server-grade installations, ensuring compatibility with headless environments. [53]
- CentOS 6: Verified legacy server-grade systems with older software versions to ensure backward compatibility. [54]
- MX Linux: A lightweight consumer-grade system, tested for performance on lower-resource hardware. [55]
- Kali Linux: Tested specifically for compatibility with security-focused environments where penetration testing tools are frequently used. [56]

Types of Test Cases:

1. Basic Functionality Tests:
    a. Verified the ability to add machines and SSH into systems.
    b. Confirmed the correct execution of remote scripts on various distributions.
    c. Tested the web interface's ability to process and display CVEs for each machine.

2. Vulnerability Detection Tests:

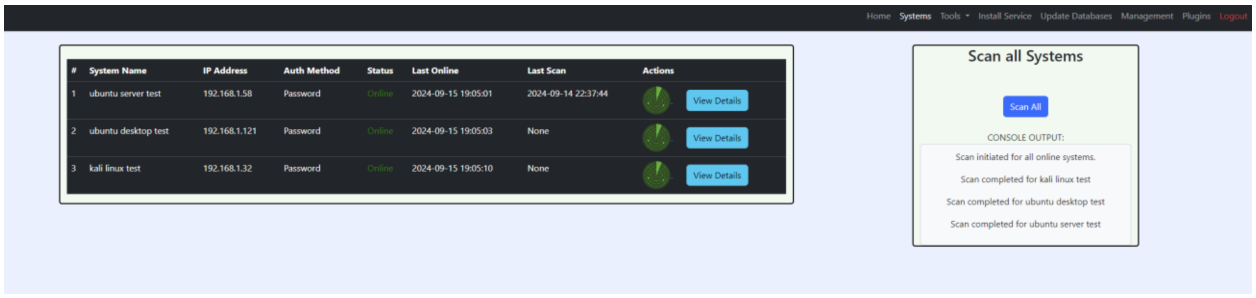
   a. Checked that the tool correctly identified vulnerable packages and services based on installed software versions.
   b. Tested the accuracy of CVE retrieval by matching CPEs from the NVD database.

3. Plugin Integration:
   a. Tested the dynamic plugin system to ensure that user-uploaded plugins could be seamlessly integrated into the remote script and executed during scans.
   b. Verified that plugin results were correctly displayed in the vulnerability report.

4. System Performance:
   a. Ensured that system performance was not significantly impacted during scans on lightweight distros like MX Linux and resource-intensive scans on server-grade distros like Ubuntu Server.
   b. Verified the proper execution of scripts without causing system crashes or interruptions.

5. Error Handling:
   a. Tested edge cases, such as missing dependencies on target machines or failed SSH connections, ensuring the system handled these gracefully with error messages and retry options.

