ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
Τμήμα Πληροφορικής

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Ανίχνευση Ευπαθειών σε πολυπλατφορμικό περιβάλλον με χρήση Python

Proactive Vulnerability Management: A Python-Based Cross-Platform Vulnerability Assessment Tool

Μιχάλης Καλλιάφας, Σπυρίδων , Π16040

Επιβλέπων: Αναπληρωτής Καθηγητής Π. Κοτζανικολάου

Πειραιάς, *Αύγουστος 24*

Επιτελική Σύνοψη

Η παρούσα πτυχιακή εργασία παρουσιάζει την ανάπτυξη ενός εργαλείου εύρεσης ευπαθειών σε υπολογιστές και servers, γραμμένο με Python. Το εργαλείο μπορεί να τρέξει στα περισσότερα λειτουργικά συστήματα (Windows και Linux).

Σκοπός της εργασίας ήταν η δημιουργία ενός εύχρηστου και αποτελεσματικού εργαλείου για τον εντοπισμό πιθανών ευπαθειών σε υπολογιστικά συστήματα. Το εργαλείο αξιοποιεί τεχνικές σάρωσης και ανάλυσης για να εντοπίσει τυχόν αδυναμίες ασφαλείας, συμβάλλοντας στην προστασία των συστημάτων από κακόβουλες ενέργειες.

Η ανάπτυξη του εργαλείου βασίστηκε στις ακόλουθες μεθοδολογίες:

- Συλλογή πληροφοριών: Έρευνα και ανάλυση υπαρχόντων ευπαθειών και τεχνικών εκμετάλλευσης αυτών.
- Σχεδιασμός: Δημιουργία σχεδίου για την αρχιτεκτονική και τις λειτουργίες του εργαλείου.
- Ανάπτυξη: Χρήση της γλώσσας Python για την υλοποίηση του κώδικα.
- Έλεγχος: Διεξαγωγή δοκιμών για την επαλήθευση της λειτουργικότητας και της ακρίβειας του εργαλείου.

Η πτυχιακή εργασία οδήγησε στην ανάπτυξη ενός λειτουργικού εργαλείου εύρεσης ευπαθειών με τις ακόλουθες δυνατότητες:

- Εκτέλεση σε πολλαπλές πλατφόρμες: Το εργαλείο μπορεί να τρέξει σε Windows και Linux.
- Σάρωση ευπαθειών: Εκτελεί σάρωση του συστήματος για τυχόν γνωστές ή πιθανές ευπάθειες.
- Αναφορά ευρημάτων: Παρέχει λεπτομερή αναφορά με τα αποτελέσματα της σάρωσης.

Η υλοποίηση ενός χρήσιμου εργαλείου για τον εντοπισμό ευπαθειών σε υπολογιστικά συστήματα αποδείχθηκε. Το εργαλείο μπορεί να αξιοποιηθεί από διαχειριστές συστημάτων, εταιρείες πληροφορικής και χρήστες για την ενίσχυση της ασφάλειας των συστημάτων τους.

Abstract

This bachelor's thesis presents the development of a vulnerability discovery tool for computers and servers, written in Python. The tool can run on various operating systems, such as Windows and Linux.

The goal of this thesis was to create a user-friendly and effective tool for identifying potential vulnerabilities in computer systems. The tool utilizes scanning and analysis techniques to detect security weaknesses, contributing to the protection of systems from malicious activities.

The development of the tool was based on the following methodologies:

- Information Gathering**:** Research and analysis of existing vulnerabilities and exploitation techniques.
- Design**:** Creation of a design for the tool's architecture and functionalities.
- Development**:** Implementation of the tool's code using Python.
- Testing**:** Conducting tests to verify the tool's functionality and accuracy.

This thesis resulted in the development of a functional vulnerability scanning tool with the following capabilities:

- Cross-Platform**:** The tool can run on Windows and Linux.
- Vulnerability Scanning**:** Performs a scan of the system for known or possible vulnerabilities.
- Findings Report**:** Provides a detailed report with the scan results.

The development of a useful tool for identifying vulnerabilities in computer systems was successfully achieved. The tool can be utilized by system administrators, IT companies, and users to enhance the security of their systems.

Ευχαριστίες

Θα ήθελα να εκφράσω την ειλικρινή μου ευγνωμοσύνη σε όλους όσους συνέβαλαν, έστω και νοητικά, στην ολοκλήρωση αυτής της πτυχιακής εργασίας.

Πρώτα απ' όλα, το μεγαλύτερο ευχαριστώ προς τους πρώην συναδέλφους μου, οι οποίοι μου έδειξαν την αξία της ομαδικότητας, της συνεργασίας και της αλληλεγγύης. Μέσω της καθημερινής εμπειρίας με αυτούς, αντιλήφθηκα πώς η αφοσίωση στην υποστήριξη των συναδέλφων μπορεί να φέρει αμοιβαία οφέλη. Ιδιαίτερα θα ήθελα να ευχαριστήσω τον Α. Γαλήνα, ο οποίος μου έδειξε μια εντελώς νέα διάσταση στον κόσμο της ασφάλειας. Κατά τη διάρκεια της διετούς συνεργασίας μας, μου μετέδωσε γνώσεις που συνήθως χρειάζονται χρόνια για να αποκτηθούν. Μου απέδειξε επανειλημμένα πως στον τομέα της ασφάλειας, τίποτα δεν θεωρείται δεδομένο, και τόνισε τη σημασία της διαρκούς επαγγελματικής ανάπτυξης και της προώθησης της κοινής εξέλιξης με τους συναδέλφους μας.

Επίσης, οφείλω να εκφράσω την ευγνωμοσύνη μου προς την οικογένειά μου και τους φίλους μου, για την ατελείωτη υπομονή και την υποστήριξή τους σε όλη τη διάρκεια των σπουδών μου.

Τέλος θα ήθελα να προσθέσω πως, παρά το γεγονός ότι τα πτυχία και οι τίτλοι μπορούν να εξασφαλίσουν ευκαιρίες, η πραγματική επιτυχία και η επαγγελματική ανάπτυξη εξαρτώνται από προσωπικές αξίες, αφοσίωση, εργατικότητα και πάθος για το επιστημονικό αντικείμενο. Αυτή η διαπίστωση θα πρέπει να λειτουργεί ως σταθερή υπενθύμιση για όλους μας, ανεξάρτητα από τις επιδόσεις που έχουμε σημειώσει στα ακαδημαϊκά μας περιβάλλοντα.

Αύγουστος 24

Μιχάλης Καλλιάφας

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΠΙΝΑΚΑΣ ΕΙΚΟΝΩΝ

ΠΙΝΑΚΑΣ ΠΙΝΑΚΩΝ

Chapter 1

1 Introduction

Our digital world is a vast network, a web of connections that allows us to share information, access services, and stay connected. But just like any intricate system, this network isn't flawless. Weaknesses can hide within these connections, like cracks in a seemingly solid wall. These weaknesses can be exploited by malicious actors, potentially leading to stolen data, disrupted services, and even threats to critical infrastructure. This thesis dives into the world of vulnerability assessment, the process of identifying and understanding these hidden weaknesses. By exploring existing tools and techniques, we'll shine light on these weaknesses, allowing us to take steps to fortify our digital defences and keep our information and systems safe.

1.1 Description of the Problem under Study

The cyber security world always demands ongoing efforts to uncover and address weaknesses lurking within our computer systems and networks.

Traditional methods of vulnerability assessment, often manual or reliant on outdated tools, can be time-consuming, resource-intensive, and potentially ineffective. This thesis addresses this challenge by presenting the development of a Python-based vulnerability assessment tool specifically designed for Windows, Red Hat-based, and Debian-based operating systems. This tool addresses the critical need for efficient and vulnerability detection across these widely used platforms.

1.2 Purpose and Objectives

The primary purpose of this work is to develop a user-friendly and efficient vulnerability assessment tool using the power and versatility of Python.

This tool aims to empower security professionals and system administrators by:

1. Automating Vulnerability Scanning: The tool will automate the process of scanning systems for vulnerabilities across Windows, Red Hat, and Debian-based environments, making vulnerability assessments easier and reducing manual effort.
2. Focusing on Critical Vulnerabilities: The tool will prioritize the identification of active Common Vulnerabilities and Exposures (CVEs) within running system services. This focus ensures that security teams address the most critical and recently discovered vulnerabilities first.
3. Enhancing Security Posture: The tool will identify weak user passwords and analyze configurations for popular services like Apache, PostgreSQL, Nftables, Filezilla and windows Registry. By addressing these areas, the tool helps strengthen an organization's overall security posture.

4. Simplifying Remediation Efforts: The tool will present identified vulnerabilities in a clear and concise format, enabling users to prioritize remediation efforts based on severity and potential impact.

1.3 Conceptual Foundations

This section explores the essential vocabulary of vulnerability assessment in the "Nomenclature" subsection. Following that, the "Methodologies" subsection examines existing approaches used in the field. By understanding this foundation, we gain a deeper appreciation for the problem we aim to address and the context within which our tool operates.

1.3.1 Nomenclature

This section defines key terms used throughout this thesis to ensure clarity and understanding.

- **Apache:** Apache is one of the most widely used web server software in the world. Developed and maintained by the Apache Software Foundation, it is an open-source project available for free. Apache is known for its role in the initial growth of the World Wide Web and remains a popular choice for delivering web content. [4]
- **Brute Force:** A brute force attack is a hacking method that uses trial and error to crack passwords, login credentials, and encryption keys. It is a simple yet reliable tactic for gaining unauthorized access to individual accounts and organizations' systems and networks. The hacker tries multiple usernames and passwords, often using a computer to test a wide range of combinations, until they find the correct login information. [2]
- **Cache:** Pronounced cash, a special high-speed storage mechanism. It can be either a reserved section of main memory or an independent high-speed storage device. Two types of caching are commonly used in personal computers: memory caching and disk caching. [2]
- **Common Information Model (CIM):** Common Information Model (CIM) is the Distributed Management Task Force (DMTF) standard for describing the structure and behavior of managed resources such as storage, network, or software components. [5]
- **Common Vulnerabilities and Exposures (CVEs):** Publicly known security vulnerabilities assigned a unique identifier for tracking and mitigation. Your tool focuses on identifying active CVEs within running system services, prioritizing recently discovered and potentially exploitable vulnerabilities. [3]
- **Dictionary Attack:** A dictionary attack is a type of brute force attack where hackers try to guess a user's password to their online accounts by quickly running through a list of commonly used words, phrases, and number combinations. When a dictionary attack has successfully cracked a password, the hacker can then use this to gain access to things like bank accounts, social media profiles, and even password-protected files. This is when it can become a real problem for the attacker's victim. [2]
- **Indicator of Compromise (IOC):** It is a piece of forensic data that identifies potentially malicious activity on a system or network. IOCs help security professionals and IT

administrators detect data breaches, malware infections, or other threat activities early, thereby enabling faster response to prevent or mitigate potential damage. [6]

- **Nftables:** Nftables is a modern packet filtering and firewalling subsystem in the Linux kernel, which replaces the older iptables, ip6tables, arptables, and ebtables systems. It provides a single framework for IPv4, IPv6, ARP, and bridge packet filtering, with one unified command-line interface and one configuration syntax. This simplifies the management and deployment of firewall rules and policies across different types of networks and systems.
- **Operating System (OS):** Program that manages a computer's resources, especially the allocation of those resources among other programs. Typical resources include the central processing unit (CPU), computer memory, file storage, input/output (I/O) devices, and network connections. Management tasks include scheduling resource use to avoid conflicts and interference between programs. Unlike most programs, which complete a task and terminate, an operating system runs indefinitely and terminates only when the computer is turned off. [7]
- **Pluggable Authentication Modules (PAM):** PAM is a flexible and powerful suite of libraries that handle authentication tasks on Unix-like operating systems. It provides a system of modules that can be used to integrate multiple low-level authentication schemes into a high-level API, allowing for programs that rely on authentication to be written independently of the underlying authentication mechanism.
- **Port Scan:** A port scan is a series of messages sent by someone attempting to break into a computer to learn which computer network services, each associated with a "well-known" port number, the computer provides. Port scanning, a favorite approach of computer cracker, gives the assailant an idea where to probe for weaknesses. Essentially, a port scan consists of sending a message to each port, one at a time. The kind of response received indicates whether the port is used and can therefore be probed for weakness. [2]
- **PostgreSQL:** PostgreSQL, commonly known simply as Postgres, is a powerful, open-source object-relational database system. It extends the SQL language combined with many features that safely store and scale the most complicated data workloads. PostgreSQL is known for its robustness, scalability, and performance and is used by a wide range of applications and organizations.
- **Powershell:** PowerShell is a cross-platform task automation solution made up of a command-line shell, a scripting language, and a configuration management framework. PowerShell runs on Windows, Linux, and macOS. [11]
- **Registry:** The Windows Registry is a crucial part of Microsoft Windows operating systems, functioning as a database that stores low-level settings for both the operating system and applications that opt to use the Registry. The Registry contains information, settings, options, and other values for hardware and software installed on all versions of Microsoft Windows operating systems. Its purpose is to provide a centralized and standardized solution for managing configuration settings and to help control system and application operation more effectively. [9]

- **Vulnerability:** A weakness within a system, network, or application that can be exploited by malicious actors to gain unauthorized access (Confidentiality), manipulate data (Integrity), or disrupt operations (Availability). These three pillars, often referred to as the CIA triad, form the foundation of information security. [1]
- **Weak Configurations:** Insecure settings on services or software that can be exploited by attackers. Your tool will analyze configurations for popular services like Apache2, PostgreSQL, and iptables to identify potential security weaknesses.
- **Weak User Passwords:** Passwords that are easy to guess or crack, such as short passwords, dictionary words, or passwords reused across multiple accounts. Strong passwords are essential for maintaining system security.
- **Windows Management Instrumentation (WMI):** Windows Management Instrumentation (WMI) is a CIM server that implements the CIM standard on Windows. [10]

1.3.2 Methodologies

This research aims to identify and evaluate potential security vulnerabilities within computer systems. The core methodology employed is a vulnerability assessment, which involves a comprehensive examination of a system's configuration, services, and user accounts to identify exploitable weaknesses. To facilitate this vulnerability assessment, a custom-designed Python-based tool was developed. This tool makes the assessment process easier by automating various tasks, including:

Operating System Detection: The tool can identify the target system's operating system (Windows or Linux) to tailor the assessment procedures accordingly.

Configuration Scanning: The tool scans the target system's configuration files and settings, searching for potential vulnerabilities specific to the identified operating system.

Service Scanning: The tool identifies services running on the target system and compares them against a maintained database of Common Vulnerabilities and Exposures (CVEs). This allows the tool to pinpoint potential vulnerabilities associated with those services.

User Account Assessment: The tool analyzes user accounts and access control practices on the target system, identifying weaknesses in user management that could be exploited by attackers. This functionality is crucial for a vulnerability assessment as it helps identify potential security risks associated with user privileges and access controls.

1.4 Work Deliverables

In this section, we outline the deliverables associated with the completion of this thesis.

- Thesis Document
- report_template.xlsx
- requirements.txt
- wordlist
- main.py

- lib directory
    - Services Folder
        - CVEUpdater.py
        - ServiceScanController.py
        - LinuxServicesScanner.py
        - WinServicesScanner.py
    - Users directory
        - UserAssessmentController.py
        - LinuxUserAssessment.py
        - WinUserAssessment.py
    - Configurations directory
        - ConfigController.py
        - LinuxConfigScanner.py
        - WinConfigScanner.py
    - OSProber.py
    - Reporter.py

1.5 Thesis Outline

This section provides a roadmap for navigating the research to develop and evaluate a custom Python tool for vulnerability assessment. Focusing on the critical field of cybersecurity, the thesis explores existing vulnerability assessment practices and identifies a specific gap or limitation. Following this exploration, the development process of the Python tool is detailed, analysing design methodologies and the tool's functionalities. The effectiveness of the tool is then evaluated, with results and insights presented. Finally, the concluding chapter summarizes the research findings, highlighting the tool's contributions and outlining potential future directions for development. All the next chapters of this thesis are briefly explained below:

- **Chapter 2: Overview of the Field under Study:** This chapter provides an overview of the field of cyber security. It explores various types of assessments (e.g., network, system) and examines existing tools and methodologies.
- **Chapter 3: Methodology: Design and Development of the Python Tool**: This chapter dives into the technical aspects of the research. It analyses the methodologies for designing and developing the Python tool and a breakdown of the tool's functionalities and modules, along with their interactions, clarifies its capabilities. The chapter concludes by summarizing the key technical aspects and implementation of the Python tool.
- **Chapter 4: Results and Evaluation of the Vulnerability Assessment Tool:** Chapter 4 explores the effectiveness of the tool. The methodology employed for evaluation, such

as test cases and comparisons, is explained. The chapter presents the evaluation results, including the types and number of vulnerabilities identified, detection rates, and any false positives encountered. Following this, a thorough analysis evaluates the tool's strengths and weaknesses in terms of accuracy and resource utilization. The chapter concludes by discussing potential improvements and future development directions for the tool.

Chapter 2

2 Overview of the field under study

This chapter explores the crucial role of vulnerability assessment in today's computing environment. We'll dive into existing techniques and tools for identifying and mitigating security weaknesses across various operating systems.

2.1 The Threat Landscape

Modern computing is characterized by a wide gamma of operating systems powering everything from personal computers and mobile devices to servers and embedded systems. This diversity offers users a wide range of functionalities, but it also presents a complex challenge for security. Malicious actors constantly refine their tactics, exploiting vulnerabilities in operating systems to gain unauthorized access, steal sensitive data, disrupt operations, or deliver malware.

Some of the most common types of cyber-attacks that malicious actors use are:

1 Malware

Malware - or malicious software - is any program or code that is created with the intent to do harm to a computer, network or server. Malware is the most common type of cyberattack, mostly because this term encompasses many subsets such as ransomware, trojans, spyware, viruses, worms, keyloggers, bots, crypto jacking, and any other type of malware attack that leverages software in a malicious way. [2]

2 Denial-of-service (DoS) attacks

A Denial-of-Service (DoS) attack is a malicious, targeted attack that floods a network with false requests in order to disrupt business operations.

In a DoS attack, users are unable to perform routine and necessary tasks, such as accessing email, websites, online accounts or other resources that are operated by a compromised computer or network. While most DoS attacks do not result in lost data and are typically resolved without paying a ransom, they cost the organization time, money and other resources in order to restore critical business operations.

The difference between DoS and Distributed Denial of Service (DDoS) attacks has to do with the origin of the attack. DoS attacks originate from just one system while DDoS attacks are launched from multiple systems. DDoS attacks are faster and harder to block than DOS attacks because multiple systems must be identified and neutralized to halt the attack.[2]

3 Phishing

Phishing is a type of cyberattack that uses email, SMS, phone, social media, and social engineering techniques to entice a victim to share sensitive information — such as

passwords or account numbers — or to download a malicious file that will install viruses on their computer or phone.[2]

4 Spoofing

Spoofing is a technique through which a cybercriminal disguises themselves as a known or trusted source. In so doing, the adversary is able to engage with the target and access their systems or devices with the ultimate goal of stealing information, extorting money or installing malware or other harmful software on the device.

