Assessment 2
Vulnerability Assessment and Remediation

EPT232 Ethical Hacking and Penetration Testing

Prepared by:
Smirti Parajuli (Student ID: MDS3000025)

Submitted to:
Sarada Hettiarachchi
Bachelor of Software Engineering (AI)
Media Design School

Date:
21st June 2025

1. Executive Summary

This report presents the outcomes of a structured vulnerability assessment carried out as part of a simulated penetration testing engagement for the EPT232 module. The objective of this exercise was to emulate real-world security assessment practices by identifying, analyzing, and proposing mitigations for potential vulnerabilities in a controlled enterprise-like environment.

The assessment consisted of three core components. First, a critical review of three publicly available penetration testing reports was conducted to extract best practices in vulnerability classification, risk prioritization, and remediation planning. These reports provided insight into professional testing methodologies and informed the structure and language of this document. Second, automated vulnerability scans were executed against a target web server located at internal IP address 10.6.6.13, using two open-source tools: Nikto, for detecting web server misconfigurations and known issues, and Greenbone Vulnerability Manager (GVM), for comprehensive system-level vulnerability analysis. Third, the findings were categorized by severity and synthesized into a professional-style penetration testing report.

This document is tailored for a technically proficient audience, including cybersecurity analysts, system administrators, and IT risk managers. It offers an evidence-based overview of the current security posture of the assessed environment and presents clearly defined remediation strategies to reduce risk exposure and improve overall system resilience.

2. Introduction

In an era of increasingly sophisticated and persistent cyber threats, conducting regular vulnerability assessments is essential for organizations aiming to safeguard their digital infrastructure. Proactive identification and mitigation of security flaws not only reduce the risk of exploitation but also support compliance with industry best practices and standards. This report presents a structured vulnerability assessment and penetration testing exercise conducted as part of the EPT232 module at Media Design School.

The primary objective of this assignment was to simulate a real-world penetration test, enabling the development of technical and analytical skills relevant to professional cybersecurity practice. The assessment involved three key components: (1) a critical review of professional penetration testing reports to extract methodological and reporting best practices; (2) the use of automated vulnerability scanning tools to evaluate a controlled testing environment; and (3) the formulation of evidence-based remediation strategies based on the findings.

The target of the assessment was a web server located at the internal IP address 10.6.6.13. Two widely adopted open-source tools were utilized for the scanning phase: Nikto, a lightweight tool for detecting common web server misconfigurations and vulnerabilities, and Greenbone Vulnerability Manager (GVM), a more comprehensive solution capable of identifying a broad range of system-level and network-based security issues.

This report is structured to provide a detailed account of the methodology, tool outputs, analytical findings, and prioritized recommendations for remediation. It is intended for an audience comprising IT security professionals, system administrators, and other technical stakeholders who are responsible for managing and securing networked systems. By emulating the structure and tone of industry-standard security assessments, this report also aims to bridge academic learning with real-world cybersecurity practice.

3. Methodology

This project was structured into three core phases: analysis of professional penetration testing reports, vulnerability scanning of a designated target server, and formal reporting of findings. The first phase involved a detailed review of three public penetration testing reports sourced from the GitHub repository Public Pentesting Reports by santosomar. These reports focused on CaseBox, Dovecot, and cURL were selected to represent a diverse range of systems, including web applications, protocol-level services, and widely-used open-source libraries. I conducted a comparative analysis of how each report identified vulnerabilities, assessed risk severity, and communicated remediation strategies. This allowed me to learn from professional security assessments and apply those principles to my own scanning and documentation.

In the second phase, I conducted active vulnerability scanning on the target server (IP: 10.6.6.13) using two industry-standard tools: **Nikto** for web-level misconfiguration detection, and **Greenbone Vulnerability Manager (GVM)** for in-depth system-wide vulnerability assessment. The scans were executed sequentially and tailored to detect outdated services, missing security headers, unpatched components, and configuration weaknesses. The findings from each tool were reviewed and cross-referenced, allowing me to prioritize vulnerabilities based on their severity level (e.g., High, Medium, Low), potential exploitability, and impact on the target system. In cases where the GVM scan was partially incomplete due to hardware constraints, the available results were still analyzed and validated for relevance.