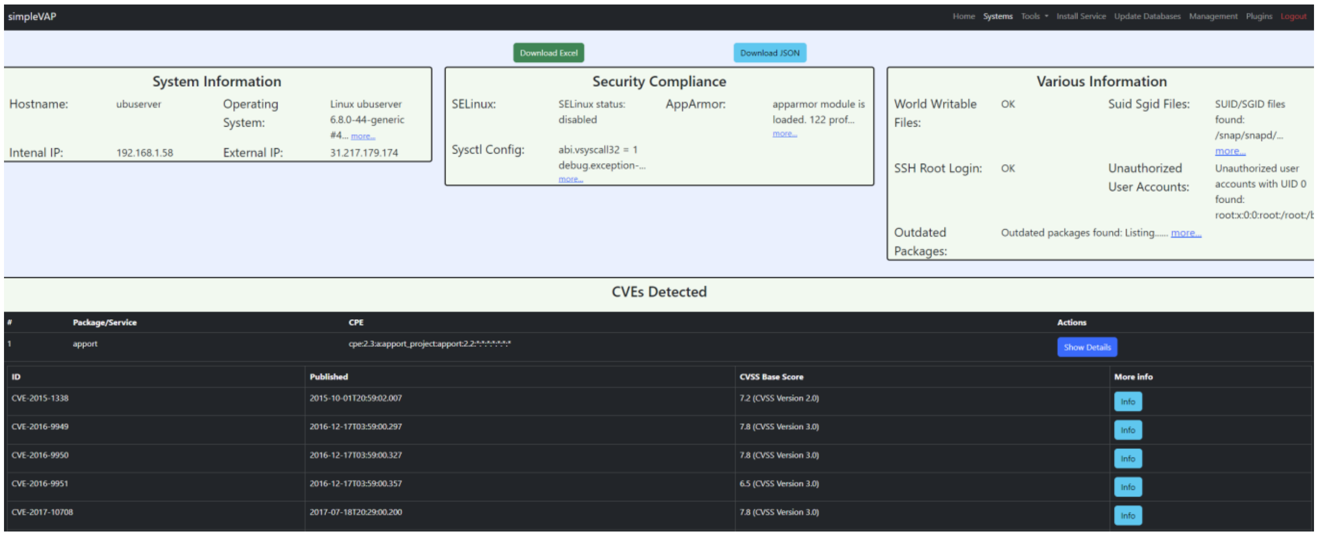
5.4 Testing Reports – screenshots



#	System Name	IP Address	Auth Method	Status	Last Online	Last Scan	Actions
1	ubuntu server test	192.168.1.58	Password	Online	2024-09-15 19:05:01	2024-09-14 22:37:44	View Details
2	ubuntu desktop test	192.168.1.121	Password	Online	2024-09-15 19:05:03	None	View Details
3	kali linux test	192.168.1.32	Password	Online	2024-09-15 19:05:10	None	View Details

```
CONSOLE OUTPUT:
Scan initiated for all online systems.
Scan completed for kali linux test
Scan completed for ubuntu desktop test
Scan completed for ubuntu server test
```

Figure 11 Adding new systems and executing a remote scan



System Information

Hostname:	ubuserver	Operating System:	Linux ubuserver 6.8.0-44-generic #4... more...
Intenal IP:	192.168.1.58	External IP:	31.217.179.174

Security Compliance

SELinux:	SELinux status: disabled	AppArmor:	apparmor module is loaded. 122 prof... more...
Sysctl Config:	abi.vsyscall32 = 1 debug.exception-... more...		

Various Information

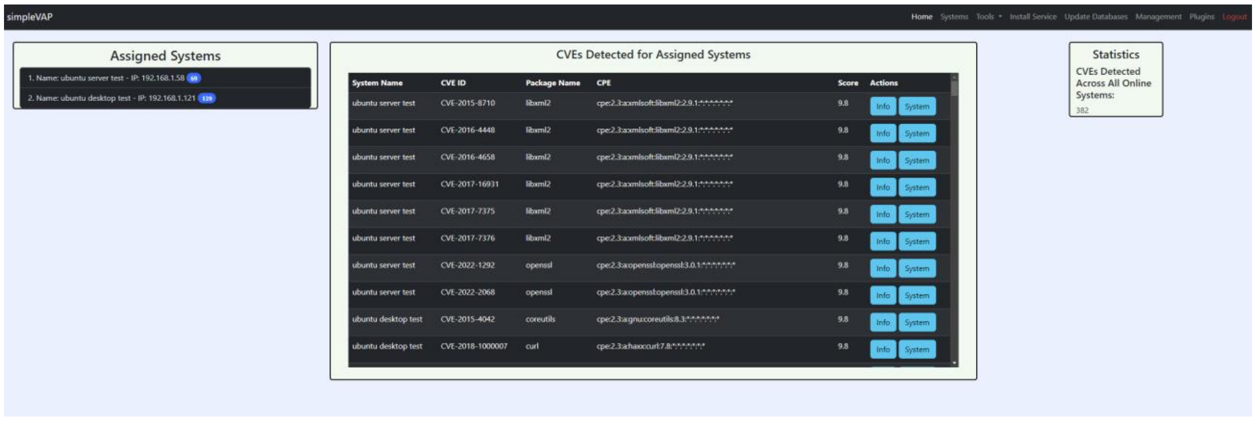
World Writable Files:	OK	Suid Sgid Files:	SUID/SGID files found: /snap/snapd/... more...
SSH Root Login:	OK	Unauthorized User Accounts:	Unauthorized user accounts with UID 0 found: root:x:0:0:root:/root:/b
Outdated Packages:	Outdated packages found: Listing...... more...		