5 Identity-based attacks

Identity-driven attacks are extremely hard to detect. When a valid user's credentials have been compromised and an adversary is masquerading as that user, it is often very difficult to differentiate between the user's typical behavior and that of the hacker using traditional security measures and tools.

6 Code injection attacks

Code injection attacks consist of an attacker injecting malicious code into a vulnerable computer or network to change its course of action. There are multiple types of code injection attacks:

- SQL Injection
- Cross-Site Scripting (XSS)
- Malvertising
- Data Poisoning

7 Supply chain attacks

A supply chain attack is a type of cyberattack that targets a trusted third-party vendor who offers services or software vital to the supply chain. Software supply chain attacks inject malicious code into an application in order to infect all users of an app, while hardware supply chain attacks compromise physical components for the same purpose. Software supply chains are particularly vulnerable because modern software is not written from scratch: rather, it involves many off-the-shelf components, such as third-party APIs, open-source code and proprietary code from software vendors.

8 Social engineering attacks

Social engineering is a technique where attackers use psychological tactics to manipulate people into taking a desired action. Through the use of powerful motivators like love, money, fear, and status, attackers can gather sensitive information that they can later use to either extort the organization or leverage such information for a competitive advantage. [2]

2.2 Vulnerability Assessment: A Proactive Approach

In order to prevent the cyber-attacks that were previously described, security measures are essential. Vulnerability assessment (VA) plays a critical role in identifying, prioritizing, and mitigating security weaknesses within systems before they can be exploited. VA helps

organizations by proactively identifying vulnerabilities so that organizations can prioritize and patch them before attackers can exploit them.

For many organizations, compliance is a requirement too and many regulations require them to conduct regular vulnerability assessments. These vulnerability assessments can help organizations meet industry standards and data security mandates.

Other than that, optimizing security resources is a critical goal for most organizations. By conducting regular Vulnerability Assessments (VA) scans, organizations can gain a much clearer picture of their security posture. VA scans systematically identify and assess vulnerabilities within an IT infrastructure, allowing security teams to prioritize their efforts effectively. This prioritization is crucial, as resources for patching and remediation are often limited. By focusing on the most critical vulnerabilities first, organizations can significantly improve their overall security posture and mitigate the risk of breaches.

In other words, if a company wants to be safe from cyber-attacks, they should always perform a vulnerability assessment at least annually.

2.2.1 Vulnerability Assessment Types

There are several types of vulnerability assessment that are being used to extract different results and target different security layers. The most common ones are explained below.

2.2.1.1 Network-based assessment

Network vulnerability scanning is the process of inspecting and reporting potential vulnerabilities and security loopholes on a computer, network, web application or other device, including switches, routers, firewalls, and wireless access points.

Types of network vulnerability scanning

There are several types of vulnerability scanning and each comes with a specific purpose. Some common vulnerability scanning approaches include the following:

- **Unauthenticated scanning.** The tester performs the scan as an intruder would, without trusted access to the network. Such a scan reveals vulnerabilities that can be accessed without logging into the network.
- **Authenticated scanning.** The tester logs in as a network user, revealing the vulnerabilities that are accessible to a trusted user or an intruder that has gained access as a trusted user.
- **Host-based scanning.** This type of scanning inspects individual servers or computers to detect weaknesses in the operating system (OS), applications and configurations that are specific to each host.
- **Network-based scanning.** This scanning identifies vulnerabilities such as weak passwords in network devices, including routers, servers, firewalls, and switches.

Limited penetration testing is also conducted through network scanning without affecting the underlying system or network performance.

- **Web application scanning.** This targeted scanning of web apps aims to identify security flaws in their code, configurations, and authentication mechanisms, which helps mitigate common web-based security breaches.
- **Database scanning.** This type of scanning focuses on identifying flaws in database management systems.
- **Port scanning.** Port scanning entails making connection requests to network servers to locate open ports and then monitoring the responses to assess their activity. Port scan attacks are conducted by threat actors to get illegal access by locating open or underutilized ports.
- **Cloud vulnerability scanning.** Cloud-based vulnerability scanning is designed to pinpoint weaknesses that are specific to Cloud services and configurations.

[12,13,15,16]

Network scanning tool

Numerous network vulnerability scanning tools are available, ranging from open source to premium options. To make the best choice, organizations should carefully assess their specific requirements and vet a tool that aligns well with their needs.

Examples of network vulnerability scanning tools include the following:

- **Burp Suite.** This web vulnerability scanner conducts automated enterprise-wide scans to check for SQL injection, cross-site scripting, and other vulnerabilities besides being used for compliance and security auditing.
- **Falcon Spotlight.** This cloud-based next-generation antivirus tool from CrowdStrike protects networks and endpoints but also comes with a network threat-hunting module.
- **Nessus.** Nessus from Tenable is an industry-standard platform that scans for security flaws in hardware, software, OSes, cloud services and other network resources. Its enterprise edition transitioned from being an open-source tool in 1998 to a commercial offering in 2005.
- **Nmap.** Short for Network Mapper, this open-source tool offers network exploration, security audits and network discovery. Nmap is specifically designed to scan large networks rapidly, although it can also be used on smaller single-node networks.
- **Wireshark.** This open-source packet analyzer is used for network troubleshooting, analysis, software, and communications protocol development as well as for educational purposes. Wireshark lets users capture and interactively browse the contents of network traffic.
- **Wiz.** Wiz is an agentless vulnerability management tool for all cloud resources. It continuously assesses workloads to detect and prioritize vulnerabilities at scale and also extends vulnerability management into the CI/CD pipeline, scanning virtual machine

and container images before deployment to prevent vulnerabilities from reaching production.*Host-based Assessment*

Host-based security refers to the creation of a perimeter around each user workstation, server, or other network hosts, that interact with the IT network. It means installing firewalls and intrusion prevention systems and patching up software on a regular basis to avoid vulnerabilities. host-based security not only prevents a host from being infected but also ensures that if a host is infected, it doesn't spread the infection across the neighboring hosts. Host-based vulnerability scanning is the process of scanning a network host for security loopholes. A scan of this kind can reveal:

- The history of security patches in said host.
- Vulnerabilities incurred through outdated patches.
- The damage that can be caused by the detected vulnerabilities.
- The level of access a hacker can gain by infecting the said host.
- Possible ways of mitigating the situation.*Wireless network assessment*

Like a regular network scan, a Wireless Security Assessment identifies potential threats. However, it dives deeper specifically for Wi-Fi vulnerabilities. This assessment establishes a minimum-security standard for your wireless network, verifies it adheres to any security regulations, and gathers information on all connected devices' firmware versions to identify potential weaknesses. It even measures the reach of your Wi-Fi signal to ensure it stays within your intended area. In essence, a Wireless Security Assessment is a comprehensive examination of your Wi-Fi's security posture, ensuring it's strong enough to keep unwanted access and breaches at bay.

2.2.1.4 Application-based assessment

An application vulnerability assessment is a process of reviewing security weaknesses in software applications (Layer 7) including websites, mobile apps and APIs. It examines if the apps are susceptible to known vulnerabilities and assigns severity/criticality levels to those vulnerabilities, recommending remediation or mitigation if and whenever needed.

These assessments typically involve testing the application for common vulnerabilities, such as SQL injection, cross-site scripting (XSS), and other OWASP Top 10 vulnerabilities. Application vulnerability assessments can be performed using both automated and manual methods.

OWASP consistently compiles a list of the most critical application vulnerabilities, updated periodically. In its latest OWASP Top 10 risks 2021 ranking, the following vulnerabilities demand attention:

- A01:2021-Broken Access Control
- A02:2021-Cryptographic Failures
- A03:2021-Injection
- A04:2021-Insecure Design

- A05:2021-Security Misconfiguration
- A06:2021-Vulnerable and Outdated Components
- A07:2021-Identification and Authentication Failures
- A08:2021-Software and Data Integrity Failures
- A09:2021-Security Logging and Monitoring Failures
- A10:2021-Server-Side Request Forgery

[17]

2.2.1.5 API-Based Vulnerability Assessment

API vulnerability assessment is conducted to identify and mitigate potential security risks in APIs. This process identifies vulnerabilities and weaknesses in the API's design, implementation, and deployment. The goal is to ensure that the API is secure, reliable, and resilient to attacks.

The following OWASP API Top 10 vulnerabilities require specific attention in vulnerability assessment process to ensure the security and integrity of API interactions:

- API1:2023 Broken Object Level Authorization
- API2:2023 Broken Authentication
- API3:2023 Broken Object Property Level Authorization
- API4:2023 Unrestricted Resource Consumption
- API5:2023 Broken Function Level Authorization (BFLA)
- API6:2023 Unrestricted Access to Sensitive Business Flows
- API7:2023 Server-Side Request Forgery (SSRF)
- API8:2023 Security Misconfiguration
- API9:2023 Improper Inventory Management
- API10:2023 Unsafe Consumption of APIs

2.3 Existing Vulnerability Assessment Tools: A Comparative Analysis

This comparative analysis section delves into the functionalities offered by a range of existing VA tools, with a focus on their suitability for different security areas (which we'll explore further in a subsequent section). We'll dissect their core strengths, such as the types of vulnerabilities they can detect (be it network, application, or endpoint vulnerabilities), the depth of analysis they provide, and their ease of use. However, no tool is perfect, and we'll also explore limitations, including the potential for false positives, specific system compatibility, and cost considerations. This analysis will encompass both open-source and commercially available tools, highlighting the advantages and disadvantages of each category. Through this examination, we aim to empower security professionals with the knowledge required to make informed decisions. By understanding the unique strengths and limitations of various VA tools, professionals can select the solution that best aligns with their organization's specific security posture, resource constraints, and budget.

2.3.1 Tools for Specific Security Areas

VA tools/solutions are designed to tackle different tasks and to provide a different view of each company's security posture. Some of these tools are summarized below:

1. Nessus

Nessus is a popular network vulnerability assessment tool. It provides features designed to help you identify, assess, and rectify security vulnerabilities. Nessus offers tools for vulnerability scanning, configuration auditing, asset profiling, and more. It is known for its speed, accuracy, and thoroughness in scanning networks. Nessus provides a comprehensive vulnerability database, frequent updates, and easy-to-use interface. Its ability to scan a wide range of devices, including network devices, databases, and web servers, makes it a versatile tool suitable for organizations of all sizes. Moreover, its extensive reporting capabilities enable you to understand your vulnerabilities in-depth and plan your mitigation strategies accordingly. [22]

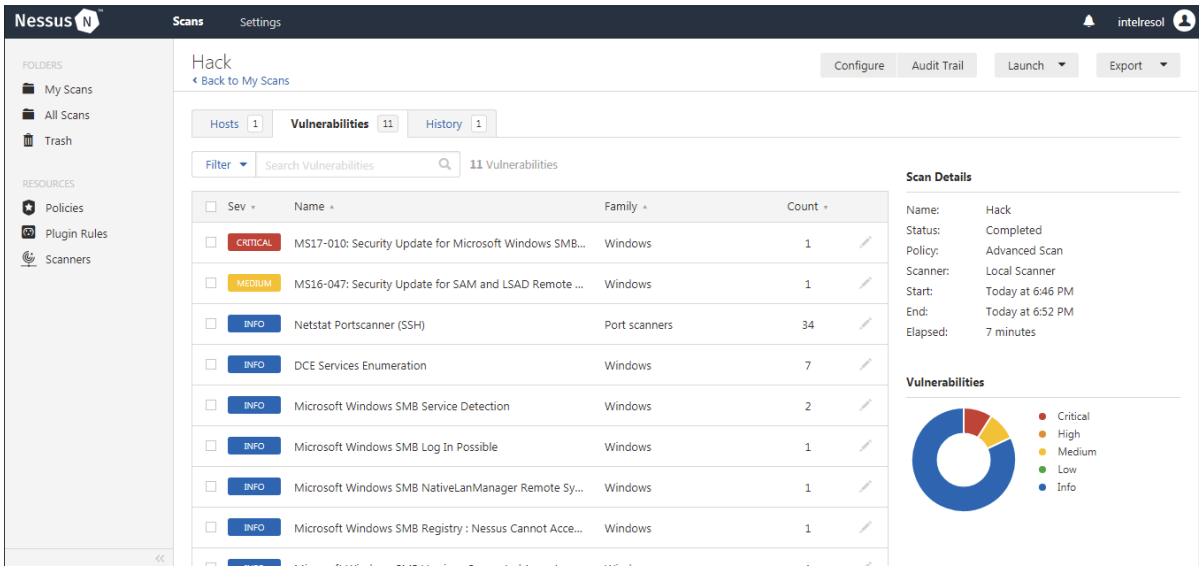


Figure 1: Nessus Interface

2. Qualys

Qualys is a cloud-based network vulnerability assessment tool. Its primary function is to identify vulnerabilities in your network and provide recommendations for their remediation. Qualys offers speed, scalability, and accuracy in its vulnerability scans. Qualys's cloud-based nature allows it to perform scans without the need for any hardware or software installations. This scalability makes it suitable for both small businesses and large enterprises. Furthermore, its real-time threat updates ensure that you are always aware of the latest vulnerabilities. [22]

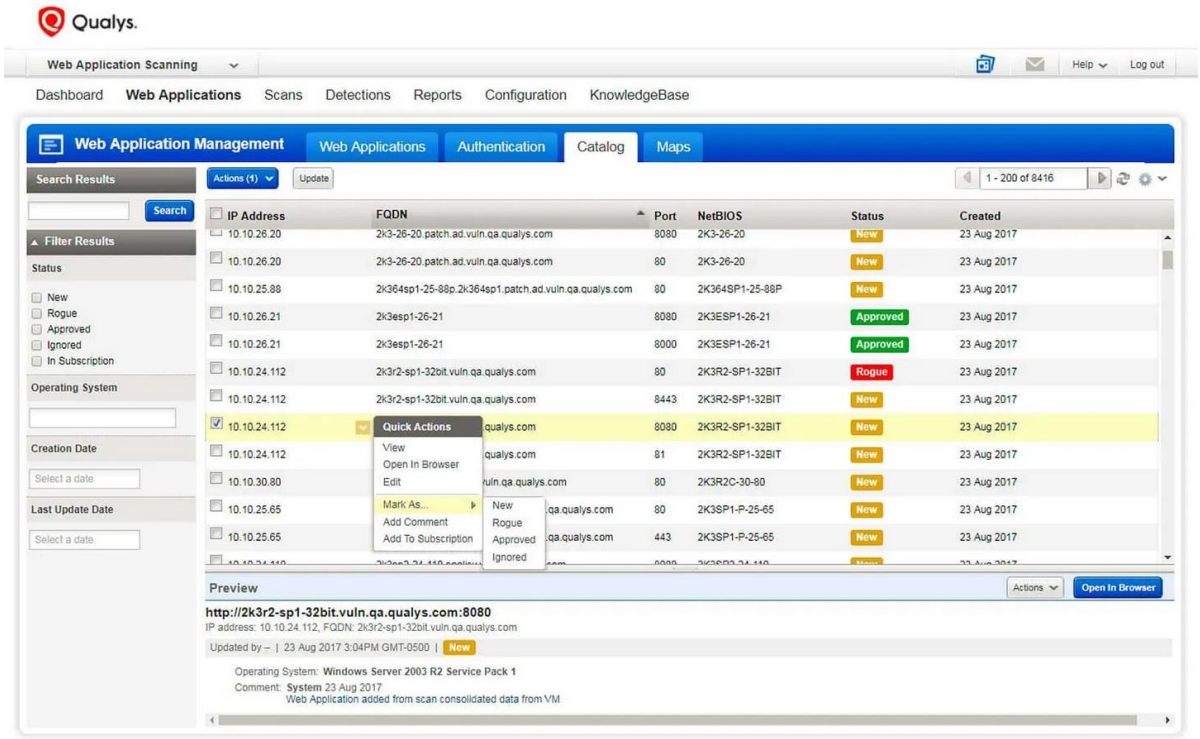


Figure 2: Qualys Interface

3. OpenVAS

OpenVAS is a free and open-source application vulnerability assessment tool. It offers a suite of tools for vulnerability scanning, management, and reporting. OpenVAS's strength lies in its vibrant community of users and developers, who continually work on improving the tool and keeping it updated with the latest threat intelligence. Its range of plugins allows for customization according to your specific needs. Additionally, its detailed reporting capabilities help you understand your vulnerabilities and devise effective mitigation strategies. [22]

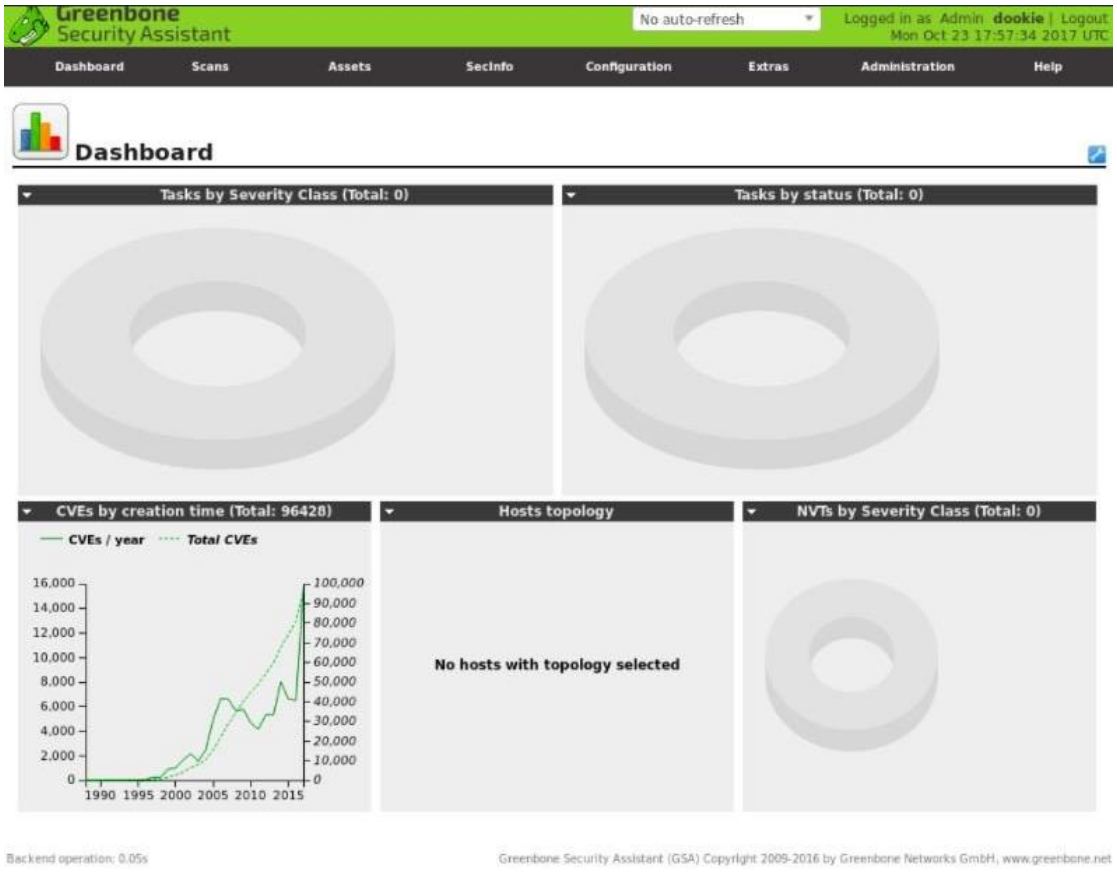


Figure 3: OpenVAS Interface

4. Nexpose

Nexpose, developed by Rapid7, is a network vulnerability management tool that offers real-time insights into your security posture. It is known for its dynamic risk scoring, which evaluates vulnerabilities in the context of their potential impact on your business. Dynamic risk scoring helps you prioritize your efforts based on the potential damage a vulnerability could cause. This context-based approach allows for more effective vulnerability management. Moreover, its integration capabilities with other security tools further enhance its effectiveness as a comprehensive security solution. [22]