The third and final phase consisted of compiling a formal penetration testing report. This included not only a summary of discovered vulnerabilities, but also technical explanations, assigned risk levels, and clearly defined remediation strategies. Screenshots from the scans and extracted outputs were appended to provide verifiable evidence. The report was written to reflect industry documentation standards, ensuring clarity, practical relevance, and readability for both technical and non-technical stakeholders. This structured methodology ensured that each stage analysis, scanning, and reporting was carried out with rigor, aligning with professional cybersecurity assessment workflows

4. Penetration Testing Report Analysis (Part 1)

This section presents an in-depth comparative analysis of three professional penetration testing reports published by Cure53, a well-respected cybersecurity consultancy known for its deep technical audits and clear documentation. The decision to analyze Cure53's reports was intentional: their work is often cited in academic literature, industry blogs, and security tool documentation due to their thorough methodology, realistic threat modeling, and focus on developer-oriented remediation. Compared to generic scan-based reports, Cure53 reports combine manual source code review, dynamic analysis, and exploitation attempts, offering richer insight into real-world security practices.

The CaseBox report was chosen in particular for several reasons. First, it focuses on a web-based open-source platform, an environment I'm familiar with through past coursework and personal projects. Web application vulnerabilities like XSS, SQL injection, and RCE are among the most common and impactful security issues, and this report provides numerous examples of such threats being successfully exploited. I had also encountered CaseBox in previous studies related to NGO and civic tech software, and was curious to understand its threat surface. Additionally, this report has been referenced in several reputable blogs and community forums as a model example of responsible disclosure and actionable reporting.

By selecting Cure53's reports on CaseBox, Dovecot, and cURL, I aimed to cover a spectrum of systems: a full-stack web application, a complex system-level mail server, and a highly portable client-side protocol tool. This allowed me to compare how different security concerns are identified and addressed across application layers from user-facing platforms to low-level libraries that form the backbone of network communication.

4.1 Report 1: CaseBox Web Application – Cure53

4.1.1 Overview:

The first penetration testing report reviewed is a 2014 security assessment of CaseBox, an open-source web application designed to support litigation-focused NGOs in managing sensitive documents, case records, and team workflows. Conducted over a 10-day period by Cure53, the assessment targeted not only the main application but also its demo environments, development VMs, and underlying source code. A total of 30 vulnerabilities were uncovered, including 6 Critical and 9 High-severity issues, highlighting serious architectural and coding oversights. The audit applied a hybrid black-box and white-box methodology, offering a realistic view of what attackers could exploit in externally hosted, widely deployed systems.

4.1.2 Vulnerability Identification and Categorization

Cure53 employed a comprehensive methodology that combined manual testing, source code auditing, and exploitation of live environments to uncover vulnerabilities in the CaseBox application. Each issue was clearly labeled (e.g., CB-01-001) and supported with proof-of-concept (PoC) links, source traces, and HTTP request samples. Among the most critical findings was CB-01-001, which enabled arbitrary file disclosure via preview.php, exposing sensitive files like config.ini that contained database credentials. CB-01-002 revealed a flawed password reset mechanism that relied on predictable MD5 hashes derived from user ID, email, and server time values that could be brute-forced using data from HTTP headers. Particularly alarming was CB-01-014, which allowed remote code execution through the AutoSetFields plugin by passing unvalidated user input directly to an eval() call. Multiple persistent cross-site scripting (XSS) vulnerabilities were also detected across folder names, user profile fields, and uploaded SVG paths. Additionally, CB-01-029 exposed a high-risk second-order SQL injection in the getUserData() function, which could be leveraged to extract or manipulate database records. Each vulnerability was documented with realistic attack vectors and contextual impact, underscoring their exploitability in real-world deployments.