CVEs Detected

#	Package/Service	CPE	Actions
1	apport	cpe:2.3:a:apport_project:apport:2.2:*:*:*:*:*:*:*	Show Details

ID	Published	CVSS Base Score	More info
CVE-2015-1338	2015-10-01T20:59:02.007	7.2 (CVSS Version 2.0)	Info
CVE-2016-9949	2016-12-17T03:59:00.297	7.8 (CVSS Version 3.0)	Info
CVE-2016-9950	2016-12-17T03:59:00.327	7.8 (CVSS Version 3.0)	Info
CVE-2016-9951	2016-12-17T03:59:00.357	6.5 (CVSS Version 3.0)	Info
CVE-2017-10708	2017-07-18T20:29:00.200	7.8 (CVSS Version 3.0)	Info

Figure 12 Viewing details of a specific system scan



Assigned Systems

1. Name: ubuntu server test - IP: 192.168.1.58 69
2. Name: ubuntu desktop test - IP: 192.168.1.121 139

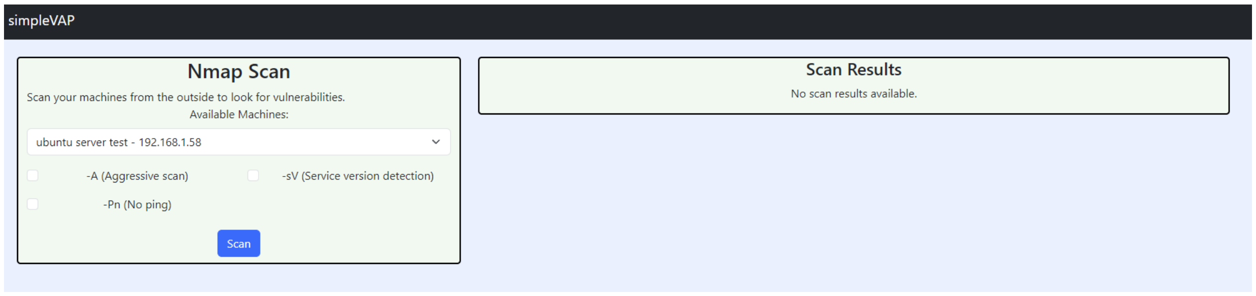
CVEs Detected for Assigned Systems

System Name	CVE ID	Package Name	CPE	Score	Actions
ubuntu server test	CVE-2015-8710	libxml2	cpe:2.3:a:xmlsoft:libxml2:2.9.1:*:*:*:*:*:*:*	9.8	Info System
ubuntu server test	CVE-2016-4448	libxml2	cpe:2.3:a:xmlsoft:libxml2:2.9.1:*:*:*:*:*:*:*	9.8	Info System
ubuntu server test	CVE-2016-4658	libxml2	cpe:2.3:a:xmlsoft:libxml2:2.9.1:*:*:*:*:*:*:*	9.8	Info System
ubuntu server test	CVE-2017-16931	libxml2	cpe:2.3:a:xmlsoft:libxml2:2.9.1:*:*:*:*:*:*:*	9.8	Info System
ubuntu server test	CVE-2017-7375	libxml2	cpe:2.3:a:xmlsoft:libxml2:2.9.1:*:*:*:*:*:*:*	9.8	Info System
ubuntu server test	CVE-2017-7376	libxml2	cpe:2.3:a:xmlsoft:libxml2:2.9.1:*:*:*:*:*:*:*	9.8	Info System
ubuntu server test	CVE-2022-1292	openssl	cpe:2.3:a:openssl:openssl:3.0.1:*:*:*:*:*:*:*	9.8	Info System
ubuntu server test	CVE-2022-2068	openssl	cpe:2.3:a:openssl:openssl:3.0.1:*:*:*:*:*:*:*	9.8	Info System
ubuntu desktop test	CVE-2015-4042	coreutils	cpe:2.3:a:gnu:coreutils:8.3:*:*:*:*:*:*:*	9.8	Info System
ubuntu desktop test	CVE-2018-1000007	curl	cpe:2.3:a:haxx:curl:7.8:*:*:*:*:*:*:*	9.8	Info System

Statistics

CVEs Detected Across All Online Systems: 382

Figure 13 Upon login the user is presented with his assigned systems, and a sorted list (high to low CVSS) of CVES