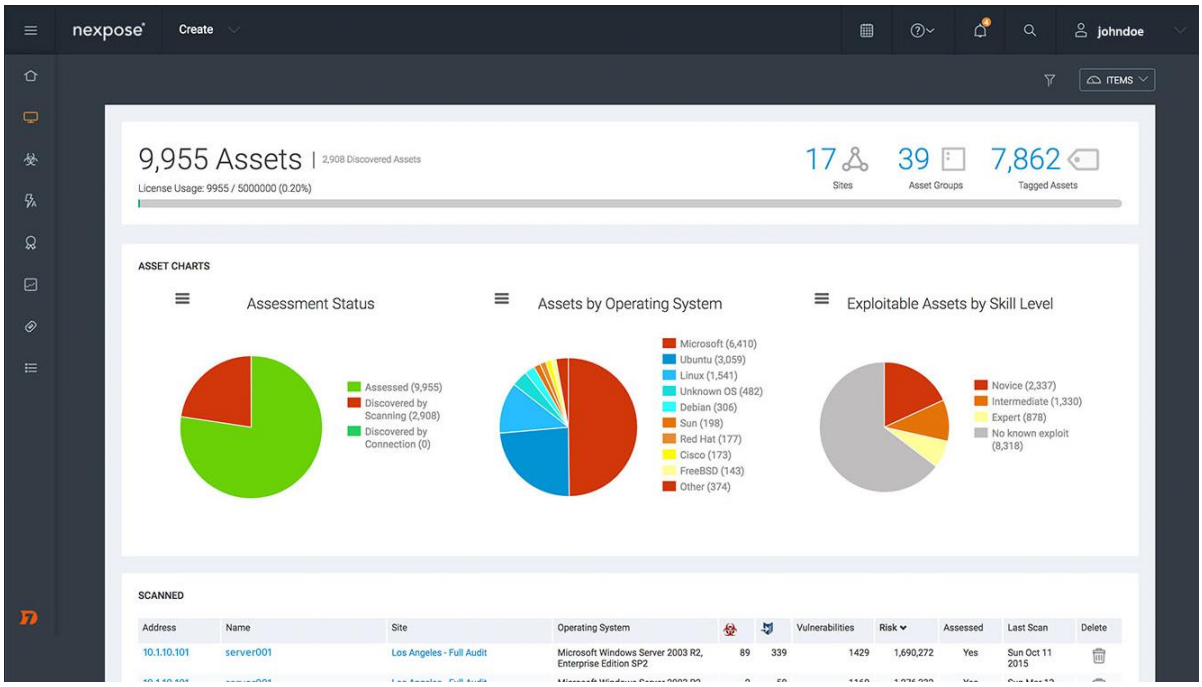


Figure 4: Nexpose Interface

5. ZAP

Zed Attack Proxy (ZAP) scores the highest overall for all open-source vulnerability scanners and provides the highest rated open-source value and ease of use of the tools tested. Pre-installed on Kali Linux, ZAP places itself between the tester's browser and the web application to intercept requests to act as a "proxy". This tests applications by modifying contents, forwarding packets, and other user behavior simulations in a comprehensive and robust fashion. [23]

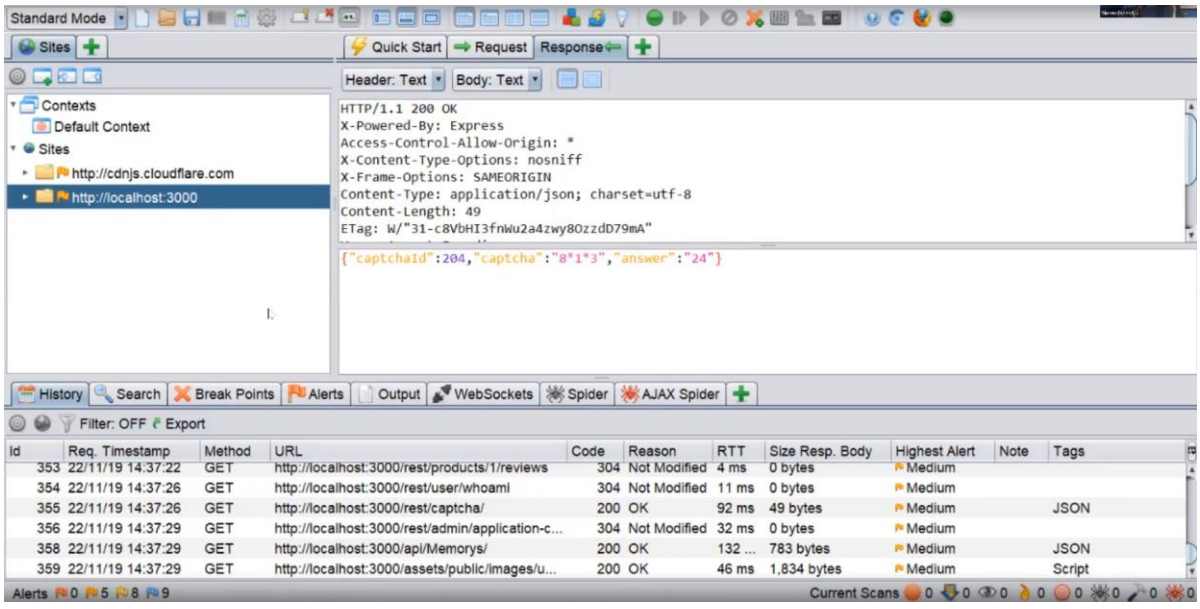


Figure 5: ZAP interface.

6. CloudSploit

Aqua acquired and continues to maintain the open-sourced cloud-infrastructure scanning engine CloudSploit so that users can download, modify, and enjoy the benefits of the specialty tool. CloudSploit scans can be performed on-demand or configured to run continuously and feed alerts to security and DevOps teams. This tool examines cloud and container deployments not only for known vulnerabilities but also for common misconfiguration issues. [23]

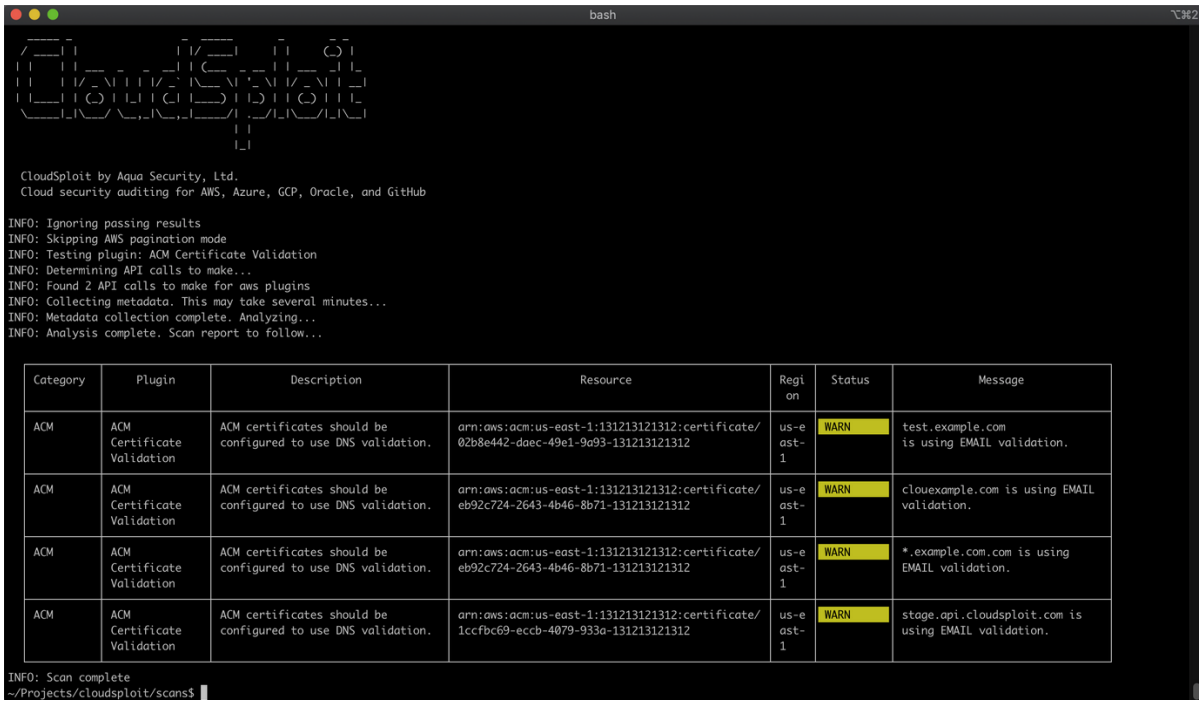


Figure 6: CloudSploit interface.

2.3.2 Open-Source vs. Commercial Tools

This section will compare and contrast open-source and commercial tools in this domain. Understanding the strengths and weaknesses of each approach will help us make an informed decision when selecting the tool that best suits our specific needs and resource limitations.

2.3.2.1 Open-Source Tools

Open-source software, often referred to as "free and open-source software" (FOSS), is characterized by its publicly accessible source code. This allows developers to freely modify, distribute, and improve the software, creating a vibrant community of collaboration and innovation. Some of the key features of FOSS are:

- **Customization:** Open-source grants you the freedom to tailor the software to your exact needs and preferences. Whether you're building a custom web application or experimenting with a new data analysis tool, the ability to modify the code unlocks incredible flexibility.
- **Cost-Effectiveness**: In an era of tight budgets, the free nature of open-source software makes it a compelling option for independent developers and startups. This can

significantly reduce development costs, allowing you to focus on building your project without financial constraints.

- **Community Support**: Open source thrives on a collaborative spirit. A vast network of developers actively contributes to and supports open-source projects, providing valuable resources and assistance through online forums, documentation, and troubleshooting guides. This collective knowledge base can be invaluable when encountering challenges or seeking new insights.
- **Transparency and Security**: With the source code readily available, open source fosters a high degree of transparency. You can scrutinize the code's functionality and security, ensuring it aligns with your ethical and security standards. This transparency also allows for rapid identification and patching of vulnerabilities, contributing to the overall security of the software.

2.3.2.2 Commercial Tools

While open source offers unique advantages, commercial software provides a different set of benefits, particularly suited for specific development needs.

- **Dedicated Support and Maintenance:** Commercial software vendors typically offer dedicated support teams and regular updates, ensuring smooth operation and addressing issues promptly. This can be important for high-stakes projects or long-term development cycles where stability and ongoing maintenance are paramount.
- **Integrated Solutions:** Many commercial software solutions provide comprehensive and pre-configured environments, streamlining your development process by eliminating the need to manage multiple disparate tools. This can significantly improve efficiency and reduce time spent on configuration and setup.
- **User-Friendly Interfaces:** Commercial software is often designed with user-friendliness in mind, featuring intuitive interfaces and well-documented features. This can be a major benefit for remote developers, as it minimizes the learning curve and allows you to focus on your core development tasks.
- **Clear Licensing and Compliance:** Purchasing commercial software grants you a clear license that outlines your rights and obligations, ensuring legal compliance and intellectual property protection. This is particularly important for projects with strict licensing requirements or those involving sensitive data.

2.3.2.3 What is better?

Ultimately, the choice between open-source vs commercial software hinges on your specific circumstances and priorities. Open source might be ideal for experienced professionals that have time to customize the tool based on their needs, while commercial software can offer a more plug-n-play solution.

The budget of each company is a vital part for its life expectancy too. Open source offers a significant cost advantage, while commercial software comes with upfront licensing fees and potentially ongoing support costs. Weigh your budget constraints against the value proposition of each option.

Another factor to consider is technical expertise**.** If you possess the technical skills and desire for customization, open source can be empowering. But if you value dedicated support, commercial software is a better fit.

The ideal choice between open-source vs commercial software will depend on your specific project needs.

2.3.3 Limitations of vulnerability scanners

Vulnerability scanning involves using either a software or hardware-based scanner to locate soft spots in your code or in your infrastructure that can be exploited by known attack vectors. Soft spots are typically a result of unsanitized code that permits illegal inputs, misconfigurations that can be used to gain unwanted access or unpatched software.

Scans involve periodic pen-tests and code reviews to uncover weak spots, followed by code updates and patches to remove vulnerabilities. The targets are rescanned afterward to ensure that vulnerabilities have been weeded out. This review and patch cycle should be conducted after updates and anytime new attack vectors that could endanger your infrastructure are discovered.

As a whole, vulnerability scanning comes with several operational issues. For one, new vulnerabilities continually pop up, making scanning a frequent and resource-intensive process. Moreover, complete code sanitization is rarely achieved, as the body code usually exists in a continual state of change. This is on top of the fact that it’s impossible to predict all attack scenarios.

Lastly, vulnerability scanning cannot help with rapid responses to newly uncovered (zero-day) threats. This is crucial, as most exploits take place soon after new vulnerabilities are made public. Response time to such threats becomes a key component of any vulnerability management strategy—one that can’t be addressed by a prolonged cycle of reviews and sanitizations.

2.4 Python: Open-source Cross-Platform Vulnerability Assessment

Figure 7: Python logo

Python's versatility and rich ecosystem of security libraries make it well-suited for developing VA tools. It offers libraries like scapy for network analysis, nmap for network scanning, and behave for security testing. These libraries provide pre-built functionalities for vulnerability detection and analysis. Even if you don’t decide to use the pre-built functionalities, developing new processes from the ground-up is easier than any other programming language.

On top of that, the vast open-source community offers numerous security libraries and frameworks readily available for integration. This offers an easy way to implement solutions that might benefit every member in the community and in a way that best suit your needs.

Lastly, python's clear and concise syntax makes code easier to write, understand, and maintain. This is crucial for security tools, which require ongoing updates and modifications as new vulnerabilities emerge.

By understanding existing vulnerability assessment techniques and tools, we can identify potential gaps and opportunities for improvement. The next chapter will delve deeper into the design and functionalities of our Python-based vulnerability assessment tool, focusing on its capabilities for identifying and analyzing vulnerabilities across Windows, Red Hat-based, and Debian-based environments.

Chapter 3

3 Methodology: Design and Development of the Python Tool

The crux of this thesis lies in the development of a Python-based vulnerability assessment tool designed for cross-platform environments. This section will dive into the inner workings of this tool. We'll begin by exploring the overall design and architecture of the tool, outlining its core components and functionalities (Section 3.1). Next, we'll examine the specific vulnerability detection techniques employed by the tool, focusing on its capabilities for identifying vulnerabilities across various operating systems (Section 3.2). Finally, Section 3.3 will shed light on the implementation process, discussing the programming languages and frameworks used to bring this tool to life.

3.1 Tool Design and Architecture

This subsection dives into the blueprint of our Python-based vulnerability assessment tool. We'll dissect its overall design, outlining the core components and their functionalities. Here's what we'll explore:

Modular Design: We'll discuss the tool's modular architecture, explaining how it's broken down into distinct modules that perform specific tasks. This modular design promotes code reusability, maintainability, and easier integration of future functionalities.

Core Components: We'll dive into the essential components that make up the tool. This includes a service assessment module, a user assessment module, a configuration assessment module and small reporting module for presenting results.

Data Flow and Interaction: We'll describe how these modules interact with each other and with the target system. This explanation will illustrate how user input triggers the scanning engine and how vulnerabilities are identified and reported.

Scalability and Extensibility: We'll explore the tool's design considerations for future growth. This might involve discussing how the architecture allows for adding new vulnerability detection modules or supporting additional operating systems without major code overhauls.

3.1.1 Design and components

Our Python-based vulnerability assessment tool adopts a modular design for enhanced code reusability, maintainability, and future expandability. This section dives into its core components and their functionalities:

Central Orchestrator: main.py

The **main.py** script acts as the central control unit, coordinating the entire vulnerability assessment process. It facilitates user interaction, initiates scans based on user input, and aggregates results for a clear presentation.

Platform-Specific Configuration Scanning

The tool employs two key modules for configuration scanning, each tailored to a specific operating system:

- **WinConfigScanner.py**: This module focuses on Windows system configurations. It parses relevant configuration files and system settings to identify potential vulnerabilities that attackers could exploit.
- **LinuxConfigScanner.py**: This module mirrors the functionality of WinConfigScanner.py but caters to the specific configuration files and settings found on Linux systems.

These modules work in conjunction with a coordinator script, **ConfigController.py**. This script acts as a hub and triggers the appropriate platform-specific scanner module (either WinConfigScanner.py or LinuxConfigScanner.py). On top of that this script calls the corresponded function for the targeted configurations. Some configurations that the script checks are Apache2, PostgreSQL and Iptables configuration files. This ensures efficient and targeted configuration scanning based on the target environment and service.

Service Scanning for Vulnerabilities

Similar to the configuration scanning modules, the tool utilizes two platform-specific scripts for service scanning:

- **WinServicesScanner.py**: This module scans for vulnerabilities in services running on Windows systems. It leverages libraries or system calls to identify active services and compares them against a database of known Common Vulnerabilities and Exposures (CVEs) retrieved and maintained by CVEUpdater.py.
- **LinuxServicesScanner.py**: This module functions similarly to its Windows counterpart, focusing on vulnerabilities in running services on Linux systems. It employs appropriate libraries or system calls to gather service information and performs CVE checks against the updated database managed by CVEUpdater.py.

These service scanners rely on a coordinator script, **ServiceScanController.py**. This script acts as a hub/controller, which gets called and initiates the corresponding service scan module (either WinServicesScanner.py or LinuxServicesScanner.py). This ensures the tool performs vulnerability assessments aligned with the target system's operating system.

CVE Database Management: CVEUpdater.py

CVEUpdater.py plays a vital role in maintaining an up-to-date vulnerability database. It retrieves CVE data for the running services from NIST and stores it locally. This ensures the tool leverages the latest CVE information for accurate vulnerability identification during service scans. The local file acts as a cache for the local services and has a 7-day update interval, meaning that if the local data are older than a week then the file will be updated with the new data.

User Assessment Module

The tool offers a user assessment functionality to gain a better understanding of potential security weaknesses.

This functionality is coordinated by **UserAssessmentController.py**, which manages platform-specific user assessment scripts. **WinUserAssessment.py** handles user account assessments on Windows systems, focusing on aspects like identifying privileged accounts and analyzing password complexity. Similarly, **LinuxUserAssessment.py** performs analogous user account assessments on Linux systems.

Operating System Detection: OSProber.py

OSProber.py is crucial for directing the tool's functionalities. This script identifies the operating system of the system where the tool is running. This information is vital, as it dictates which platform-specific scanning modules are executed. By identifying the target system's OS, the tool ensures efficient and accurate vulnerability assessments.

Report Generation: Reporter.py

The **Reporter.py** script is designed to generate reports in both PDF and Excel (XLSX) formats based on the results produced by the tool. This script automates the process of compiling, formatting, and exporting data, making it easier to share and review the findings.