4.1.3 Risk Assessment and Prioritization Methodology

The CaseBox report employed a severity rating system that considered three main factors: the impact of the vulnerability (such as data leakage or remote code execution), the ease of exploitation, and the scope of potential damage whether it affected individual users or the entire system. Although formal CVSS scores were not assigned, each issue was clearly categorized as Critical, High, Medium, or Low, with accompanying justifications. For instance, critical vulnerabilities like arbitrary file disclosure or remote code execution posed threats to the entire platform's integrity and confidentiality. High-risk cross-site scripting (XSS) vectors had the potential to compromise session tokens or allow full account hijacking. What stood out in this methodology was its practical and grounded approach to risk assessment; rather than relying solely on theoretical models, the report focused on real-world exploitability, often demonstrated through working attacks on publicly available demo environments. This made the prioritization both credible and immediately actionable.

4.1.4 Remediation Recommendation Clarity and Actionability

Cure53's remediation recommendations in the CaseBox report were notably clear, actionable, and grounded in secure development practices. The suggestions included applying proper input sanitization techniques specifically using functions like htmlentities() and libraries such as HTMLPurifier to mitigate XSS vulnerabilities. They also advised replacing weak password hashing algorithms like MD5 with stronger, modern alternatives such as bcrypt. Insecure default SSL configurations were flagged for correction, and directory traversal risks were addressed by recommending structural changes to isolate preview logic from sensitive file paths. One of the standout aspects of this report was its iterative nature: several critical issues were not only identified but also patched during the course of the audit, with Cure53 confirming the effectiveness of the fixes. This real-time feedback loop exemplifies what a high-quality penetration test should deliver. Additionally, the report cited reputable tools and best practices including SSL Labs and Duraconf to guide developers toward long-term security resilience.

4.1.5 Reflections & Valuable Takeaways

What stood out to me most in the CaseBox report was the remarkable depth of exploitation Cure53 didn't just highlight obvious flaws but demonstrated how even subtle logic errors, like time-based password reset tokens, could be effectively weaponized. The combination of black-box testing and white-box auditing gave the report a uniquely comprehensive feel, blending real-world attacker perspectives with deep technical scrutiny. What I particularly appreciated was the empathetic tone of the remediation advice: every vulnerability was not only explained, but paired with clear and practical solutions that showed respect for the developer's workflow. Through this report, I learned the value of thinking like an attacker leveraging vectors like path traversal, SVG uploads, and CSRF chaining as well as the importance of constantly questioning assumptions, such as trusting file extensions or input formats. Just as crucially, it reinforced the need to communicate security findings with precision, structure, and respect. Overall, this report set a high standard for what professional penetration testing should achieve: thoroughness, ethical conduct, technical rigor, and actionable guidance.

4.2 Report 2: Dovecot Mail Server – Cure53

4.2.1. Overview

This second report evaluates the Cure53 security audit of Dovecot, a widely adopted open-source IMAP and POP3 mail server for UNIX/Linux systems. Commissioned by Mozilla, the audit was conducted over a 20-day period in October–November 2016 and focused on version 2.2.26.0. The assessment covered critical components including protocol implementations, authentication mechanisms, encryption APIs, and the SSL wrapper. Notably, only three low-severity vulnerabilities were identified, a rare outcome for such a large and complex codebase. This result reflects Dovecot's security-conscious architecture, mature development lifecycle, and strong commitment to secure coding practices.