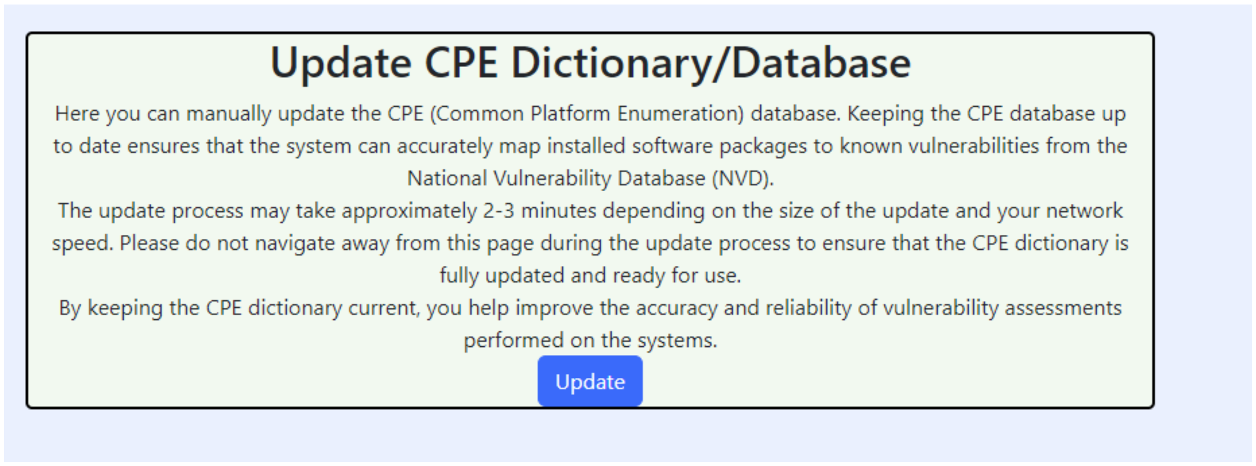
Scan your machines from the outside to look for vulnerabilities.

Available Machines: ubuntu server test - 192.168.1.58

☐ -A (Aggressive scan)
☐ -sV (Service version detection)
☐ -Pn (No ping)

No scan results available.

Figure 14 System Administrators can use tools like nmap to further evaluate a system


Update CPE Dictionary/Database

Here you can manually update the CPE (Common Platform Enumeration) database. Keeping the CPE database up to date ensures that the system can accurately map installed software packages to known vulnerabilities from the National Vulnerability Database (NVD).

The update process may take approximately 2-3 minutes depending on the size of the update and your network speed. Please do not navigate away from this page during the update process to ensure that the CPE dictionary is fully updated and ready for use.

By keeping the CPE dictionary current, you help improve the accuracy and reliability of vulnerability assessments performed on the systems.

Figure 15 An admin of the platform can start an CPE database update

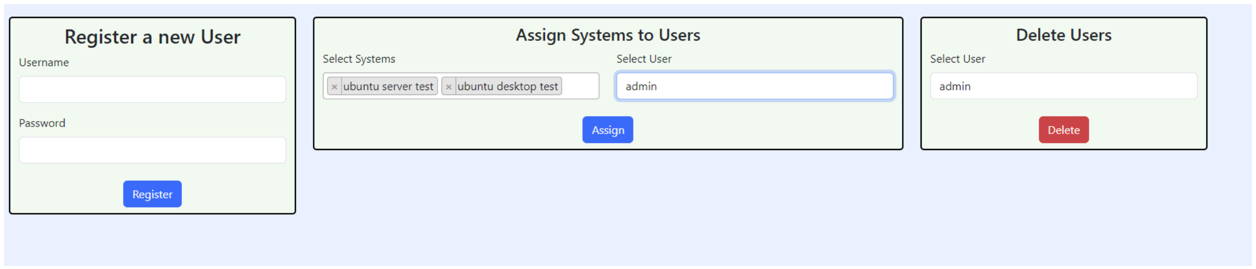


Figure 16 Users are registered through and already existing admin of the platform, who can also assign systems for them

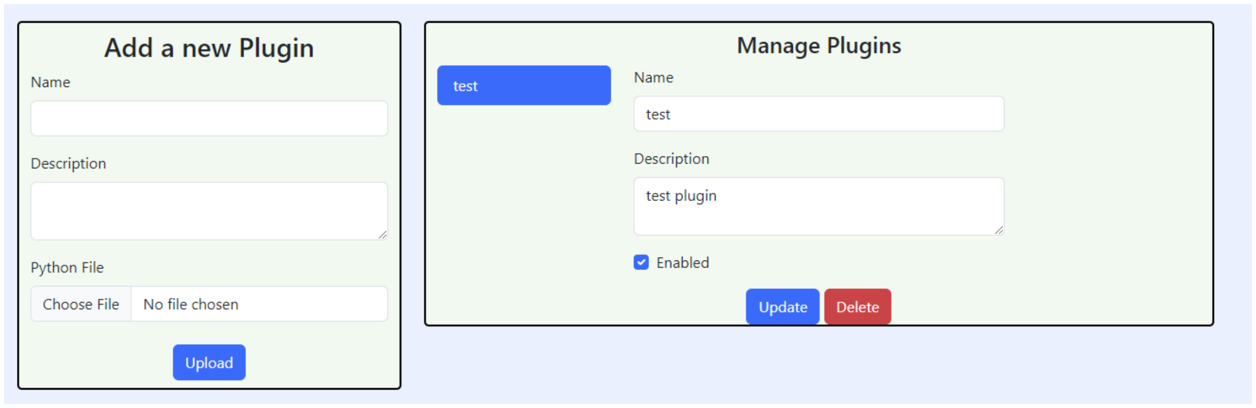


Figure 17 In the plugin page custom python scripts can be uploaded to the platform to be executed along the standard scan