In the table below we can see a summarization of the scripts and their functionalities.

Table 1: Scripts and their roles

Script Name	Role
main.py	Central control unit, orchestrates execution, handles user interaction, initiates scans, aggregates results
CVEUpdater.py	Retrieves and updates the local CVE database used for vulnerability identification. Creates a cache file that contains the local services and their CVEs.
OSProber.py	Identifies the operating system (and its distribution if available) of the system where the tool is running
ConfigurationController.py	Coordinates platform-specific configuration scanners (WinConfigScanner.py & LinuxConfigScanner.py)
WinConfigScanner.py	Scans Windows system configurations for potential vulnerabilities
LinuxConfigScanner.py	Scans Linux system configurations for potential vulnerabilities
ServiceScanController.py	Coordinates platform-specific service scanners (WinServicesScanner.py & LinuxServicesScanner.py)
WinServicesScanner.py	Scans for active vulnerabilities in services running on Windows systems

LinuxServicesScanner.py	Scans for active vulnerabilities in services running on Linux systems
UserAssessmentController.py	Coordinates platform-specific user assessments (WinConfigScanner.py & LinuxConfigScanner.py)
WinUserAssessment.py	Performs user account assessments on Windows systems (e.g., identifying privileged accounts, analyzing password complexity)
LinuxUserAssessment.py	Performs user account assessments on Linux systems (e.g., identifying privileged accounts, analyzing password complexity)
Reporter.py	Creates an xlsx and a pdf report with all the results.

3.1.2 Data Flow and Interaction

The modular design of our tool facilitates a seamless flow of data between components, orchestrating a comprehensive vulnerability assessment process. Let's delve into this interaction.

Firstly, the user initiates the assessment by interacting with the central main.py script. The interaction starts by specifying the type of assessment desired (configuration scan, service scan, user assessment or a combination of them).

Next, OSProber.py plays a crucial role by identifying the target system's operating system (Windows or Linux) and its distribution (Debian, RedHat etc.). This information is critical, as it dictates the subsequent course of action. Based on user input and the identified OS, main.py triggers the appropriate scanning modules.

For configuration scanning, ConfigController.py receives the user's selection and the OS information. It then acts as a conductor, launching the relevant platform-specific scanner. WinConfigScanner.py handles Windows systems, parsing relevant configuration files and system settings to identify potential vulnerabilities. Similarly, LinuxConfigScanner.py tailors its approach to the specific configuration files and settings found on Linux systems.

Service scanning follows a similar path. ServiceScanController.py receives user’s selections and OS information before initiating the appropriate service scanner. WinServicesScanner.py focuses on Windows systems, leveraging libraries and system calls to identify active services and compare them against the CVE database maintained by CVEUpdater.py. Likewise, LinuxServicesScanner.py functions on Linux systems, employing appropriate libraries and system calls to gather service information and perform CVE checks against the updated database.

CVEUpdater.py plays a vital role behind the scenes, keeping the CVE database fresh. It retrieves CVE data based on the running services from NIST and stores it locally, ensuring the tool identifies vulnerabilities based on the latest information.

For the User assessment procedure, we follow the same idea. For Windows systems, WinUserAssessment.py takes charge. For Linux systems, LinuxUserAssessment.py handles the user assessment tasks. These scripts act as platform-specific counterparts and are called by the UserAssessmentController.py script. Both WinUserAssessment.py and LinuxUserAssessment.py interact with their respective operating systems to gather information about user accounts. This involves querying the local system to identify accounts with login capabilities. The scripts then analyze these accounts, assessing the complexity of their passwords and their group memberships.

Finally, main.py collects the results from all scanning modules. It then processes and organizes this data, with the help of the Reporter.py, into a comprehensive report, presented to the user in terminal, in a pdf and in an xlsx file. This report details identified vulnerabilities, potential risks associated with those vulnerabilities, and recommendations for remediation.

This coordinated data flow, facilitated by the modular design, ensures a user-friendly and informative vulnerability assessment experience.

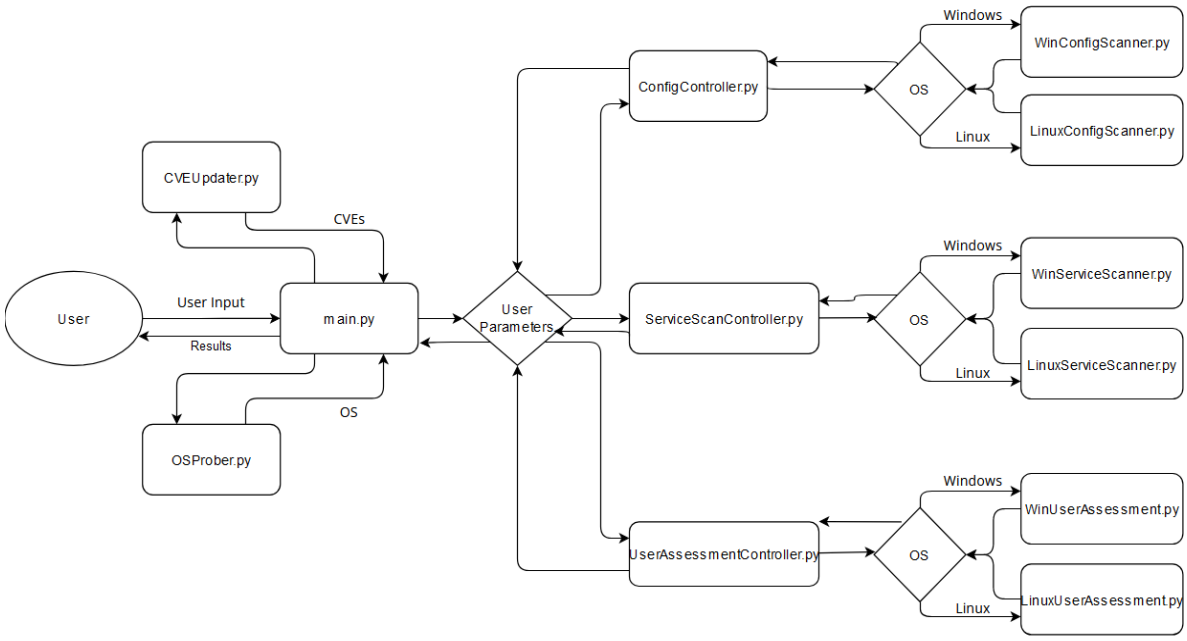


Figure 8: DataFlow diagram

3.2 Vulnerability Detection Techniques Employed

The tool leverages various techniques and attacks to identify potential vulnerabilities within a target system. These techniques work in tandem with the data flow described earlier to provide

a comprehensive vulnerability assessment. For the most part, the scanning analysis is based on string parsing, string comparisons and static code analysis, but let's dig deeper into the techniques:

Static Code Analysis (SCA)
Static Code Analysis principles are employed during configuration and service scanning. By examining configuration files and system settings specific to the target operating system (Windows or Linux), the tool can detect potential vulnerabilities arising from insecure configurations or the lack of configuration hardening. For example, it might identify weak access controls or insecure service configurations.

LSASS Dumping
The Local Security Authority Subsystem Service (LSASS.exe) is a critical process in Microsoft Windows operating systems. It acts as the gatekeeper, enforcing security policies and ensuring only authorized users gain access. LSASS verifies user logins, manages password changes, and creates access tokens that function like digital keys for accessing resources. It also keeps a record of security-related events in the Windows Security Log. However, due to the sensitive information it stores, including hashed passwords and access tokens, LSASS becomes a prime target for attackers. By dumping the contents of LSASS memory, attackers can potentially steal this information and bypass normal login security, gaining unauthorized access to user accounts and systems. This technique is known as LSASS dumping. [24,25]

User/Group enumeration through WMI/CIM queries
WMI (Windows Management Instrumentation) acts as a bridge between applications and the inner workings of a Windows system. Imagine it as a universal translator that allows programs to request and interpret information about various aspects of the system, including hardware components, software applications, and security configurations.
For user and group information, WMI provides specific classes that function like data blueprints. One such class is **Win32_UserAccount**, which holds details about user accounts like usernames, full names, and account status. Another class, **Win32_LocalGroup**, stores information about local groups on the system, including their names, descriptions, and member users.
By crafting WMI queries that target these classes, administrators can retrieve specific user and group information. For instance, a query might target the Win32_UserAccount class to retrieve a list of all usernames on the system. Another query could target the Win32_LocalGroup class in conjunction with the Win32_GroupUser association class to determine which users belong to a specific local group.
By taking advantage of this functionality, we can retrieve information about user accounts and their privileges. [26,28]

Dictionary Attack
A dictionary attack is a type of brute force attack where we try to guess a user's password to their accounts by quickly running through a list of commonly used words, phrases, and number combinations. When a dictionary attack has successfully cracked a password, the hacker can then use this to gain access to things like bank accounts, social media profiles, and even

password-protected files. This is when it can become a real problem for the attacker's victim. In this python tool we employ the same attack in order to find the user's password. If the password is found, then it means that it's weak and we recommend a password change.

Registry Scanning

A critical technique that involves scanning the Windows registry to identify potential misconfigurations. The Windows registry is a hierarchical database that stores low-level settings for the operating system and installed applications. Misconfigurations within the registry can lead to security vulnerabilities, system instability, or compliance issues. Our tool systematically navigates through registry hives and keys, extracting values and settings that are known to affect system security. By comparing these settings against best practice baselines and security benchmarks, the tool identifies deviations that could indicate misconfigurations.

3.3 Tool Implementation

This section dives into the technical details of constructing the Python vulnerability assessment tool.

3.3.1 Programming Language and Libraries

Python was chosen as the primary development language due to several factors. First and foremost, Python's clean and concise syntax promotes readability. This makes the codebase easier to understand not only for the developer but also for potential collaborators or future maintainers. This clarity simplifies future modifications to the tool.

Beyond readability, Python provides a vast ecosystem of well-established libraries that offer functionalities directly applicable to vulnerability assessment. These libraries make the development easier and empower the tool with specific capabilities. For instance, the regex library provides powerful parsing functionalities. The tool uses a few libraries for the main reason that Python provides all the core functionalities using its built-in methods. Functionalities like reading files, stripping string, searching sentences etc, are some of the easiest things to do with python compared to other programming languages. In the end, by leveraging this, we can focus our development efforts on crafting the core logic of the tool rather than reinventing the wheel for essential functionalities.

Table 2: Libraries Used

Library Name	Description
re (Regular Expressions)	This built-in library provides powerful functionalities for pattern matching and text manipulation. It's instrumental in parsing configuration files, log files, and other text-based data for vulnerability identification.

pprint (Pretty Printing)	This built-in library helps in formatting and presenting complex data structures (like dictionaries or nested lists) in a more human-readable format. This can be valuable for debugging purposes or for generating user-friendly reports.
os (Operating System)	This built-in library offers functionalities to interact with the operating system. It can be used for tasks like file system operations (reading/writing files), path manipulation, and potentially querying system information relevant to vulnerability assessment.
json (JavaScript Object Notation)	This built-in library facilitates working with JSON data format, which is commonly used for data exchange between applications. It can be used for parsing vulnerability databases or configuration files stored in JSON format.
datetime (Date and Time)	This built-in library provides functionalities for working with dates, times, and timedeltas. It can be crucial for tasks like recording timestamps for scans, scheduling scans, or manipulating timestamps associated with vulnerabilities.
time (Time Access)	This built-in library offers functionalities for measuring elapsed time, scheduling tasks, and controlling the program flow based on timing requirements. This can be relevant for measuring scan duration or implementing timeouts
subprocess (Subprocesses)	This built-in library allows you to execute external programs or commands from within your Python script. This might be useful for situations where you need to leverage external tools for specific tasks during vulnerability assessment.
platform (System Platform Information)	This built-in library provides access to information about the system platform (operating system, architecture, version). This information can be used to tailor vulnerability assessments based on the specific platform or identify platform-specific vulnerabilities.

requests (HTTP Requests)	This external library simplifies making HTTP requests to web servers. It can be used for tasks like fetching vulnerability data from online databases or interacting with web-based services for vulnerability checks.
packaging (Packaging Utilities)	[External Library] The packaging library has a module called Versions which provides functionalities for parsing and comparing version strings. This can be particularly useful for comparing the versions of local services with known vulnerable versions listed in CVE databases.
progressbar (Progress Bars)	This external library provides functionalities for displaying progress bars to the user during long-running operations like vulnerability scans. This can improve user experience by providing visual feedback on the progress of the scan.
aspose.cells	This external library allows for the creation, manipulation, and conversion of Excel files programmatically. It's used in scenarios where automation of Excel file processing is needed, such as generating reports or data analysis.
openpyxl	The openpyxl library provides tools for reading and writing Excel files in the `.xlsx` format. It’s widely used for data extraction, report generation, and automating Excel-based workflows.
shutil	The shutil module offers high-level operations on files and collections of files, such as copying, moving, and archiving. It's essential for file management tasks, backups, and deployment scripts.
winreg	The winreg module provides access to the Windows registry, allowing for the reading, writing, and manipulation of registry keys and values. This is useful for Windows-specific configuration and system management tasks.
grp	The grp module provides access to the Unix group database, allowing for retrieval of group information by name or ID. This is useful for user management and access control on Unix-like systems.

pam	The pam module provides an interface to the Pluggable Authentication Modules (PAM) library, used for authentication in Unix-like systems. It's essential for implementing authentication mechanisms and integrating with system-level security.

3.3.2 Development Environment

The tool was primarily developed using Visual Studio Code (VS Code), a popular code editor that offers cross-platform compatibility. This flexibility allowed for development between Windows and Linux environments. On top of that, VS Code provides the possibility to manage a Git repository from inside the tool, further simplifying development between platforms.

The logic of the tool is to work in as many operating systems as possible. So, to maximize development flexibility and ensure the tool's adaptability, a multi-platform development approach was adopted. This involved working on the tool across a range of operating systems, including Windows 10, Windows 11, CentOS 9, Fedora 28-30, Debian 11, and RaspberryPi OS. This allows for seamless development and testing regardless of the underlying operating system.

3.3.3 CVE Data Source: National Vulnerability Database (NVD)

Our Python vulnerability assessment tool relies on the **National Vulnerability Database (NVD)** maintained by the National Institute of Standards and Technology (NIST) as its primary source of vulnerability information. The NVD offers a comprehensive repository of publicly known cybersecurity vulnerabilities, including details like:

- **CVE Identifier:** A unique identifier assigned to each vulnerability.
- **Affected Software:** Software products or libraries susceptible to vulnerability.
- **Vulnerability Details:** Descriptions of the vulnerability, potential impact, and mitigation strategies.
- **Severity Level:** A classification of the vulnerability's severity (e.g., critical, high, medium, low) based on its potential impact.

Utilizing the NVD Data:

The tool leverages the NVD data in two keyways:

1. **Periodic Updates:** To ensure our vulnerability assessments reflect the latest threats, the tool incorporates a mechanism to periodically download and parse the relevant NVD data. The downloaded data is in JSON format, but this can differ depending on the chosen access method. Libraries like requests can be used to interact with the NVD API for data retrieval.
2. **CVE Matching During Scans:** During vulnerability assessments, the tool compares the software versions identified on the target system with the affected software versions listed in the NVD data. If a match is found, the tool raises a potential vulnerability alert,

highlighting the specific CVE identifier, severity, Exploitability score and Impact Score from the NVD.

The NVD's established reputation as a trusted source of vulnerability information offers several advantages for our tool. Firstly, it ensures our assessments are based on comprehensive and reliable data, minimizing the risk of basing security decisions on outdated or inaccurate information. Secondly, the NVD utilizes a standardized data format for vulnerabilities. This standardized format allows for efficient parsing and seamless integration into the tool's vulnerability assessment logic, making the processing of vulnerability data simpler and easier. Finally, the NVD team actively maintains the database, continually adding new vulnerabilities and updating existing entries with the latest information. This ongoing maintenance guarantees our tool remains effective against the evolving landscape of cybersecurity threats, ensuring the accuracy and relevance of our vulnerability assessments.

3.3.4 Code-Level Analysis: Implementing Techniques

Our Python vulnerability assessment tool leverages various techniques to identify potential security vulnerabilities on target systems and/or to have a better performance on the detection. This subsection sheds light into the code-level implementation of these key techniques, showcasing how they contribute to the tool's functionality.

1. Parsing information from the systems

A critical aspect of vulnerability assessment involves retrieving information about the system and its installed services. This information is often stored in configuration files with varying formats. To address this challenge, we employed regular expressions. By defining patterns that match specific keywords and delimiters within these files, we can efficiently extract relevant details like operating system version, service names, installed software versions and configuration files.
There are many places in our code where regex is being employed, some examples are shown in the following figures (9-10).

```
# Parse the full path of the exe and get the version
pattern = "(C:.*?exe)"
for service in services_paths:
    services_paths[service] = services_paths[service].replace("\\", "\\\\")
    match = re.search(pattern, services_paths[service])
    if match:
        filtered_service_path = match.group(1)
    else:
        filtered_service_path = ""

    if filtered_service_path:
        cmd = f"wmic datafile where 'name=\"{filtered_service_path}\"' get version"
        proc = (
            (subprocess.run(cmd, capture_output=True))
            .stdout.decode()
            .split("\n")
        )
        service_version[service] = proc[1].strip()
    else:
        service_version[service] = "Unknown"

return service_version
```

Figure 9: Regex is used in order to parse the file path.

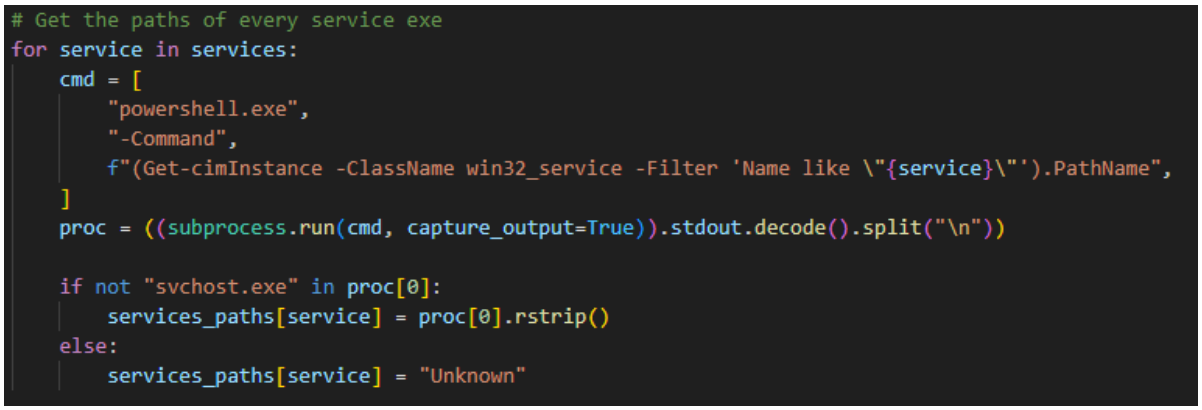
```python
def GetUsers(self):
    pattern = r"^(.*?):(.*?):"
    local_users = []
    users = subprocess.run(["cat", "/etc/shadow"], stdout=subprocess.PIPE).stdout.decode().split("\n")
    for user in users:
        captures = re.search(pattern, user)
        if captures:
            if captures.groups()[0] and len(captures.groups()[1])>=3:
                local_users.append(captures.groups()[0])

    return local_users
```

Figure 10: Regex for parsing the username and the password from the shadow file.