4.2.2. Vulnerability Identification and Categorization

Cure53 employed a two-phase methodology comprising manual source code auditing and code-assisted penetration testing on live Dovecot services. The scope included the POP3 and IMAP protocol stacks, login architecture, SSL certificate handling, MySQL and LDAP authentication plugins, the crypt encryption module, and inter-process socket communication systems. Despite this broad surface area, only three security-relevant issues were found. The first, DOV-01-001, was a format string protection bypass due to insufficient pattern detection in the printf_format_fix_noalloc() function, allowing potentially unsafe specifiers like %1$n. The second, DOV-01-002, involved missing compiler hardening flags in the Makefile, such as -fstack-protector and -D_FORTIFY_SOURCE=2, which are essential for secure binary compilation. The third, DOV-01-003, pointed to an integer overflow risk in the memory allocator, stemming from the lack of boundary checks in array allocations. Each issue was clearly documented with file references, technical explanations, and rationales for its classification.

4.2.3. Risk Assessment and Prioritization Methodology

Rather than emphasizing exploit-ready flaws, this audit framed its risk assessment through the lens of proactive architecture and sustainable coding practices. All three vulnerabilities were categorized as low-severity but were flagged for their potential to introduce memory corruption or weaken future-proof defenses. Cure53 stressed that while these issues may not present immediate exploitability, they weaken the overall resilience of the software and should be addressed to uphold Dovecot’s high security standards. The report’s emphasis on defensive programming, compiler-level mitigation, and long-term security posture reinforces that low-risk does not mean low-priority, especially in mission-critical infrastructure like mail servers.

4.2.4. Remediation Recommendation Clarity and Actionability

Cure53’s remediation guidance was technically sound and clearly actionable for developers. For DOV-01-001, the recommendation was to enhance the printf_format_fix_noalloc() function to recognize patterns like %1$n, not just %n. For DOV-01-002, the report suggested updating the build configuration to include full stack protections, such as -pie -fPIE -fstack-protector-all -D_FORTIFY_SOURCE=2 -O1, which enable RELRO, PIE, and stack canaries. Lastly, for DOV-01-003, the team recommended integrating arithmetic safety checks using compiler intrinsics like builtin_mul_overflow() or adopting memory-safe allocation wrappers akin to PHP's safe_emalloc(). These recommendations were well-contextualized, easy to implement, and incurred minimal performance overhead demonstrating a thoughtful balance between security and usability

4.2.5 Reflections & Valuable Takeaways

This audit provided important insights into what secure software engineering looks like, even in the absence of high-impact vulnerabilities. It revealed how well-designed systems like Dovecot can resist exploitation through architectural foresight, layered memory protections, and strict development practices. Several technical practices stood out, including pointer nulling, memory zeroing, secure allocation routines, and protocol fuzz testing to verify robustness. The report also highlighted how even minor flaws such as missing hardening flags can accumulate into risk over time if left unaddressed. For me, this emphasized that security isn't only about responding to current threats but about proactively reinforcing software to withstand future ones. Ultimately, the Dovecot audit demonstrated that rigorous, independent security assessments remain valuable, even for projects with strong internal discipline, by uncovering overlooked details and strengthening long-term code resilience.

4.3 Report 3: cURL & libcurl – Cure53 Security Audit

4.3.1. Overview

The third report in this analysis reviews the 2022 penetration test and secure code audit of cURL and libcurl, conducted by Cure53. Commissioned by the Mozilla Open Source Support (MOSS) Program, the assessment spanned 20 days in September 2022. cURL is a foundational component of modern computing, embedded in billions of devices from Linux distributions and IoT products to mobile applications making its security posture vital to global infrastructure.

The audit focused on version 7.83.1, with particular emphasis on libcurl, the core library that enables developers to incorporate data transfer protocols into software. The test revealed only two security-related issues, reflecting the maturity, robustness, and well-maintained nature of the codebase..