6.1 Evaluation methodology and metrics

The evaluation of simpleVAP was based on how effectively the platform performed vulnerability assessments across various Linux distributions, focusing on its accuracy, performance, and usability. To assess the tool, several things were considered:

- Vulnerability Detection: The accuracy of the platform was assessed by comparing its scan results to known vulnerabilities in installed software on test systems. The ability to correctly identify and retrieve relevant CVEs (Common Vulnerabilities and Exposures) was a central aspect of the evaluation.

- System Compatibility: Tests were performed on multiple Linux distributions, ranging from consumer-grade systems (like Ubuntu Desktop) to server-grade systems (like Ubuntu Server and CentOS 6). The platform's ability to consistently execute scans and generate reports across different environments was an important metric.

- Performance: The speed of vulnerability scans and resource consumption (CPU, memory, and network) during the scans were monitored. This helped ensure that the tool operates efficiently without overloading the target systems or the server hosting the platform.

- Scalability: The platform was evaluated for its ability to handle multiple system simultaneously without affecting performance or accuracy. This was important for environments where large-scale vulnerability assessments are required.

- Security of the Web Panel: The security of the web interface was a vital part of the evaluation. Access to the web panel was secured through user authentication, and session management ensured that sensitive operations, such as vulnerability scans and plugin uploads, were only accessible to authenticated users. The platform was also evaluated for protection against common web vulnerabilities, such as unauthorized access and injection attacks.

- Ease of Use: The usability of the web interface, including how easily users can navigate between functionalities, upload plugins, view results, and manage systems, was evaluated through user feedback.

6.2 Evaluation report

The evaluation showed that simpleVAP performs reliably across all tested Linux distributions, offering both accuracy in detecting vulnerabilities and scalability in handling multiple systems.

- Vulnerability Detection: The tool successfully detected relevant CVEs on all systems, along with identifying outdated packages and misconfigurations. The CPE matching was accurate and manual cross referencing was done. CVE retrieval from the NVD API worked as it should, providing administrators with comprehensive reports.

- System Compatibility: The platform worked across a wide range of Linux distributions with differing pre-installed software and package managers. This should ensure that platform can continue its operation indefinitely with small adjustments to the code.

- Performance: Scans were performed fast enough across all systems, with minimal resource usage observed. Even when scanning multiple systems consecutively, the platform maintained its performance, showing scalability and efficiency in handling significant numbers of systems.

- Security of the Web Panel: While the entire platform is not meant to be exposed to the internet it still proved to be secure, with proper user authentication and session management ensuring that only authorized users could perform critical tasks. During testing, the platform showed resistance to common web vulnerabilities, such as unauthorized access and injection attempts, ensuring data protection and secure operations.

- Ease of Use: The process of adding machines, scheduling scans, and viewing CVEs was straightforward, and the ability to upload custom plugins provided additional flexibility. In addition, the use of the Bootstrap framework, allows the panel to be usable on mobile devices.

7. Conclusion and further research

7.1 Potential improvements

Although simpleVAP being a comprehensive tool, there are several areas for future enhancement to further improve the platform’s functionality, security, and ease of use. These potential improvements would provide even greater value for system administrators and security professionals.

1. Forced Two-Factor Authentication (2FA) for Admin Users

To increase the security of the web panel, a mandatory 2FA [57] for all admin users would be an essential improvement. Given the sensitive nature of the vulnerability data and the elevated permissions of admin users, implementing a second authentication layer (such as TOTP or hardware-based keys) would help mitigate the risk of unauthorized access to the system. This would significantly improve the overall security of the web platform, especially in environments dealing with highly sensitive data.

2. More Sandboxed and Smarter Handling of Plugins

The current plugin system allows administrators to upload custom Python code to extend the platform's scanning capabilities. However, to enhance security and stability, a more sandboxed approach should be adopted. Plugins should be executed in isolated environments where they cannot interfere with the core system or cause potential security risks.