Beyond regular expressions, Python's built-in capabilities offer another powerful approach to searching configuration files for specific strings. The ***in*** operator within an if statement allows you to efficiently detect patterns within variables. We can see some in-code examples in the figures 11 and 12.

```python
if "ServerTokens Prod" in line:
    hardening[file]["ServerTokens Prod"] = True
if "ServerSignature Off" in line:
    hardening[file]["ServerSignature Off"] = True
if "FileETag None" in line:
    hardening[file]["Etag"] = True
if "TraceEnable off" in line:
    hardening[file]["TraceReq"] = True
if "Header edit Set-Cookie ^(.*)$ $1;HttpOnly;Secure" in line:
    hardening[file]["CookieProtection"] = True
if "Header always append X-Frame-Options SAMEORIGIN" in line:
    hardening[file]["ClickJacking Attack"] = True
if 'Header set X-XSS-Protection "1; mode=block"' in line:
    hardening[file]["X-XSS protection"] = True
if "SSLCertificateFile" in line or "SSLCertificateKeyFile" in line or "SSLCertificateChainFile" in line:
    hardening[file]["SSL"] = True
if "<LimitExcept" in line:
    hardening[file]["HTTP Request Methods Restriction"] = True
```

Figure 11: Finding specific configurations in the configuration files



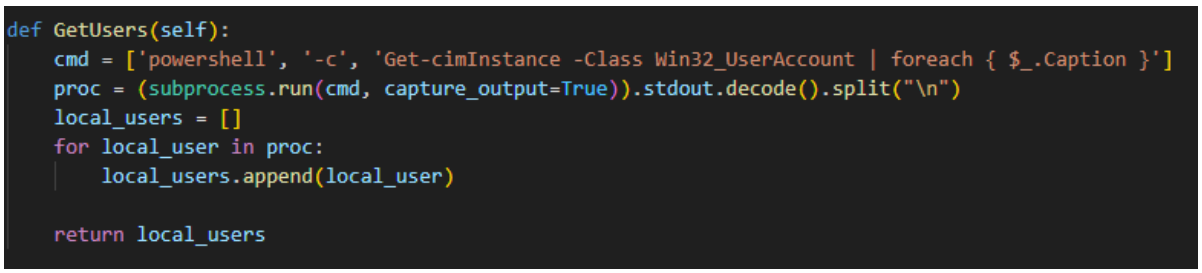
```
# Get the paths of every service exe
for service in services:
    cmd = [
        "powershell.exe",
        "-Command",
        f"(Get-cimInstance -ClassName win32_service -Filter 'Name like \"{service}\"').PathName",
    ]
    proc = ((subprocess.run(cmd, capture_output=True)).stdout.decode().split("\n"))

    if not "svchost.exe" in proc[0]:
        services_paths[service] = proc[0].rstrip()
    else:
        services_paths[service] = "Unknown"
```

Figure 12: Verifying that the output of the command doesn't contain a specific value

2. User/Group Enumeration through WMI/CIM Queries (Windows Systems)

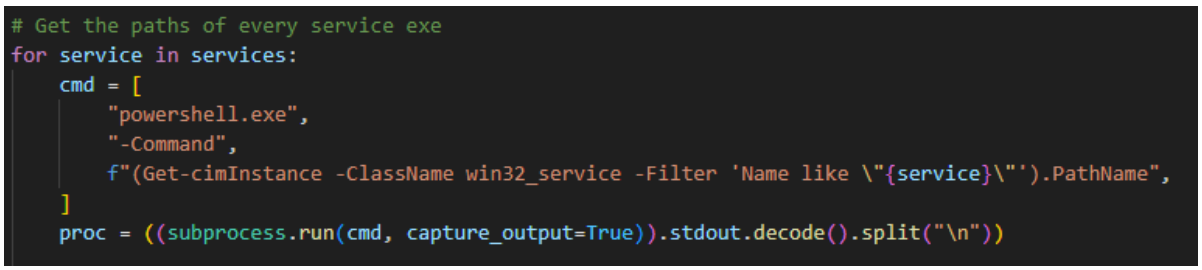
On Windows systems, we can leverage the Windows Management Instrumentation (WMI) to retrieve information about users and groups on the target machine. This information can be valuable for identifying weak security configurations, such as accounts with excessive privileges or unused accounts. However, it's important to note that this technique should only be used on authorized systems and only for good purposes.

The implementation involves utilizing the subprocess library to execute specific CIM queries. Figure 13 and Figure 14 provide some examples of this implementation:



```
def GetUsers(self):
    cmd = ['powershell', '-c', 'Get-cimInstance -Class Win32_UserAccount | foreach { $_.Caption }']
    proc = (subprocess.run(cmd, capture_output=True)).stdout.decode().split("\n")
    local_users = []
    for local_user in proc:
        local_users.append(local_user)

    return local_users
```

Figure 13: Using the Get-CIMInstance command to interact with the Win32_UserAccount WMI object.



```
# Get the paths of every service exe
for service in services:
    cmd = [
        "powershell.exe",
        "-Command",
        f"(Get-cimInstance -ClassName win32_service -Filter 'Name like \"{service}\"').PathName",
    ]
    proc = ((subprocess.run(cmd, capture_output=True)).stdout.decode().split("\n"))
```

Figure 14: Retrieving the path of a service querying the WMI database.

3. Dictionary Attack

Disclaimer: Due to the potential for misuse, performing a dictionary attack is generally discouraged. It can be used for malicious purposes and can be ineffective against strong passwords.
However, for educational purposes only, here's an overview of how a dictionary attack is implemented in this thesis:

Windows

- The tool requires a dictionary file containing potential passwords to be used in the attack. By default, a wordlist is provided with the tool, but there is the option to use a custom one.
- In order to perform an authentication, the tool tries to start the cmd using the discovered user's credentials.
- A rate limiter and proper error handling is implemented to avoid overwhelming the target system or triggering security measures.

In the following code snippet, you can see the implementation of the dictionary attack in the tool.

```
def PassCracker(self, wordlist, local_user):
        length = len(wordlist)
        count = 0
        widgets = ['Progress: ', Percentage(), ' | ', Timer(), ' | ', AdaptiveETA()]
        bar = ProgressBar(widgets=widgets, max_value=100).start()

        for password in wordlist:
            #password = ""
            scriptBlockLine1 = "{"+f'$pass="{password}"|ConvertTo-SecureString -AsPlainText -Force'
            scriptBlockLine2 =  f"\n$Cred=New-Object
System.Management.Automation.PsCredential('{local_user}',$pass)"
            scriptBlockLine3 = '\nStart-Process -FilePath cmd.exe /c -Credential $Cred }'
            scriptBlock = scriptBlockLine1 + scriptBlockLine2 + scriptBlockLine3

            cmd = ['powershell', '-c', f'Invoke-Command -ScriptBlock {scriptBlock}']
            proc = (subprocess.run(cmd, capture_output=True))
            error = proc.stderr.decode().split("\n")
            output = proc.stdout.decode().split("\n")
            time.sleep(1) # Rate limiter to avoid overwhelming the target PC
            if not error[0] and not output[0]:
                bar.update(100)
                return True, password
```

```
            count+=1
            bar.update(self.TranslateTo100(count, length))

        return False, password
```

Linux

- The tool requires a dictionary file containing potential passwords to be used in the attack. By default, a wordlist is provided with the tool, but there is the option to use a custom one.
- In order to authenticate we simulate a pam authentication using the PAM library in python for a specified user.
- A rate limiter and proper error handling is implemented to avoid overwhelming the target system or triggering security measures.

While the tool's dictionary attack functionality shares similarities with the Windows version, the Linux implementation differs in how it handles authentication. Here's a code snippet demonstrating the Linux-specific approach.

```
def PassCracker(self, wordlist, local_user):

        length = len(wordlist)
        count = 0
        widgets = ['Progress: ', Percentage(), ' | ', Timer(), ' | ', AdaptiveETA()]
        bar = ProgressBar(widgets=widgets, max_value=100).start()
        for password in wordlist:
            p = pam.pam()
            auth = p.authenticate(local_user, password)
            if auth:
                bar.update(100)
                return True, password
            time.sleep(0.5) # Rate limiter to avoid overwhelming the target PC

            count+=1
            bar.update(self.TranslateTo100(count, length))

        return False, password
```

4. **Data Caching for Performance Optimization**

To enhance the efficiency of vulnerability assessments, our tool implements a local CVE cache. The National Institute of Standards and Technology (NIST) National Vulnerability Database (NVD) serves as a source for our CVE data. However, repeatedly fetching the latest CVE information during scans can be resource-intensive, especially if the internet connection is slow.
Our approach involves storing a locally accessible file containing relevant CVE data retrieved from the NVD. This cached data includes details like CVE identifier, affected software versions, and severity levels. Here's a breakdown of the implementation:

- Downloading and Parsing CVE Data: The tool periodically downloads the latest CVE data from the NVD using libraries like requests to interact with the NVD API. It then parses the downloaded data (in JSON format) to extract the required CVE details.

- Local Storage: The extracted CVE information is stored in a JSON format within a local file. This file serves as the local CVE cache.

- CVE Lookup During Scans: During vulnerability assessments, the tool first checks the local CVE cache for relevant entries. If the required CVE information is present and sufficiently recent (based on a defined refresh interval), it utilizes the cached data for comparisons. This eliminates the need for repeated NVD queries, improving scan performance.

- Cache Invalidation: To ensure the cached data remains up to date, the tool implements a cache invalidation strategy. This strategy involves weekly updates to download fresh CVE data from the NVD and update the local cache.

Our tool uses the CVEUpdater script to take care of this process. This script downloads information about security vulnerabilities (CVEs) and stores them in a local file for future reference. It focuses on vulnerabilities that are relevant to the specific services it finds running on the system. In the following screenshot you can see the code implementation for the vulnerability retrieval process:

```python
def GetVulnerabilities(self):
    cached = False
    cached_cves_f = [pos_json for pos_json in os.listdir('.') if pos_json.startswith('CachedCVEs')]
    if cached_cves_f:
        if exists(cached_cves_f[0]):
            if not os.stat(cached_cves_f[0]).st_size == 0:
                print("Cached file found")
                cached = True
                cached_vulnerabilities = self.get_CVEs_Local(cached_cves_f[0])

                # Check if there is a new service that doesn't exist in the cached file
                for service, version in self.versions.items():
                    if  not version == "Unknown" and not service in cached_vulnerabilities.keys():
                        self.get_CVEs_NIST({service:version})

                return cached_vulnerabilities, cached

    return self.get_CVEs_NIST(), cached
```

Figure 15: CVE retrieval process.

Finally, by using a local CVE cache, our tool reduces reliance on external resources during scans and streamlines the vulnerability assessment process. This approach minimizes network traffic and improves overall scan execution times, particularly in scenarios with limited internet bandwidth.

Chapter 4

4 Results and Evaluation of the Vulnerability Assessment Tool

The preceding chapters explored the design and implementation of our Python vulnerability assessment tool. This chapter analyses the results of our evaluation process, aiming to assess the tool's effectiveness in identifying vulnerabilities and its overall performance characteristics. Through an extensive testing methodology, we analyze the tool's ability to detect various vulnerabilities, its accuracy in reporting findings, and its resource utilization during scans. The insights gleaned from this evaluation will be useful in refining the tool and ensuring its continued effectiveness.

4.1 Testing Methodology

To thoroughly evaluate the effectiveness of our Python vulnerability assessment tool, we implemented a simple and efficient testing methodology. This methodology aimed to assess the tool's ability to identify various vulnerabilities, its accuracy in reporting findings, and its overall performance characteristics.

Building a Test Environment

The foundation of our testing methodology was a controlled environment containing both virtual machines (VMs) and bare-metal devices. This environment was crafted to mirror real-world system deployments and included a variety of commonly used operating systems.

The VM environment featured CentOS 9, Fedora versions 38, and 40, Debian 11 and Windows 10, each running with a different configuration ranging from 4GB to 8GB of RAM and 4 to 6 CPU cores. This configuration allowed for consistent testing across different operating systems within the virtualized environment.

For bare-metal testing, we utilized a spectrum of devices to represent real-world scenarios. A high-performance desktop PC equipped with an AMD Ryzen 5 3600 CPU and 16GB of RAM. We also tested on a Lenovo IdeaPad 3 15IAU7 laptop, a commonly encountered configuration in personal and professional settings, equipped with an Intel i5 1235U CPU and 8GB of RAM. Finally, the tool's capability on resource-constrained devices was evaluated by running tests on Raspberry Pi 3 and 4 running Raspberry Pi OS.

Table 3: VM Testing Environment

Operating System (OS)	Specifications
CentOS 9	8GB RAM, 6 CPU cores
Debian 11 (Bullseye)	4GB RAM, 2 CPU cores

Fedora (38, 40)	4GB RAM, 4 CPU cores
Windows 10	8GB RAM, 6 CPU cores

Table 4: Bare Metal Testing Environment

Bare Metal	Device	Operating System (OS)
Desktop	AMD Ryzen 5 3600 CPU 16GB RAM	Windows 11, Fedora 40
Laptop	Intel i5 1235U CPU 8GB RAM	Fedora 37, Fedora 38, Kali Linux
Raspberry Pis	Versions 3 & 4	Raspberry Pi OS

Introducing Vulnerabilities for Assessment

To effectively evaluate the tool's vulnerability detection capabilities, we introduced a variety of vulnerabilities into the test environment. This process involved installing software packages with known vulnerabilities and modifying system configurations to create exploitable weaknesses. By deliberately introducing these vulnerabilities, we ensured the testing process comprehensively assessed the tool's ability to identify a diverse range of security risks across different operating systems and hardware configurations.

Benefits of Testing in Limited Resource Environments

While real-world servers often possess significantly more RAM, testing with limited resources offers a valuable advantage. By demonstrating the tool's effectiveness in a constrained environment, we establish a strong foundation for its performance in scenarios with more ample resources. This approach provides confidence that the tool can efficiently identify vulnerabilities even in situations where hardware limitations might exist.

4.2 Results and Analysis

Building upon the testing methodology outlined in the previous section, this section dives into the results obtained from evaluating our Python vulnerability assessment tool. We'll analyze the tool's performance across various metrics, including its ability to detect diverse vulnerability types, the accuracy of its reported findings, and its overall resource utilization during scans. By closely examining the test data, we aim to gain a clear understanding of the tool's strengths, limitations, and potential for further refinement. The insights gleaned from this analysis will be valuable in shaping future iterations of the tool and ensuring its continued effectiveness.

The results will be gathered from each Operating System and analyzed based on each component of the tool starting with the services.

4.2.1 Windows Assessment results

Here we will gather and analyze all the results that the tool brought during the assessment in Windows OS.

4.2.1.1 Services Assessment

The results that are described here have been gathered from the desktop system. The tool didn't use any cached data yet.

Table 5: Execution in the desktop system without cache

Category	Description
System - OS	Desktop – Windows 11
CPU	AMD Ryzen 5 3600
RAM	16GB
Services Discovered	135 out of 135
Services with Version Info	42 out of 135
Services with Active Vulnerabilities	0
Services with Potential Vulnerabilities	7
Execution Time	6 minutes and 15 seconds

Our analysis reveals that the tool identified seven potentially vulnerable services. These services either had a reported CVE (Common Vulnerability and Exposures) in the past, but the version on the system didn't match the reported vulnerability, or the service name had a partial match with a known vulnerable service. In this case, further investigation is required to confirm the validity of these vulnerabilities. We'll need to manually check the associated CVEs for each service.

After the manual investigation we can break down the results. Six out of the seven identified vulnerabilities had a correct service name match, but a version mismatch. This lowered the tool's confidence level in these findings. The remaining service, EFS, was flagged incorrectly due to a mismatch in the service name itself.

```
== Services ==

'133 services have been discovered'


== Versions ==

'42 services report their versions'


=== Vulnerabilities ===

== Active ==

None found

== Other Possible Matches ==

{'CorsairService': {'CVE': 'CVE-2018-12441',
                    'Exploitability Score': 3.9,
                    'Impact Score': 10.0,
                    'Service Version': '3.38.0.4',
                    'Severity': 'HIGH'},
 'EFS': {'CVE': 'CVE-2002-1814',
         'Exploitability Score': 3.9,
         'Impact Score': 6.4,
         'Service Version': '10.0.22621.3235',
         'Severity': 'MEDIUM'},
 'MSDTC': {'CVE': 'CVE-2002-0224',
           'Exploitability Score': 10.0,
           'Impact Score': 2.9,
           'Service Version': '2001.12.10941.16384',
           'Severity': 'MEDIUM'},
 'Spooler': {'CVE': 'CVE-1999-0898',
             'Exploitability Score': 3.9,
             'Impact Score': 10.0,
             'Service Version': '10.0.22621.3672',
             'Severity': 'HIGH'},
 'Steam Client Service': {'CVE': 'CVE-2015-4016',
                          'Exploitability Score': 10.0,
                          'Impact Score': 2.9,
                          'Service Version': '8.98.79.13',
                          'Severity': 'MEDIUM'},
 'VMnetDHCP': {'CVE': 'CVE-2019-5540',
               'Exploitability Score': 8.0,
               'Impact Score': 2.9,
               'Service Version': '16.1.0.683',
               'Severity': 'MEDIUM'},
 'VMware NAT Service': {'CVE': 'CVE-2017-4949',
                        'Exploitability Score': 3.4,
                        'Impact Score': 10.0,
                        'Service Version': '16.1.0.683',
                        'Severity': 'MEDIUM'}}


===== Execution time =====
Tool execution total time: 0:06:15.291782
Service scan duration: 0:06:15.289779
```

Figure 16: Execution results.

In the next step we removed the “unpatched” applications and safely introduced a vulnerable one (teamviewer version 14.2 with a reported CVE-2019-18196) to see If our tool can successfully detect it. Remarkably our tool returned the results with the reported application and with some more CVEs that we couldn’t find easily through the website. Figure 17 shows the results from the report generated by the tool.

Services Assessment

Service Name	TeamViewer	
	Active Vulnerabilities	Possible Vulnerabilities
CVE	CVE-2019-18196	CVE-2019-18988
Exploitability Score	3,4	3,4
Impact Score	10	6,4
Service Version	14.2.2558	14.2.2558
Severity	MEDIUM	MEDIUM
Starting version	12.0.0	0
Ending Version	14.7.1965	14.7.1965
CVE	CVE-2021-34803	CVE-2021-34858
Exploitability Score	3,4	8,6
Impact Score	6,4	6,4
Service Version	14.2.2558	14.2.2558
Severity	MEDIUM	MEDIUM
Starting version	10.0.2551	0
Ending Version	14.7.48644	15.21.2
CVE		CVE-2021-35005
Exploitability Score		3,9
Impact Score		2,9
Service Version		14.2.2558
Severity		LOW
Starting version		0
Ending Version		15.18.5.0
CVE		CVE-2022-23242
Exploitability Score		3,4
Impact Score		2,9
Service Version		14.2.2558
Severity		LOW
Starting version		0
Ending Version		15.28

Figure 17: Active vulnerabilities reported by the tool

Results with the usage of a cache file

To assess the impact of caching on execution time, we ran the analysis again using a cached file. Table 6 summarizes the results.