4.3.2. Vulnerability Identification and Categorization

Cure53 employed a hybrid assessment methodology that combined manual source code analysis with guided fuzz testing. Their evaluation focused on critical areas of the cURL/libcurl codebase, including HTTP/FTP parser logic, SSL/TLS bindings (such as OpenSSL and GnuTLS), DNS resolution mechanisms, proxy handling protocols (e.g., SOCKS and HTTP CONNECT), and the APIs used for cookie and credential storage. The audit revealed two security-relevant issues. The first, CURL-01-001, involved insecure reliance on environment variables (CURL_CA_BUNDLE and CURL_SSL_BACKEND), which could be manipulated in restricted or containerized environments that were classified as medium severity. The second, CURL-01-002, related to incomplete memory cleanup, allowing sensitive authentication credentials to persist in memory beyond their necessary lifespan classified as low severity. Although neither issue was directly exploitable (e.g., through remote code execution), both highlighted subtle but meaningful implementation weaknesses in an otherwise mature codebase.

4.3.3. Risk Assessment and Prioritization Methodology

The audit emphasized systemic risk factors that could manifest under specific deployment conditions. Key concerns included assumptions about environmental control particularly in sandboxed or containerized environments, risks of data leakage due to improper memory handling, and the potential for abuse of runtime configurations through untrusted environment variables. The medium-severity issue (CURL-01-001) stemmed from the possibility of untrusted users influencing SSL trust chains or cryptographic behavior via manipulated environment variables. The low-severity issue (CURL-01-002) focused on the residual presence of sensitive authentication data in memory due to incomplete cleanup. While neither flaw was directly exploitable in typical use cases, both findings underscored the presence of subtle risks in edge-case or hardened environments, reinforcing the need for defensive coding even in highly trusted libraries.

4.3.4. Remediation Recommendation Clarity and Actionability

Cure53 provided clear and technically actionable remediation strategies for both identified vulnerabilities. To address CURL-01-001, the recommendation was to harden environment variable usage by validating the source and restricting override permissions such as enforcing that only root-owned sessions can modify trust paths. For CURL-01-002, the proposed solution involved implementing secure memory wiping procedures to ensure that authentication credentials are zeroed out from memory immediately after use. Both issues were addressed and patched promptly by the cURL maintainers, demonstrating not only the maturity of the project's governance but also the responsiveness of its security practices.

4.3.5 Reflections & Valuable Takeaways

This audit illustrated that even long-established and widely deployed projects like cURL are not immune to edge-case vulnerabilities. The findings reinforce several critical lessons. First, reliance on environmental defaults such as assuming the safety of runtime configuration variables can introduce hidden risks, particularly in restricted execution environments. Second, memory safety is not limited to preventing crashes or buffer overflows; it also encompasses the responsible handling of sensitive data to maintain confidentiality. Finally, the true value of a security audit lies not only in the volume of issues discovered but in its ability to reinforce secure coding practices and elevate system resilience from “good enough” to “zero-trust ready.” This report stands as a testament to how focused and methodical testing can lead to meaningful improvements, even in projects with strong security foundations.

4.4 Comparison of All Three Reports

This section provides a structured comparison of three professional penetration testing reports conducted by Cure53, each assessing a different system with distinct security challenges. The first report evaluates CaseBox, a web-based document management platform; the second examines Dovecot, a secure IMAP/POP3 email server; and the third analyzes cURL, a popular command-line tool and networking library.

The comparison focuses on key elements such as vulnerability discovery techniques, severity assessments, remediation strategies, and reporting clarity. Each system presented varying levels of complexity, risk exposure, and technical maturity, offering diverse perspectives on secure software development practices. As summarized in Table 4, CaseBox exhibited the highest number of critical vulnerabilities, largely due to its broad web application surface. In contrast, Dovecot demonstrated resilience, with only low-severity findings related to memory handling. The cURL audit revealed only minimal issues, reflecting its mature, well-hardened codebase.