Additionally, introducing a smarter plugin management system that can validate plugin code for potential issues (e.g., performance bottlenecks) before execution would prevent poorly written plugins from slowing down the system. Enhanced error reporting and rollback mechanisms could also be introduced to ensure that any faulty plugin does not affect ongoing scans or the integrity of the system.

3. Risk Analysis for Each System

Currently, the platform focuses primarily on identifying vulnerabilities. However, introducing risk analysis functionality for each system could offer more comprehensive insights into the overall security. By combining CVSS scores with additional factors—such as the criticality of the system, the exposure to external networks, and the presence of mitigations—the platform could offer a clearer overview of the actual risk each system poses. This feature would enable administrators to prioritize their responses more effectively, focusing on high-risk systems first.

4. Integration of AI and Machine Learning

AI could significantly improve simpleVAP's ability to detect vulnerabilities and predict potential risks. By incorporating machine learning models that analyze historical data, the system could offer proactive security recommendations based on emerging trends. AI could also be integrated to dynamically prioritize vulnerabilities by learning which CVEs are most commonly exploited and offering tailored recommendations to administrators.

5. Enhanced Reporting and Visualization

While the platform currently displays vulnerabilities and CVEs, enhancing the reporting and visualization capabilities would make it easier for administrators to interpret scan results. Introducing interactive charts, graphs, and heat maps that represent the overall vulnerability landscape for all systems could provide a more intuitive overview of the platform's findings. This improvement would also help in tracking the progress of security over time, as the platform could visualize the reduction in risks as systems are patched.

6. Improved System Resource Monitoring

Adding system resource monitoring during scans would allow the platform to track CPU, memory, and network usage on each system during vulnerability assessments. This would provide administrators with feedback on whether scans are impacting the system's performance and could help them adjust scheduling to minimize disruption to production environments.

7. Cross-Platform Support Beyond Unix Systems

The platform is currently focused on Unix-based systems, but extending support for Windows-based systems would make simpleVAP more versatile. This could be accomplished by developing Windows-compatible scripts for remote scans and vulnerability assessments, allowing administrators to manage both Unix and Windows environments from a single dashboard.

8. 8. Offline Scanning Capabilities

An offline scanning mode could be introduced, allowing administrators to run scans on isolated systems that do not have direct access to the platform's web server. By enabling users to export scan data manually (e.g., through USB drives) and import the results back into the platform for analysis, simpleVAP could become more flexible for use in environments with stringent network restrictions.

7.2 Suggestions for further research

As the development of simpleVAP is continued, there are several avenues for further research that can expand the platform's capabilities. These areas of research explore advancements in automation, security, and intelligence integration, offering opportunities to enhance the effectiveness and efficiency of vulnerability detection and risk management.

- AI-Driven Vulnerability Prediction

While integrating AI for vulnerability detection has been proposed as an immediate improvement, further research could explore predictive models to forecast potential vulnerabilities before they are publicly disclosed. By analyzing historical data on vulnerabilities and attack patterns, machine learning algorithms could potentially predict which software components are likely to become vulnerable, allowing administrators to take proactive security measures. [58]

- Behavioral-Based Threat Detection

Traditional vulnerability assessment tools focus on identifying known vulnerabilities in software and configurations. However, combining behavioral analysis with traditional scanning techniques could provide deeper insights into potential threats. Research could focus on integrating behavioral monitoring to detect anomalous or suspicious activities within the system, complementing the CVE-based vulnerability detection methods. [59]

- Cross-Platform Vulnerability Assessment

While simpleVAP focuses on Unix-based systems, further research into cross-platform compatibility could provide a unified solution for both Unix and Windows systems. Investigating the differences in vulnerability assessment techniques between different operating systems and integrating those findings into a cohesive platform would greatly enhance the versatility of tools like simpleVAP.

- Vulnerability Remediation Automation

Future research could explore automation of remediation steps in addition to detection. Developing a system that not only identifies vulnerabilities but also applies patches or reconfigures systems automatically (with admin approval) could save significant time and reduce the risk of human error. This would involve extensive research into automation frameworks and secure patching techniques that ensure minimal disruption to live systems.

- Human Factor in Vulnerability Management

Another important area of research is the impact of the human factor in vulnerability management. This includes how system administrators prioritize vulnerabilities, manage patching schedules, and interact with the tools they use. Research into improving the usability of vulnerability management platforms, combined with behavioral studies on how administrators react to security advisories, could result in the development of more user-centric and effective tools. [60]

8. References

[1] NIST, "Security Automation and Vulnerability Management," [Online]. Available: https://csrc.nist.gov/nist-cyber-history/automation-metrics/chapter.