Table 6: Results using cache file.

Category	Description
Services Discovered	135 out of 135
Services with Version Info	42 out of 135
Services with Active Vulnerabilities	0
Services with Potential Vulnerabilities	7
Execution Time	2 minutes and 39 seconds

From this execution we can safely deduce that the cache file brought about a dramatic ~200% reduction in execution time since the first execution finished in 6 minutes and 15 seconds and the execution with the cache file finished in 2 minutes and 39 seconds.

Figure 18: Execution results with cache file

4.2.1.2 User Assessment

Next up, let's look at how the tool performed when assessing user accounts. We'll be using the same approach as with the services assessment, testing the tool on the Desktop. Here, we'll see how well the tool finds user accounts, identifies situations where someone might have too much access (high privileged groups), and uncovers weak passwords.

Starting again with the desktop system, we executed the tool and asked it to try and find the password of all the discovered accounts. For the testing purposes we used a wordlist with 60 passwords. One of them was the correct password for the two accounts in the desktop. Table 7 projects the findings.

Table 7: Desktop results

Category	**Result**
Discovered Accounts	7 out of 7
Discovered Vulnerable Users	2 out of 2
Discovered Vulnerable High Privileged Users	0 out of 0
User Assessment Execution Time	10 minutes, 49 seconds

As expected, the 2 vulnerable users were found. None of them were a member of a privileged group. In Figure 18 we can see the output of the tool with the two vulnerable accounts.

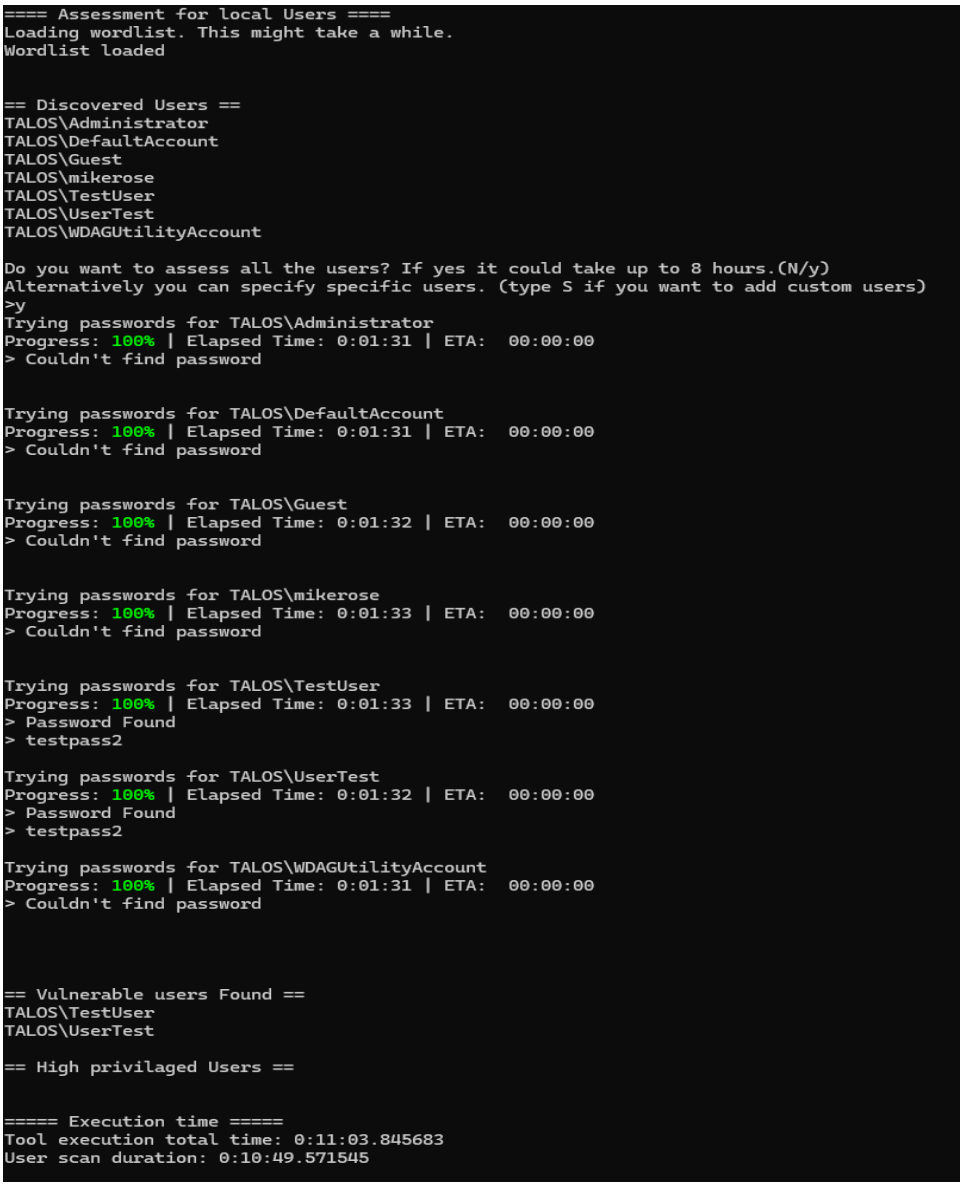


Figure 19: Results from first user assessment.

Our next step involves evaluating the tool's effectiveness in identifying privilege escalation scenarios. To achieve this, we will introduce a controlled vulnerability. Specifically, we will add the previously identified vulnerable user *TestUser* to the administrators group. Following this modification, we will re-run the user assessment to determine if the tool detects this elevated privilege level for *TestUser*.

In a positive outcome, the tool successfully detected the elevated privilege level for the *TestUser*. This finding confirms the tool's effectiveness in identifying situations where users might possess excessive permissions, potentially posing a security risk. In figure 20 we can see that the tool has reported the *TestUser* account as a member of the *Administrators* group, thus making it a privileged account**.**

Table 8: Results after high privileged user was added.

Category	Result
Discovered Accounts	7 out of 7
Discovered Vulnerable Users	2 out of 2
Discovered Vulnerable High Privileged Users	1 out of 1
User Assessment Execution Time	8 minutes, 46 seconds

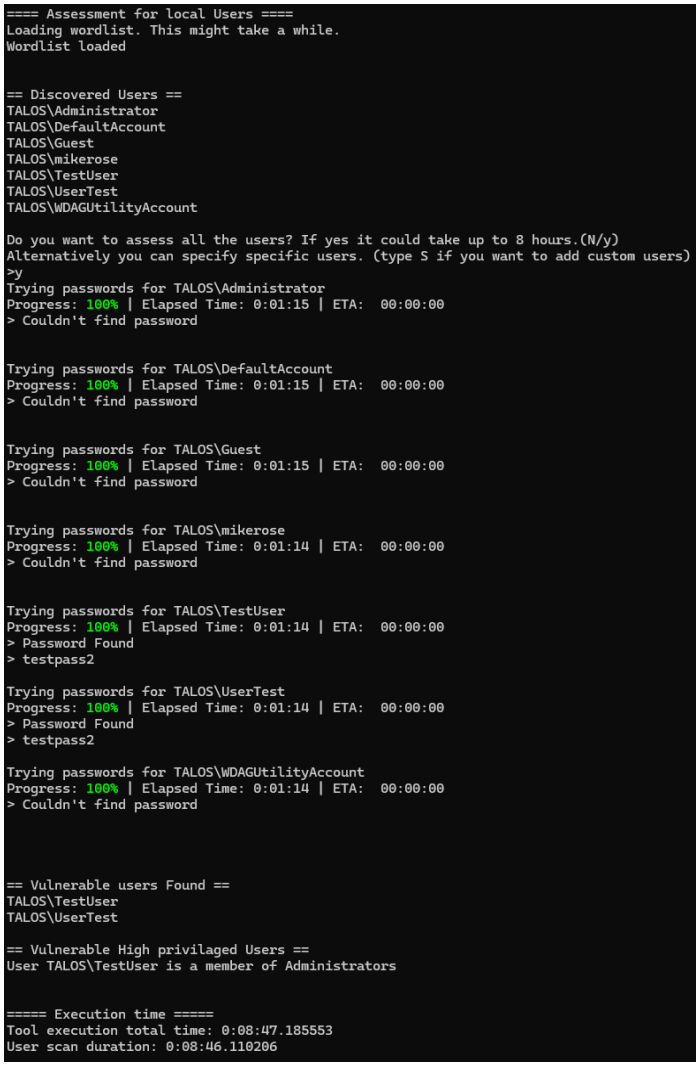


Figure 20: Output with the identified vulnerable privileged account.

4.2.1.3 Configurations assessment

To evaluate the tool's effectiveness in assessing Windows system configurations, we conducted a configuration assessment across multiple systems. The tool examined a variety of settings and software configurations to identify potential misconfigurations that could compromise security.

Apache assessment

Our initial assessment of Apache configuration files on the windows systems focuses on identifying missing security settings based on CIS benchmarks. Table 13 summarizes the configuration discrepancies found during this process.

Table 9: httpd.conf assessement

Recommended Setting	Presence in the configuration file (httpd.conf)
ServerTokens Prod	X
ServerSignature Off	X
ApacheOptions	X
Etag	X
TraceReq	X
CookieProtection	X
ClickJacking Attack	X
X-XSS protection	X
SSL	X
Browser Listing	X
System Setting Protection	X
HTTP Request Methods Restriction	X

In order to evaluate if the tool can pick up the changed in the configurations, we changed some parameters in the httpd.conf file to see if the tool will report them as present. Table 14 depicts the output of the tool and its findings. In figures 20 and 21 we can see the output for the tool with the results from before and after the configuration hardening.

Table 10: httpd.conf assessment after modifications

Recommended Setting	**Presence in the configuration file (httpd.conf)**
ServerTokens Prod	✓
ServerSignature Off	✓
ApacheOptions	✗
Etag	✓
TraceReq	✓
CookieProtection	✗
ClickJacking Attack	✓
X-XSS protection	✓
SSL	✗
Browser Listing	✓
System Setting Protection	✗
HTTP Request Methods Restriction	✗

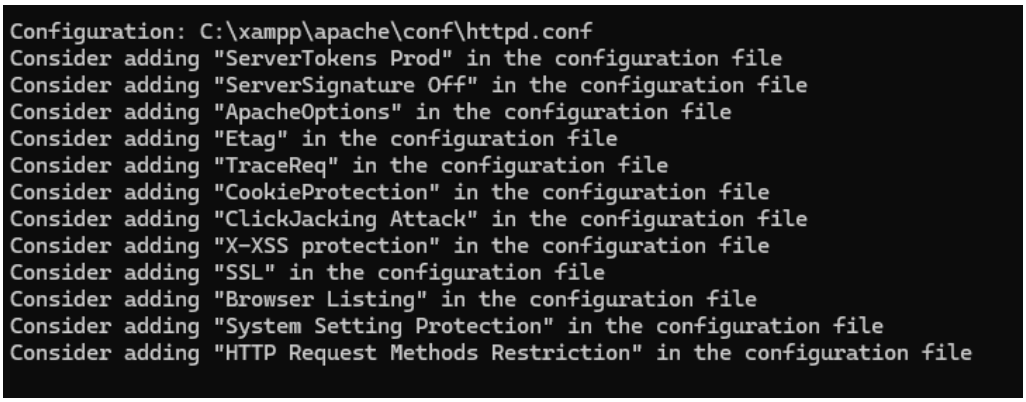


Figure 21: Assessment results before the hardening

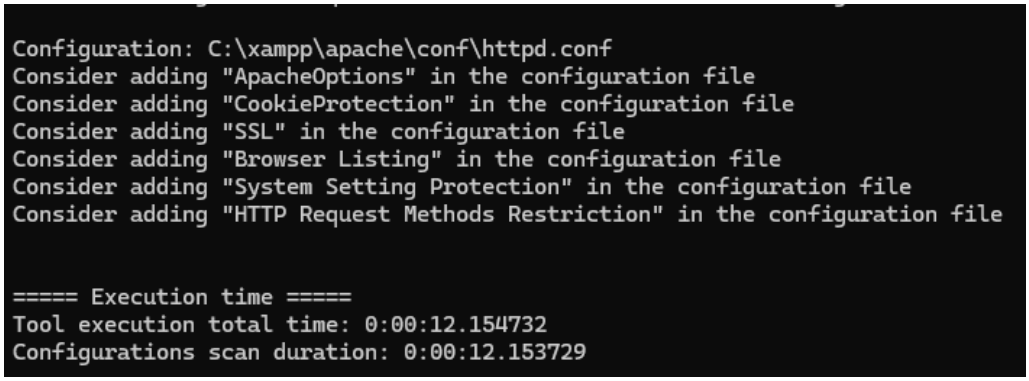


Figure 22: Assessment results after the hardening

Registry assessment

To identify potential persistence mechanisms and unauthorized program executions, the tool incorporates a registry analysis module. By searching in the registry hives, such as those associated with startup programs, services, and software installation, the tool aims to discover suspicious entries or unexpected configurations. This will help us detect potential indicators of compromise (IOCs).

To test our tool's ability to accurately identify unauthorized startup programs, a series of tests were conducted involving the manipulation of startup registry keys. By intentionally adding and removing programs from the startup locations, the tool's performance in detecting these changes was assessed.

In the first run we left out registry untouched to see the already registered programs as shown in figure 22.

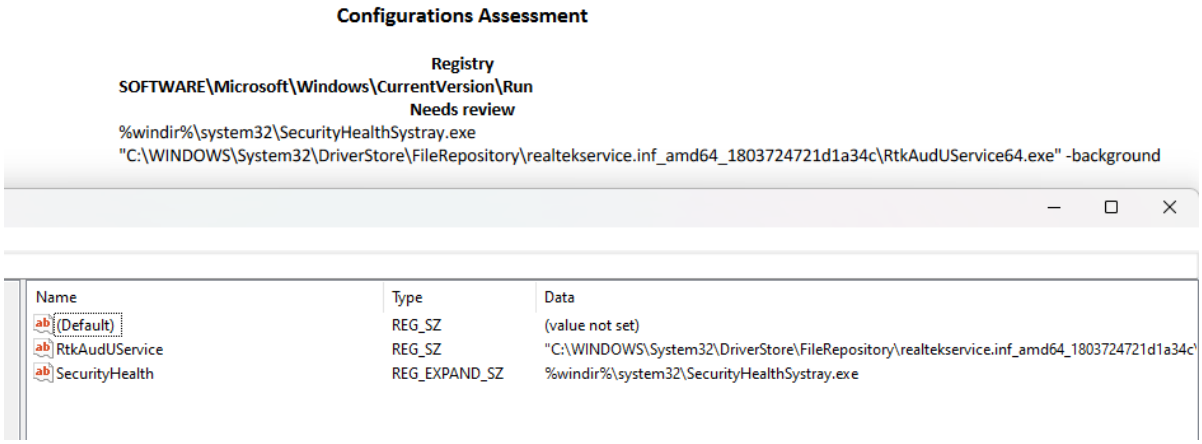


Figure 23: Registry keys identified by the tool

The tool successfully identified all pre-existing entries within the specified registry keys, demonstrating accurate detection capabilities. Figure 22 visually represents the tool's output, aligning with the expected registry configurations. This initial evaluation serves as a baseline for assessing the tool's ability to identify changes in registry configurations.

The next step was to add a new program to run during the startup process and see if the tool could detect the difference. After we added the **malicious_persistant_malware.exe** we re-run the tool and got the results as seen in figure 23.

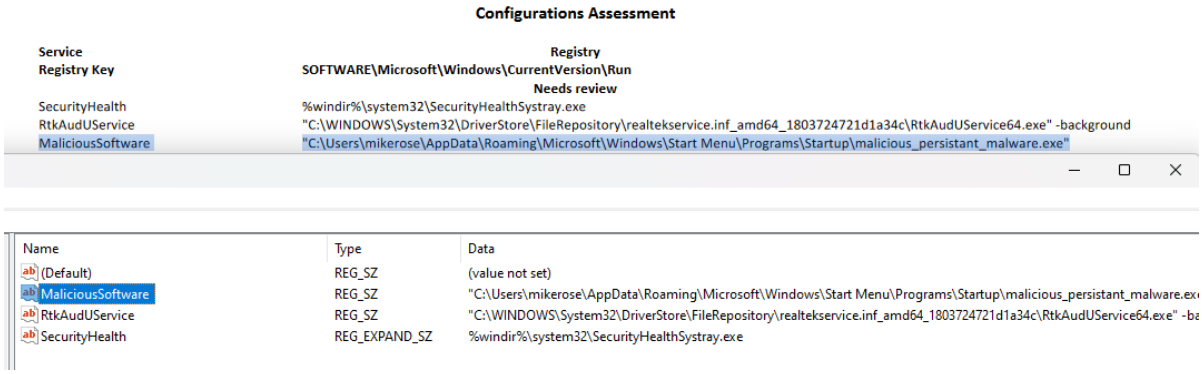


Figure 24: Registry keys with malicious software

The tool successfully identified the newly added program during the scan, demonstrating its effectiveness in detecting changes to startup configurations. In general, this capability is crucial for identifying potential persistence mechanisms employed by malicious actors.

Another key aspect of the assessment focused on the firewall's status. The tool examined the EnabledFirewall registry value to determine if the Windows Firewall was active. By checking if this value was set to 1, indicating an enabled firewall. If the value is set to 0 then the tool

reports it for the administrators to further investigate it. Figure 24 depicts a disabled firewall in the report where the tool has highlighted the possible misconfiguration.

Configurations Assessment

Service	Registry
Registry Key	**SOFTWARE\Microsoft\Windows\CurrentVersion\Run**
	Needs review
SecurityHealth	%windir%\system32\SecurityHealthSystray.exe
RtkAudUService	"C:\WINDOWS\System32\DriverStore\FileRepository\realtekservice.inf_amd64_1803724721d1a34c\RtkAudUService64.exe" -background
Registry Key	**System\CurrentControlSet\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile**
	Needs review
EnableFirewall	0

Figure 25: Disabled Firewall discovered by the tool

Filezilla assessment

The assessment of Filezilla configurations was undertaken in the windows desktop system to identify basic misconfigurations that could jeopardize the security and functionality of the application. In the initial execution, our VA tool identified numerous default misconfigurations depicted in figure 25.

Configurations Assessment

Service	Filezilla
Configuration File	**C:\xampp\FileZillaFTP\FileZilla Server Interface.xml**
	Recommendations
	Warning: Please set the password minimum length to >12
	Consider adding "TLSRequired" in the configuration file
	Consider adding "MaxClients" in the configuration file
Configuration File	**C:\xampp\FileZillaFTP\FileZilla Server.xml**
	Recommendations
	Warning: Please set the password minimum length to >12
	Consider adding "TLSRequired" in the configuration file
	Consider adding "MaxClients" in the configuration file
Configuration File	**C:\xampp\FileZillaFTP\FileZillaServer.xml**
	Recommendations
	Warning: Please set the password minimum length to >12
	Consider adding "TLSRequired" in the configuration file
	Consider adding "MaxClients" in the configuration file

Figure 26: Discovered misconfigurations in filezilla configuration files

Following these findings, we patched the identified vulnerabilities and re-ran the assessment to evaluate the effectiveness of the changes and to see if the tool could detect the improvements. Figure 26 highlights the differences post-patching.