Table Comparison of Key Characteristics Across Reports

Aspect	CaseBox (Web App)	Dovecot (Mail Server)	cURL (Protocol Tool)
# of Findings	30 total (6 Critical, multiple High)	3 Low-severity issues	2 issues (Medium & Low)
Testing Scope	Full-stack: code, live server, sandbox VM	POP/IMAP, dcrypt, SSL, login stack	Build config, memory handling, environment use
Methodology	Static + dynamic testing, code + VM execution	Manual code audit + code-assisted testing	Code review + fuzzing + threat modeling
Risk Focus	Active exploitation: RCE, XSS, SQLi, CSRF	Memory safety, compiler hardening	Environmental misuse, memory hygiene
Remediation Style	Highly actionable (patches suggested directly)	Developer-focused (low-level C fixes)	Environmental + usage hygiene fixes

Clarity & Structure	Very clear, numbered findings, real examples	Technical, thorough, fewer findings	Minimal but precise, tailored to devs
Maturity of Codebase	Early-stage app, many issues	High maturity, very few bugs	High maturity, secure defaults
Value Gained	Common web app mistakes and mitigation tactics	Depth in system-level hardening	Insight into safe defaults & edge-case risks

4.4.1. Comparative Analysis

1. **Vulnerability Identification & Categorization:**

CaseBox revealed the highest volume and severity of vulnerabilities, including remote code execution, SQL injection, and persistent XSS. The report used consistent formatting, clearly labeled findings, and actionable categorizations. In contrast, Dovecot's audit identified only three minor issues, with a strong focus on code safety and secure memory handling. cURL's assessment was even more minimal, highlighting just two vulnerabilities related to environmental hygiene and memory cleanup.

2. **Risk Prioritization and Impact Assessment**

The CaseBox report took a threat-centric approach, prioritizing risks based on real-world exploitability and business impact. Dovecot and cURL, however, placed emphasis on proactive resilience, focusing on compiler flags, hardened defaults, and architecture-level protections. While their findings had lower practical exploitability, they reflected deeper engineering maturity.

3. **Remediation Clarity and Actionability**

All three reports were tailored to their respective development teams. CaseBox offered immediate fixes (e.g., CSRF protection, input sanitization), Dovecot recommended C-level memory safety improvements, and cURL emphasized configuration and environmental hygiene. Each report effectively addressed its audience from web app developers to systems programmers.