[2] P.-D. O. A. S.-O. PAQUETTE, "Leveraging Machine Learning to Modernize Vulnerability Management," [Online]. Available: https://www.secureworks.com/blog/leveraging-ai-to-modernize-vulnerability-management-and-remediation.

[3] "chatgpt," [Online]. Available: https://chatgpt.com.

[4] "image," no. https://www.technologysolutions.net/wp-content/uploads/2023/09/pros-and-cons-scaled-2560x1280.jpeg.

[5] owasp.org, "A06:2021 – Vulnerable and Outdated Components," [Online]. Available: https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/.

[6] R. Timalsina, "Securing Linux Networks: A Checklist for IT Security Teams," [Online]. Available: https://tuxcare.com/blog/securing-linux-networks-a-checklist-for-it-security-teams/.

[7] MITRE, "Abuse Elevation Control Mechanism: Setuid and Setgid," [Online]. Available: https://attack.mitre.org/techniques/T1548/001/.

[8] MITRE, "Privilege Escalation," [Online]. Available: https://attack.mitre.org/tactics/TA0004/.

[9] OWASP, "Command Injection," [Online]. Available: https://owasp.org/www-community/attacks/Command_Injection.

[10] "Linux Security: The Role of sudo Command and the Significance of /etc/sudoers File," [Online]. Available: https://medium.com/@The_Anshuman/linux-security-the-role-of-sudo-command-and-the-significance-of-etc-sudoers-file-0a8402bd63f6.

[11] "Guidelines on Firewalls and Firewall Policy," no. https://doi.org/10.6028/NIST.SP.800-41r1 , 2009 .

[12] hackerone, "What Is Vulnerability Assessment? Benefits, Tools, and Process," [Online]. Available: https://www.hackerone.com/knowledge-center/what-vulnerability-assessment-benefits-tools-and-process.

[13] Greenbone, "Greenbone OpenVAS," [Online]. Available: https://www.openvas.org/.

[14] tenable, "Tenable Nessus," [Online]. Available: https://www.tenable.com/products/nessus.

[15] nmap, "Nmap Reference Guide," [Online]. Available: https://nmap.org/book/man.html.

[16] A. Williams, "Modernizing Vulnerability Management: Beyond Scanning to Continuous Exposure," [Online]. Available: https://www.cpomagazine.com/cyber-security/modernizing-vulnerability-management-beyond-scanning-to-continuous-exposure/.

[17] "What is Vulnerability Scanning?," [Online]. Available: https://jfrog.com/learn/devsecops/vulnerability-scanning/.

[18] google, "Supercharging security with generative AI," [Online]. Available: https://cloud.google.com/blog/products/identity-security/rsa-google-cloud-security-ai-workbench-generative-ai.

[19] "AI in Cybersecurity: How It’s Used + 8 Latest Developments," [Online]. Available: https://secureframe.com/blog/ai-in-cybersecurity.

[20] "Tenable Integrates Generative AI Capabilities Across Cybersecurity Platform With Launch of ExposureAI," [Online]. Available: https://www.tenable.com/press-releases/tenable-integrates-generative-ai-capabilities-across-cybersecurity-platform-with-exposure-ai.

[21] paramiko, "paramiko," [Online]. Available: https://www.paramiko.org/.

[22] "secrets — Generate secure random numbers for managing secrets," [Online]. Available: https://docs.python.org/3/library/secrets.html.

[23] "Flask," [Online]. Available: https://flask.palletsprojects.com/en/3.0.x/.

[24] "bootstrap 5.3," [Online]. Available: https://getbootstrap.com/docs/5.3/getting-started/introduction/.

[25] NIST, "Vulnerabilities," [Online]. Available: https://nvd.nist.gov/vuln.

[26] "CVE API," [Online]. Available: https://nvd.nist.gov/developers/vulnerabilities.

[27] "CVE-2020-15778 Detail," [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2020-15778.

[28] LinuxSecurity, "Monitoring Files with Special Permissions," [Online]. Available: https://linuxsecurity.com/howtos/learn-tips-and-tricks/monitoring-files-with-special-permissions.

[29] redcanary, "Setuid and Setgid," [Online]. Available: https://redcanary.com/threat-detection-report/techniques/setuid-setgid/.

[30] cisa.gov, "Apache Releases Security Update for HTTP Server," [Online]. Available: https://www.cisa.gov/news-events/alerts/2021/12/22/apache-releases-security-update-http-server.