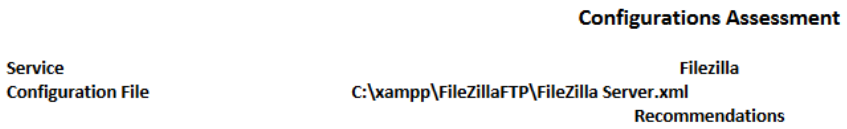


Figure 27: Example of no misconfiguration to report

4.2.2 Linux Assessment results

Continuing our evaluation, we will examine how well the tool functions and produces accurate results within a Linux operating system.

4.2.2.1 Services Assessment

Following the analysis of service assessment on Windows-based systems, we will now take a look at a Linux environment. This execution was done on the desktop system running fedora 40. The results in table 11 are from the first execution of the services assessment module (In Figure 20 you can see the results given by the tool).

Table 11: Fedora 40 execution results

Category	Description
Services Discovered	42 out of 42
Services with Version Info	14 out of 42
Services with Active Vulnerabilities	0
Services with Potential Vulnerabilities	8
Execution Time	1 minutes and 46 seconds

The tool successfully identified all 42 active services running on the system. While version information was available for 14 of these services, enabling vulnerability checks, no active vulnerabilities were detected within this subset. However, 8 services were flagged as having potential vulnerabilities based on service name recognition alone. Further investigation is required to determine the accuracy of these potential vulnerabilities and their actual risk level. In figure 27 we can see the output of the potential vulnerabilities.

Results with the usage of a cache file

To evaluate the potential performance benefits of caching previously gathered service information, we repeated the service assessment on the Fedora 40 desktop system while utilizing a locally stored cache.

With the cached service data, the tool successfully identified all 42 active services within 15 seconds. This represents an 85.85% reduction in assessment time compared to the initial assessment without caching, demonstrating the effectiveness of the caching mechanism in improving performance (In Table 12 you can find the result overview and in figure 28 the execution time with the potential vulnerabilities).

Table 12: Fedora 40 execution results with cache file

Category	Description
Services Discovered	42 out of 42
Services with Version Info	15 out of 42
Services with Active Vulnerabilities	0
Services with Potential Vulnerabilities	8
Execution Time	15 seconds

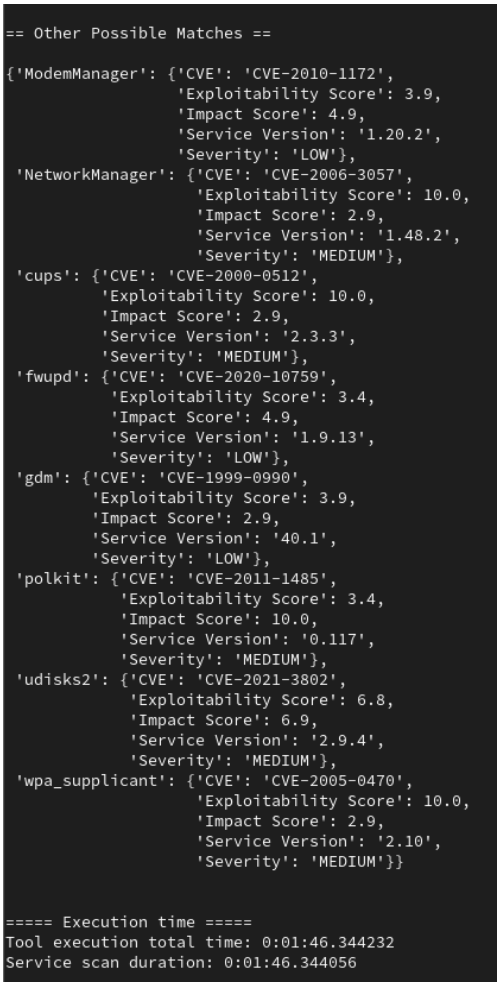


Figure 28: Printed Results without cache

Figure 29: Printed Results with caching

4.2.2.2 Users Assessment

The user assessment on the Fedora 40 desktop system successfully discovered all user accounts on the system. A critical test involved adding the previously identified vulnerable user *TestUser* to the wheel group. The tool accurately detected this elevated privilege level, highlighting its effectiveness in identifying potential privilege escalation scenarios. In the figure 29 we can see the results printed with the TestUser being a normal user with weak password. Figure 30 shows the elevated privilege attribute.

Table 13: User assessment results without a vulnerable privileged account

Category	Result
Discovered Accounts	3 out of 3
Discovered Vulnerable Users	1 out of 1
Discovered Vulnerable High Privileged Users	0 out of 0
User Assessment Execution Time	7 minutes 46 seconds

```
Running on RedHat based



==== Assessment for local Users ====
Loading wordlist. This might take a while.
Wordlist loaded


== Discovered Users ==
root
mikerose
TestUser
Do you want to assess all the users? If yes it could take up to 3 hours.(N/y)
Alternatively you can specify specific users. (type S if you want to add custom users)
>y
Trying passwords for root
Progress: 100% | Elapsed Time: 0:02:37 | ETA:  00:00:00
> Couldn't find password


Trying passwords for mikerose
Progress: 100% | Elapsed Time: 0:02:29 | ETA:  00:00:00
> Couldn't find password


Trying passwords for TestUser
Progress: 100% | Elapsed Time: 0:02:40 | ETA:  00:00:00
> Password Found
> weakpass



== Vulnerable users Found ==
TestUser

== Vulnerable High privilaged Users ==


===== Execution time =====
Tool execution total time: 0:07:54.488187
User scan duration: 0:07:46.697633
```

Figure 30: User with weak password was discovered

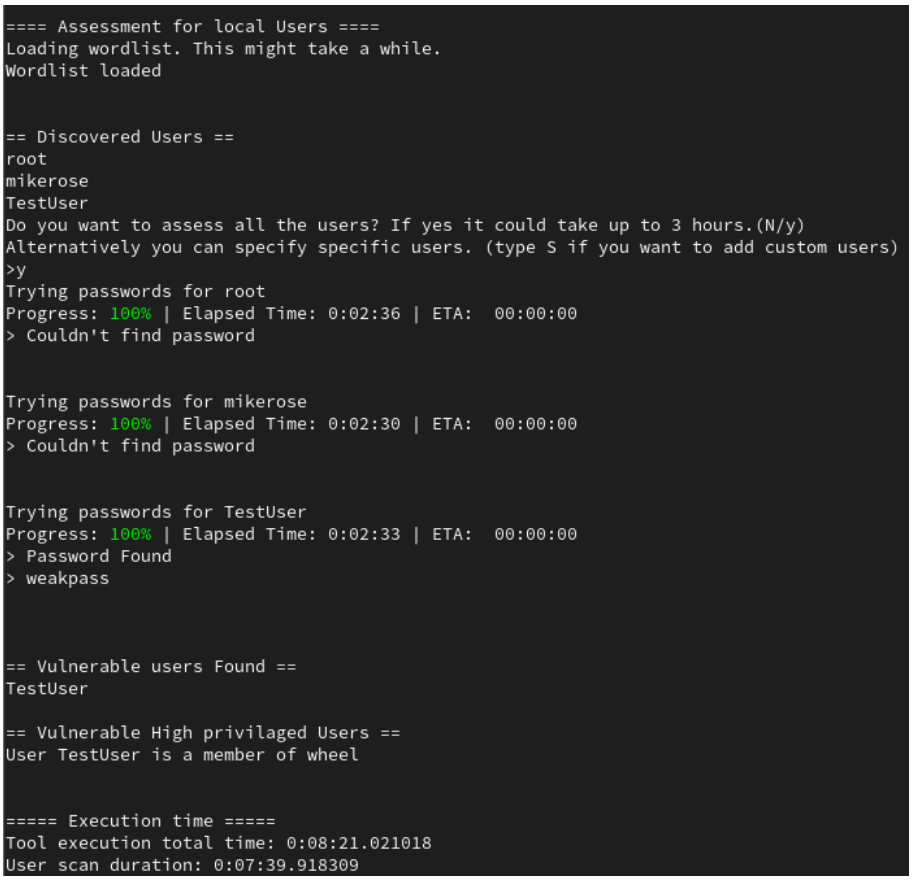


Figure 31: Privileged user with weak password discovered

Table 14: User Assessment results with vulnerable privileged account

Category	**Result**
Discovered Accounts	3 out of 3
Discovered Vulnerable Users	1 out of 1
Discovered Vulnerable High Privileged Users	0 out of 0
User Assessment Execution Time	7 minutes 39 seconds

4.2.2.3 Configurations assessment

To evaluate the tool's ability to assess linux system configurations, we conducted a configuration assessment across multiple distros. The tool analyzed various system settings and software configurations to identify potential misconfigurations that could pose security risks.

Apache assessment

Firstly, we will see the assessment we conducted for the apache configuration files. In short, we search for missing configurations that help harden the web server (based on CIS security recommendations). In the table 15 we can see the discovered configurations from the first execution in the fedora system.

Table 15: Default Apache configurations in Fedora system

Recommended Setting	Presence in the configuration file (httpd.conf)
ServerTokens Prod	✗
ServerSignature Off	✗
ApacheOptions	✗
Etag	✗
TraceReq	✗
CookieProtection	✗
ClickJacking Attack	✗
X-XSS protection	✗
SSL	✗
Browser Listing	✗
System Setting Protection	✗
HTTP Request Methods Restriction	✗

Table 15 summarizes the recommended security configurations for the /etc/httpd/conf/httpd.conf file, along with their presence in the analyzed system. A checkmark (✓) indicates the presence of the recommended setting, while a cross (✗) indicates its absence. The default apache configuration doesn't have any hardening options in it and the tool manage to discover the missing setting.

To assess the tool's ability to detect configuration deviations, a custom configuration file was created incorporating several recommended security settings from the previous analysis.

By comparing the results from the custom configuration assessment to the previous assessment using the default configuration, we can evaluate the tool's accuracy in identifying implemented security measures and highlighting any remaining misconfigurations. In the table 16 we can see the discovered results from the new configuration file.

Table 16: Partially Hardened apache configuration file

Recommended Setting	Presence in the configuration file (test.conf)
ServerTokens Prod	✓
ServerSignature Off	✗
ApacheOptions	✗
Etag	✓
TraceReq	✓
CookieProtection	✗
ClickJacking Attack	✗
X-XSS protection	✓
SSL	✓
Browser Listing	✗
System Setting Protection	✓
HTTP Request Methods Restriction	✓

In Figure 31 the output from the configuration assessment is depicted. We can easily identify the different levels of hardening between the two files.

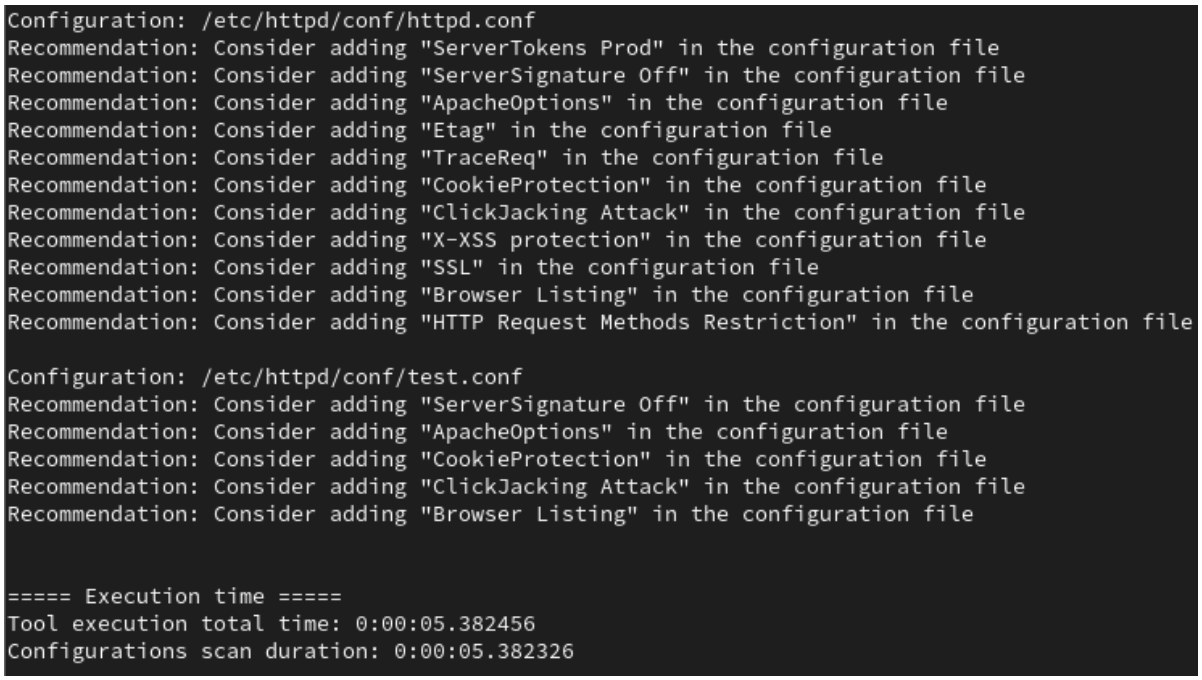


Figure 32: Configuration assessment output

Postgresql Assessment

Next, to assess probable missconfigurations of the PostgreSQL database, an analysis of the primary configuration files was conducted. This evaluation focused on identifying potential misconfigurations that could compromise the database's integrity and availability. In table 17 we can see the options that should be considered for hardening a postgresql database.

Table 17: Parameters to be checked

Option	Description
listen_addresses	Defines allowed client connections
ssl	Enables/disables SSL encryption
keepalives	Configures TCP keep-alive parameters
Authentication methods	[List of allowed methods, e.g., md5, scram-sha-256, trust]

In the first execution of our configuration assessment the tool identified some missing options as depicted in figure 32.

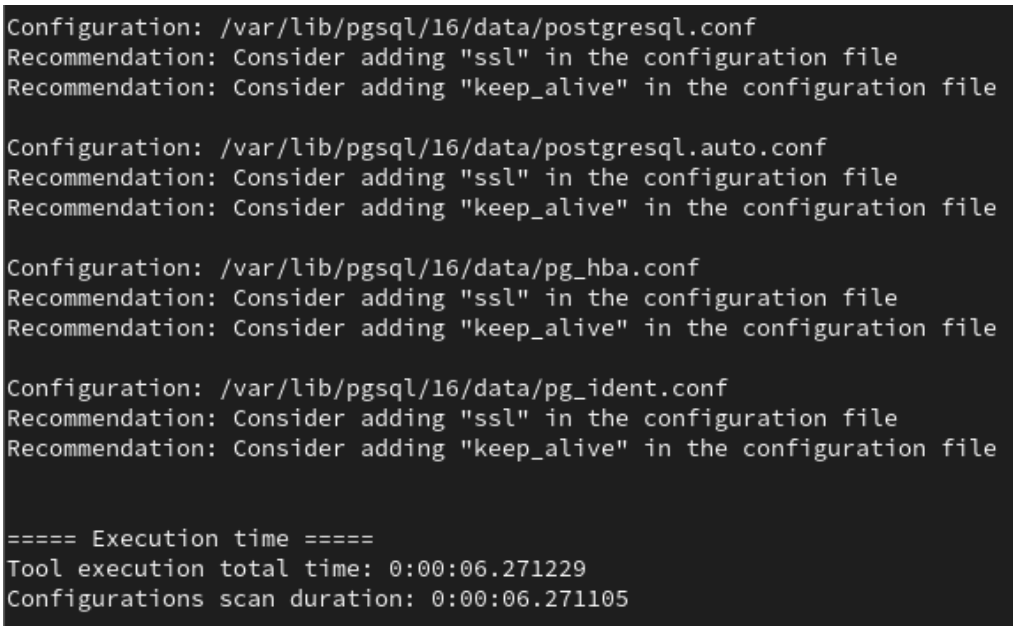


Figure 33: Discovered weak configurations

To further evaluate the tool's accuracy in identifying configuration deviations we modified some of the files to have hold missconfigurations and some others to be perfectly fine. Then we re-run the tool to assess its ability to detect these configurations.

Figure 33 shows that the tool detected the changes from the files. In the **pg_hba.conf** file we added a connection that doesn't require any authentications and a parameter that allows every IP to connect to the database with no restrictions. The tool successfully detected those configurations and reported them as warnings. On the other hand, the configuration file **postgresql.conf** had some additional hardening done, thus the tool couldn't find any bad configurations to report.

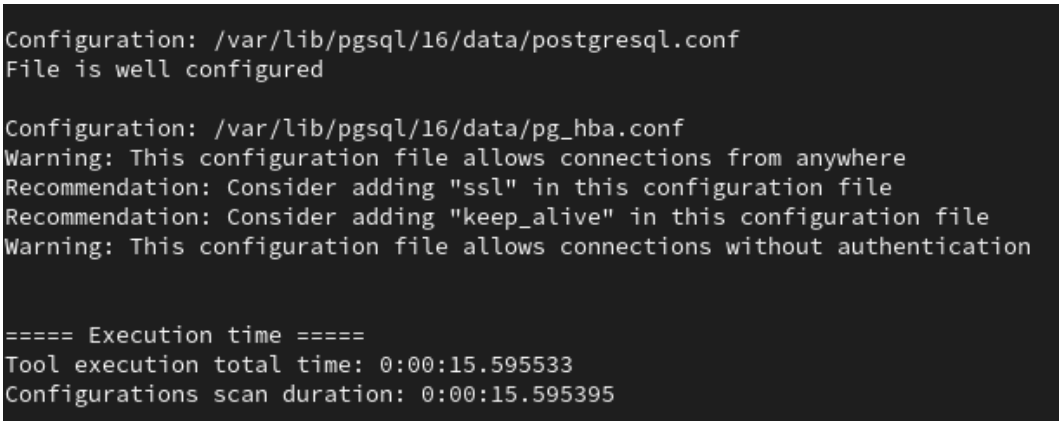


Figure 34: Tool Output after the changes in the files

Nftables Assessment

For the nftables the checks were simple. Firstly, it needs to be running at all costs so we implemented a health check in our tool were it will get the status of the service. If the status is inactive then it will report it, as depicted in figure 34.

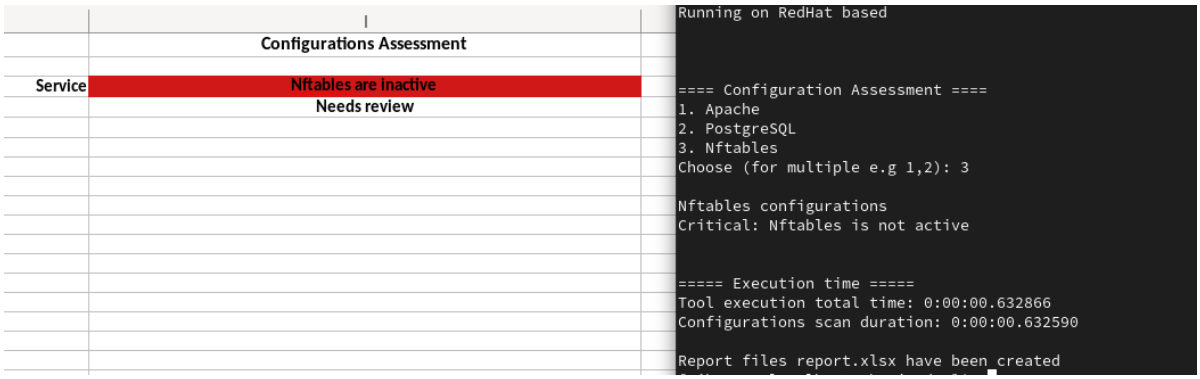


Figure 35: Results with inactive Nftables

Following our test, we started the nftables and added some but rules, specifically we allowed for every source IP to communicate with every destination IP, meaning that there is no filter in the firewall. Then we re-run our tool and got the following results in figure 35.