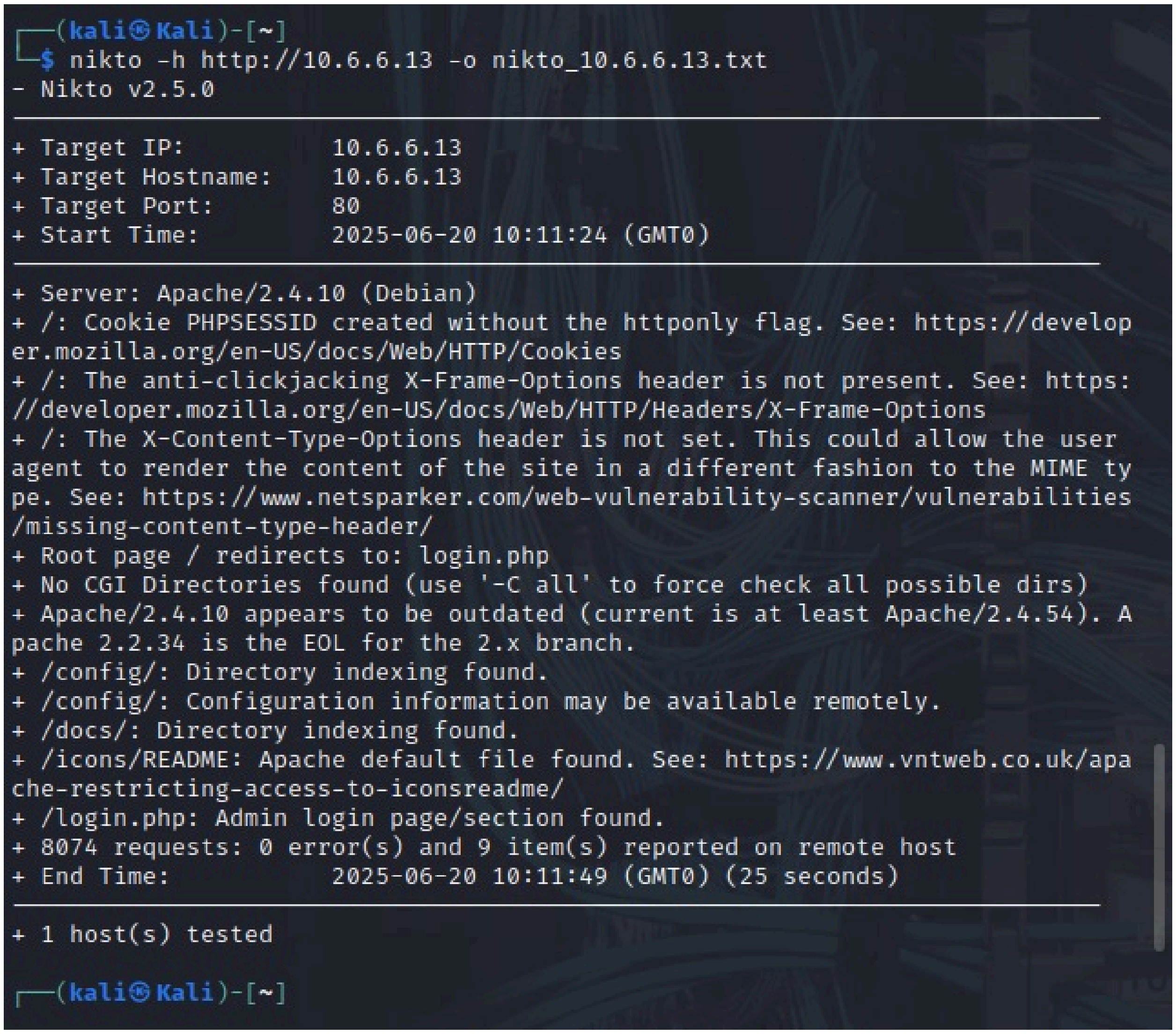
4. **Overall Value and Insights**

This comparison demonstrates the varying security needs of different software types. CaseBox underscored the risks of insufficient input validation in early-stage applications. Dovecot highlighted the power of secure-by-design development and long-term codebase resilience. cURL reinforced the importance of clean build environments, defensive memory practices, and secure defaults. Together, the three assessments provide a well-rounded perspective on penetration testing strategies and the evolving landscape of software security best practices.

5. Vulnerability Findings (Part 2)

5.1 Nikto Scan: Vulnerability Identification and Analysis

A vulnerability scan was conducted on the target server (IP: 10.6.6.13) using Nikto v2.5.0. The scan identified multiple web server misconfigurations and outdated components that could increase the likelihood of exploitation. These issues are prioritized based on severity and exploitability and are summarized in Table 1. in the table below and are prioritized based on severity and likelihood of exploitation.



```
┌──(kali㉿Kali)-[~]
└─$ nikto -h http://10.6.6.13 -o nikto_10.6.6.13.txt
- Nikto v2.5.0
---------------------------------------------------------------------------
+ Target IP:          10.6.6.13
+ Target Hostname:    10.6.6.13
+ Target Port:        80
+ Start Time:         2025-06-20 10:11:24 (GMT0)
---------------------------------------------------------------------------
+ Server: Apache/2.4.10 (Debian)
+ /: Cookie PHPSESSID created without the httponly flag. See: https://develop
er.mozilla.org/en-US/docs/Web/HTTP/Cookies
+ /: The anti-clickjacking X-Frame-Options header is not present. See: https:
//developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options
+ /: The X-Content-Type-Options header is not set. This could allow the user
agent to render the content of the site in a different fashion to the MIME ty
pe. See: https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities
/missing-content-type-header/
+ Root page / redirects to: login.php
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ Apache/2.4.10 appears to be outdated (current is at least Apache/2.4.54). A
pache 2.2.34 is the EOL for the 2.x branch.
+ /config/: Directory indexing found.
+ /config/: Configuration information may be available remotely.
+ /docs/: Directory indexing found.
+ /icons/README: Apache default file found. See: https://www.vntweb.co.uk/apa
che-restricting-access-to-iconsreadme/
+ /login.php: Admin login page/section found.
+ 8074 requests: 0 error(s) and 9 item(s) reported on remote host
+ End Time:           2025-06-20 10:11:49 (GMT0) (25 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested

┌──(kali㉿Kali)-[~]
```