[31] owasp, "INT01:2023 – Outdated Software," [Online]. Available: https://owasp.org/www-project-top-10-insider-threats/docs/2023/INT01_2023-Outdated_Software.

[32] websitesecuritystore.com, "Outdated Software Vulnerability: Meaning, Risks, and Updating Methods," [Online]. Available: https://websitesecuritystore.com/blog/security-risks-of-outdated-software/.

[33] specops, "TCP port 21 FTP vulnerabilities," [Online]. Available: https://specopssoft.com/blog/tcp-port-21-ftp-vulnerabilities/.

[34] securityboulevard, "9 SSH Key Management Best Practices You Need to Know," [Online]. Available: https://securityboulevard.com/2024/03/9-ssh-key-management-best-practices-you-need-to-know/.

[35] cyrisk, "Mitigation Instructions for OpenSSH," [Online]. Available: https://cyrisk.com/security/openssh.

[36] A. I. Nagori, "Why Is chmod -R 777 / Destructive?," [Online]. Available: https://www.baeldung.com/linux/fully-open-directories.

[37] algosec.com, "How to fix misconfigured firewalls (and prevent firewall breaches)," [Online]. Available: https://www.algosec.com/blog/five-common-firewall-configuration-mistakes-and-how-to-avoid-them.

[38] zenarmor, "What are the Top Firewall Vulnerabilities and Threats?," [Online]. Available: https://www.zenarmor.com/docs/network-security-tutorials/what-are-the-top-firewall-vulnerabilities-and-threats.

[39] cisa.gov, "Nikto," [Online]. Available: https://www.cisa.gov/resources-tools/services/nikto.

[40] "sqlmap," [Online]. Available: https://sqlmap.org/.

[41] "Flask APScheduler," [Online]. Available: https://viniciuschiele.github.io/flask-apscheduler/.

[42] sqlite, "What Is SQLite?," [Online]. Available: https://www.sqlite.org/.

[43] "bootstrap," [Online]. Available: https://getbootstrap.com/.

[44] ubuntu, "Cron How To," [Online]. Available: https://help.ubuntu.com/community/CronHowto.

[45] Mozilla, "Javascript," [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript.

[46] dev.to, "jQuery's Role in Modern Web Development: Beginnings, 2024, and Beyond," [Online]. Available: https://dev.to/wendyver/jquerys-role-in-modern-web-development-beginnings-2024-and-beyond-1223.

[47] "Shell Scripting Tutorial: How to Create Shell Script in Linux/Unix," [Online]. Available: https://www.guru99.com/introduction-to-shell-scripting.html.

[48] gbhackers.com, "7 Best Vulnerability Database Sources to Trace New Vulnerabilities," [Online]. Available: https://gbhackers.com/sources-trace-new-vulnerabilities/.

[49] M. Rana, "Data Serialization in Python: JSON vs. Pickle," [Online]. Available: https://www.inexture.com/data-serialization-in-python-json-vs-pickle/.

[50] "Network Automation with Python and Paramiko," [Online]. Available: https://dev.to/julianalmanzar/network-automation-with-python-and-paramiko-4iof.

[51] Debian, "Debian Operating System," [Online]. Available: https://www.debian.org/.

[52] Ubuntu, "Ubuntu Desktop," [Online]. Available: https://ubuntu.com/download/desktop.

[53] Ubuntu, "Ubuntu Server," [Online]. Available: https://ubuntu.com/download/server.

[54] Centos, "Centos Operating System," [Online]. Available: https://www.centos.org/download/.

[55] M. Linux, "MX Linux Operating System," [Online]. Available: https://mxlinux.org/.

[56] Kali, "Kali Linux Operating System," [Online]. Available: https://www.kali.org/.

[57] techtarget, "two-factor authentication (2FA)," [Online]. Available: https://www.techtarget.com/searchsecurity/definition/two-factor-authentication.

[58] G. R. S. A. W. e. a. Jabeen, "https://doi.org/10.1007/s10489-022-03350-5," *Machine learning techniques for software vulnerability prediction: a comparative study. Appl Intell 52, 17614–17635 (2022).*

[59] L. Stanham, "What is AI-Powered Behavioral Analysis in Cybersecurity," no. https://www.crowdstrike.com/cybersecurity-101/secops/ai-powered-behavioral-analysis/.

[60] oryxalign, "The Human Factor in the vulnerability management process," no. https://www.oryxalign.com/blog/vulnerability-management-process-human-factor.