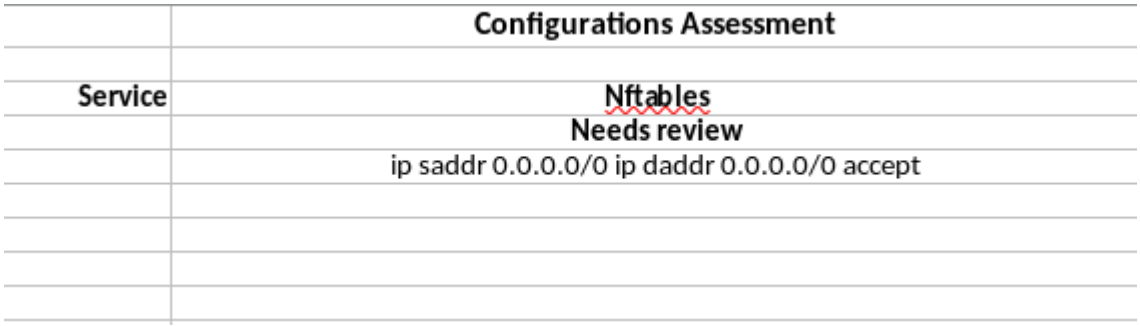


Figure 36: Reported rule from our tool

Our tool listed this rule for review, providing visibility to the system administrators and requesting for their actions.

4.2.3 Summarized and visualized results

In this subsection, we present a summary and visual representation of the key findings from our assessments and analyses. By synthesizing the results into clear and easily interpretable formats, we try to provide an overarching view of the outcomes. The use of charts, graphs, and tables not only facilitates a more comprehensive understanding of the data but also allows for more effective communication of the results. This section summarizes our findings, enabling readers to quickly grasp the core conclusions.

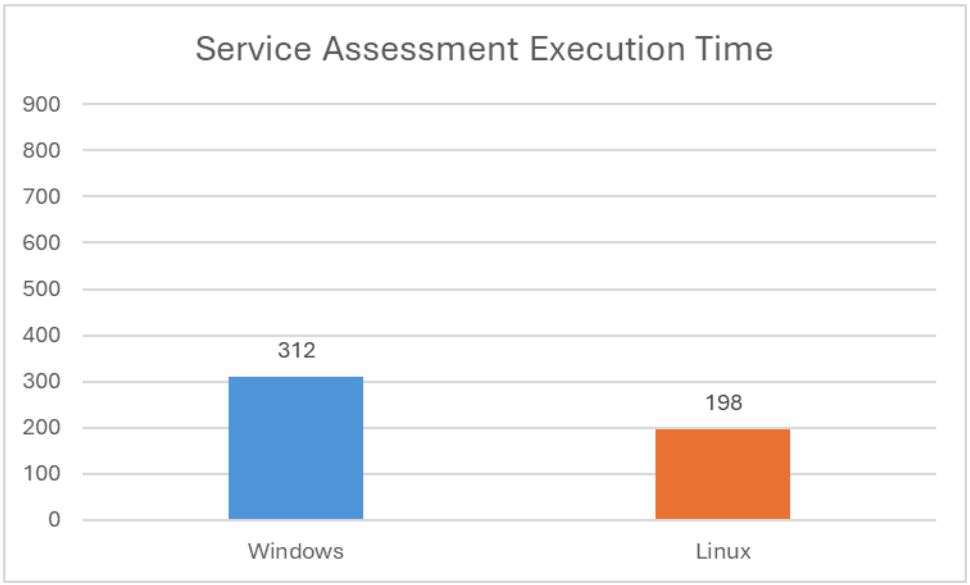


Figure 37: Service Assessment bar chart

The first bar chart (Figure 37) illustrates the average time taken by the services assessment module across different operating systems. The chart reveals that the assessment of services on Windows systems generally requires more time compared to Linux systems. This discrepancy can be attributed to the more complex and diverse nature of services running on Windows platforms, which often include a wider range of background processes and dependencies that need to be analyzed. In contrast, Linux services, being more modular and standardized, tend to be assessed more quickly.

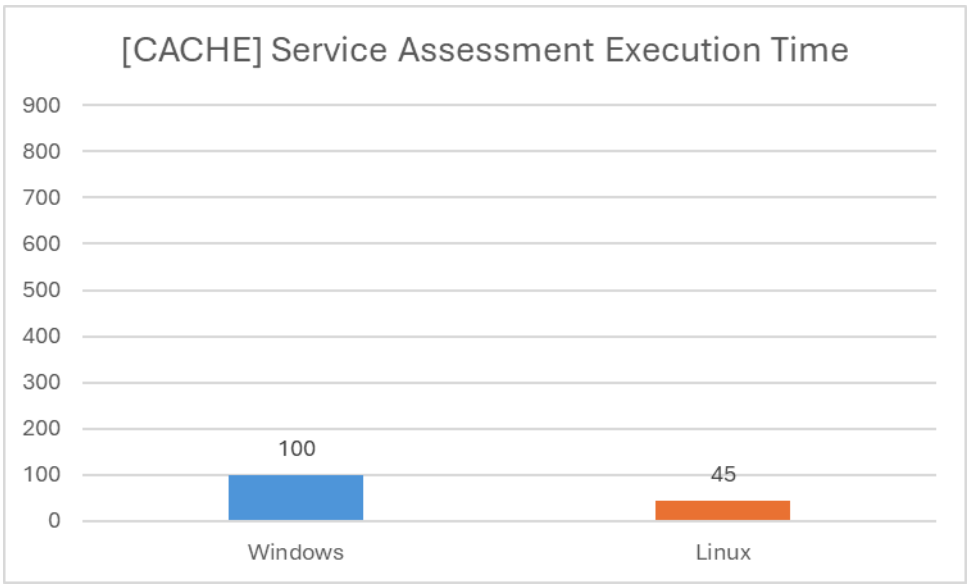


Figure 38: Service Assessment with cache bar chart

The second bar chart (Figure 38) displays the average time taken by the services with cache assessment module for both Windows and Linux systems. Similar to the previous chart, the assessment on Windows systems takes longer but overall, the average duration has dropped dramatically. These results suggest that optimizing cache analysis techniques could yield significant improvements in assessment times.

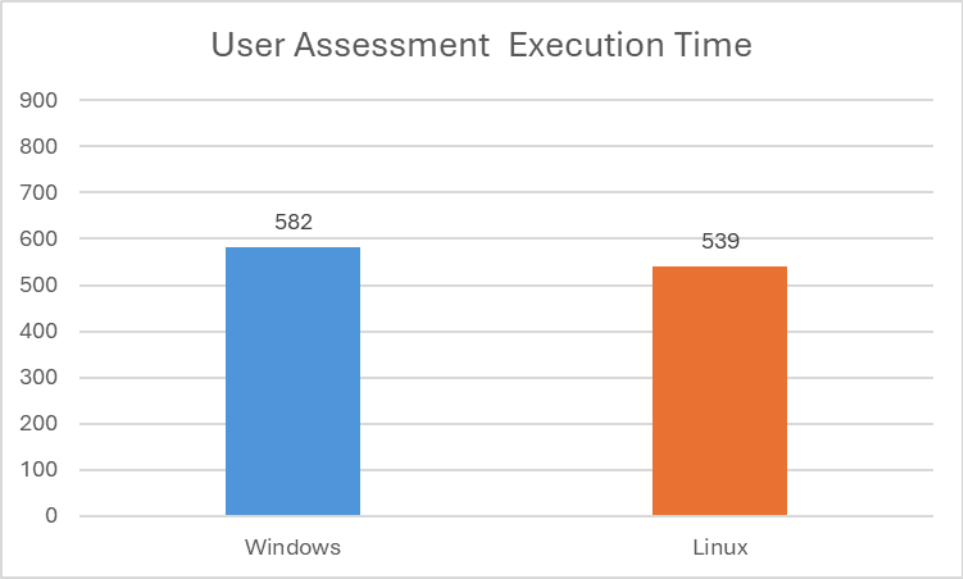


Figure 39:User Assessment bar chart

The third bar chart (Figure 39) shows the average time required for the users' assessment module across Windows and Linux systems. Interestingly, this chart highlights a more balanced performance between the two operating systems, with only a slight increase in assessment time for Windows. This indicates that the complexity of user authentication is relatively comparable between Windows and Linux environments. However, the slight edge in speed for Linux could still be linked to the more streamlined and transparent user authentication methods inherent to Unix-like systems.

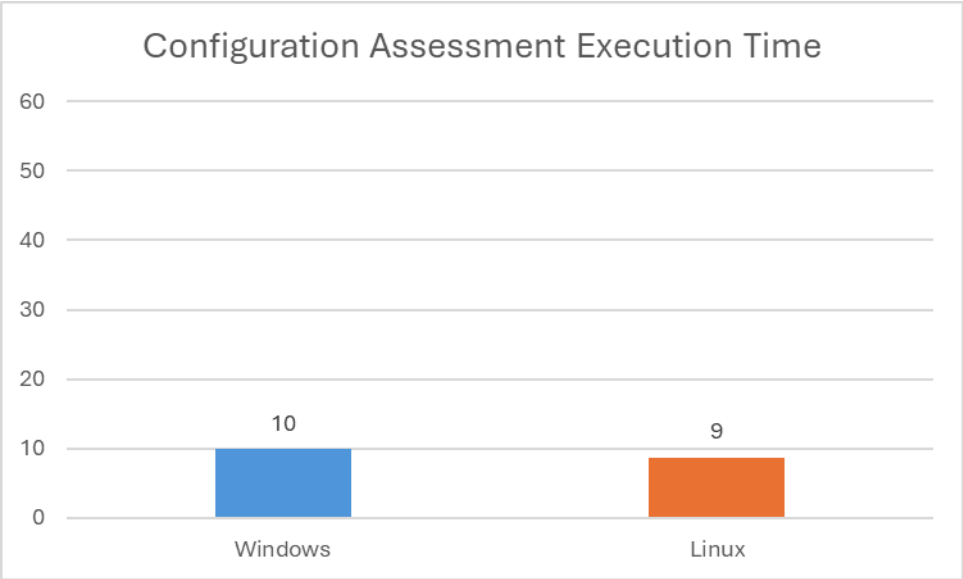


Figure 40: Configuration Assessment bar chart

The fourth bar chart (Figure 40) represents the average time taken by the configurations assessment module for Windows and Linux systems. Unlike the other modules, the assessment times for configurations are nearly identical for both operating systems. This surprising result indicates that, despite the inherent differences in how configurations are managed (with Windows using the registry and Linux using text-based configuration files), the complexity and effort required for thorough assessment are comparable. This suggests that our assessment tools are equally effective at parsing and evaluating configuration settings across both platforms, highlighting the adaptability of our methodology.

4.3 Conclusion

In summary, the results from our assessments across different modules reveal distinct patterns in performance between Windows and Linux operating systems, with notable similarities in the configurations and users assessment modules where the times are nearly identical. Generally, Windows systems, due to their inherent complexity and extensive configurations, require more time for the assessment, while the modular and transparent nature of Linux systems allows for more efficient analysis. These insights emphasize the need for specialized tools and techniques tailored to the specific characteristics of each operating system to enhance the efficiency and accuracy of vulnerability assessments.

Service Assessment Module: Although the current implementation effectively identifies vulnerable services, some false positives may occur due to the way service names are searched in the NIST database or due to the unavailability of the service version in the system. Future improvements could involve refining the search method to reduce false positives and incorporating additional CVE databases to broaden the scope and accuracy of vulnerability identification.

User Assessment Module: The assessment of user passwords and permissions is relatively efficient across both Windows and Linux systems. However, the process could be expedited by implementing offline password cracking methods. For Windows, this could involve dumping

passwords and performing an offline password crack. For Linux, an offline attack using the shadow file and encoding with yescript could significantly reduce assessment time while maintaining accuracy. On top of that, searching during the execution the high privileged groups would be a better way to identify the high privileged accounts.

Configurations Assessment Module: While the current tool checks for a range of configuration settings, there is potential for infinite expandability by adding more configurations to the checklist. This could include newer security benchmarks and best practices, ensuring the tool remains relevant and comprehensive over time.

By continuously refining our methodologies and tools, we can achieve more effective and timely evaluations, ultimately contributing to more secure and robust computing environments. The enhancements proposed above aim to improve the accuracy, speed, and scope of the assessment tool, ensuring it remains a valuable asset in the ongoing effort to maintain system security. This thesis helps us understand the differences between Windows and Linux assessments and opens the door for future improvements, making vulnerability assessments more efficient and thorough across different computing environments.

5 References

1. Contributor, T. (2019, February 19). vulnerability (information technology). WhatIs. https://www.techtarget.com/whatis/definition/vulnerability

2. Glossary of Security Terms | SANS Institute. (n.d.). https://www.sans.org/security-resources/glossary-of-terms/

3. Awati, R. (2023, November 3). *Common Vulnerabilities and Exposures (CVE)*. Security. https://www.techtarget.com/searchsecurity/definition/Common-Vulnerabilities-and-Exposures-CVE

4. Sheldon, R. (2023, April 14). Apache. WhatIs. https://www.techtarget.com/whatis/definition/Apache

5. Olsen, G. (2022, April 29). *Common Information Model (CIM)*. Storage. https://www.techtarget.com/searchstorage/definition/Common-Information-Model

6. Sheldon, R., & Bacon, M. (2024, January 29). *indicators of compromise (IOC)*. Security. https://www.techtarget.com/searchsecurity/definition/Indicators-of-Compromise-IOC

7. Bigelow, S. J. (2023, April 27). *operating system (OS)*. WhatIs. https://www.techtarget.com/whatis/definition/operating-system-OS

8. Bigelow, S. J., Moore, J., Jones, D., & Bertram, A. (2023, March 16). *What is PowerShell and how to use it: The ultimate tutorial*. SearchWindowsServer. https://www.techtarget.com/searchwindowsserver/definition/PowerShell

9. Stevewhims. (2021a, January 7). *Registry - Win32 apps*. Microsoft Learn. https://learn.microsoft.com/en-us/windows/win32/sysinfo/registry

10. Microsoft Learn. (2023, August 3). Windows Management Instrumentation - Win32 Apps | Microsoft Learn. https://learn.microsoft.com/en-us/windows/win32/wmisdk/wmi-start-page

11. JasonGerend. (2023, February 3). *PowerShell*. Microsoft Learn. https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/powershell

12. Yasar, K. (2023, August 23). Network vulnerability scanning. Security. https://www.techtarget.com/searchsecurity/definition/vulnerability-scanning

13. Hasson, E. (2023, December 20). What is Vulnerability Assessment | VA Tools and Best Practices|Imperva. Learning Center. https://www.imperva.com/learn/application-security/vulnerability-assessment/

14. What is a vulnerability assessment? and how to conduct one | UpGuard. (n.d.). https://www.upguard.com/blog/vulnerability-assessment

15. Wireless network vulnerability Scanning | CISA. (n.d.). Cybersecurity and Infrastructure Security Agency CISA. https://www.cisa.gov/resources-tools/services/wireless-network-vulnerability-scanning

16. Chauhan, A. S. (n.d.). Practical network scanning. O'Reilly Online Learning. https://www.oreilly.com/library/view/practical-network-scanning/9781788839235/27f6192b-3a22-4043-9fb8-eb782167f44d.xhtml

17. Chinnasamy, V. (2024, March 4). The Importance of Vulnerability Assessment: Types and Methodology. Indusface. https://www.indusface.com/blog/explore-vulnerability-assessment-types-and-methodology/

18. Roldán-Molina, G., Almache-Cueva, M., Silva-Rabadão, C., Yevseyeva, I., & Basto-Fernandes, V. (2017). A comparison of cybersecurity risk analysis tools. Procedia Computer Science, 121, 568–575. https://doi.org/10.1016/j.procs.2017.11.075

19. Khounborine, C. (n.d.). A survey and comparative study on vulnerability scanning tools. ScholarWorks@UARK. https://scholarworks.uark.edu/csceuht/124/

20. Dinis B. Cruz1, João R. Almeida1, And José L. Oliveira (2016). Open Source Solutions for Vulnerability Assessment: A Comparative Analysis

21. Hasson, E. (2023a, December 20). Vulnerability management | Patches & scanners vs input validation | Imperva. Learning Center. https://www.imperva.com/learn/application-security/vulnerability-management/

22. Chopskie, E. (2023, December 18). Vulnerability assessment tools: Key features and 5 tools you should know. Bright Security. https://brightsec.com/blog/vulnerability-assessment-tools-key-features-and-5-tools-you-should-know/

23. Kime, C. (2024, June 3). 6 Top Open-Source Vulnerability Scanners & Tools. eSecurity Planet. https://www.esecurityplanet.com/networks/open-source-vulnerability-scanners/

24. JasonGerend. (2023b, September 13). Credentials processes in Windows Authentication. Microsoft Learn. https://learn.microsoft.com/en-us/windows-server/security/windows-authentication/credentials-processes-in-windows-authentication#BKMK_LSA

25. Intelligence, M. T. (2024, May 2). *Detecting and preventing LSASS credential dumping attacks*. Microsoft Security Blog. https://www.microsoft.com/en-

us/security/blog/2022/10/05/detecting-and-preventing-lsass-credential-dumping-attacks/

26. Microsoft Learn. (2021, July 1). Operating System Availability of WMI Components - Win32 Apps | Microsoft Learn. https://learn.microsoft.com/en-us/windows/win32/wmisdk/operating-system-availability-of-wmi-components

27. Microsoft Learn. (2021, July 1). Managed Object Format (MOF) - Win32 Apps | Microsoft Learn. https://learn.microsoft.com/en-us/windows/win32/wmisdk/managed-object-format--mof-

28. Scripto, D. (2019, February 18). Should I use CIM or WMI with Windows PowerShell? Scripting Blog [Archived]. https://devblogs.microsoft.com/scripting/should-i-use-cim-or-wmi-with-windows-powershell/

29. CIS BenchmarksTM. (n.d.). CIS. https://www.cisecurity.org/cis-benchmarks

30. CIS CentOS Linux Benchmarks. (n.d.). CIS. https://www.cisecurity.org/benchmark/centos_linux

31. Stevewhims. (2021, January 7). Registry Hives - Win32 apps. Microsoft Learn. https://learn.microsoft.com/en-us/windows/win32/sysinfo/registry-hives

32. M. Burmester and P. Kotzanikolaou: *"Securing Networks against Extreme Attacks"*. In Volume of essays in honour of Professor Antonios C. Panayotopoulos, pp. 875–886, University of Piraeus, Greece, 2006.

33. C. Douligeris and P. Kotzanikolaou: *"Introduction to Network Security"***.** In "Network Security: Current Status and Future Directions", IEEE Press – Wiley Interscience, ISBN 978-0-471-70355-6, pp.1-12, April 2007.

34. Stallings, W., & Brown, L. (2023b). *Computer security: Principles and Practice*. Pearson Educational.