***Figure 1**: Nikto Scan Output for 10.6.6.13*

Table 1: Summary of Web Server Vulnerabilities Identified by Nikto including their severity, potential impact, recommended remediations, and reference sources.

Vulnerability Description	Priority	Potential Impact	Recommended Mitigation	References
Outdated Apache version (2.4.10	High	May contain known vulnerabilities. Could be exploited for remote code execution or privilege escalation.	Upgrade to the latest stable version of Apache.	Apache HTTP Server
Lack of HttpOnly flag on PHPSESSID cookie	Mediu m	Increases risk of client-side script access via XSS attacks.	Set HttpOnly attribute for session cookies.	MDN Cookies
Missing X-Frame-Options header	Mediu m	Can allow MIME-type sniffing attacks.	Add X-Frame-Options : SAMEORIGIN or use Content-Security-Policy.	OWASP Clickjacking
Missing X-Content-Type-Options header	Mediu m	Exposes sensitive configuration or documentation files.	Set header X-Content-Type-Options: nosniff.	MDN nosniff
Directory indexing enabled (/config/, /docs/)	High	Exposes sensitive configuration or documentation files.	Disable directory listing or restrict access via .htaccess.	Apache Config Docs
Presence of Apache default README file	Low	Reveals server software details and documentation paths.	Remove or restrict public access to default files.	VNTWeb Advisory
No CGI directories	Info	Not a vulnerability	Run additional	

found" with a cleaner explanation in the table		but indicates lack of CGI scanning.	scans if CGI functionality is expected.	—

5.1.1 Analysis and Recommendations

The results of the Nikto scan indicate that the target server is affected by multiple web server misconfigurations and security oversights that significantly increase its vulnerability to exploitation. Most notably, the server is running Apache version 2.4.10, which is outdated and no longer maintained with security patches. This version is associated with several publicly disclosed Common Vulnerabilities and Exposures (CVEs) that could enable remote code execution, privilege escalation, or denial-of-service attacks. Additionally, the absence of critical HTTP security headers such as X-Frame-Options, X-Content-Type-Options, and the HttpOnly flag on session cookies exposes the server to various client-side threats including cross-site scripting (XSS), clickjacking, and MIME-type sniffing. The scan further identified that directory indexing is enabled on sensitive paths (e.g., /config/, /docs/) and that default files like README are publicly accessible, potentially disclosing internal information useful to attackers. To address these risks, it is recommended to upgrade Apache to the latest supported version, enforce HTTP header security policies through .htaccess or configuration files, disable directory listing, restrict access to default content, and adopt a structured patch management process. Implementing these remediations would substantially improve the server's security posture and minimize its exposure to both automated and targeted attacks.

5.2 GVM (Greenbone Vulnerability Manager) Scan and Vulnerability Analysis

As part of the second task in this project, an automated vulnerability scan was conducted using Greenbone Vulnerability Manager (GVM). The objective was to assess the security posture of the target server (10.6.6.13) and identify exploitable weaknesses. Several critical and high-severity issues were identified, indicating both system-level and web application-level risks. These findings are summarized below and are prioritized based on severity and the likelihood of exploitation.

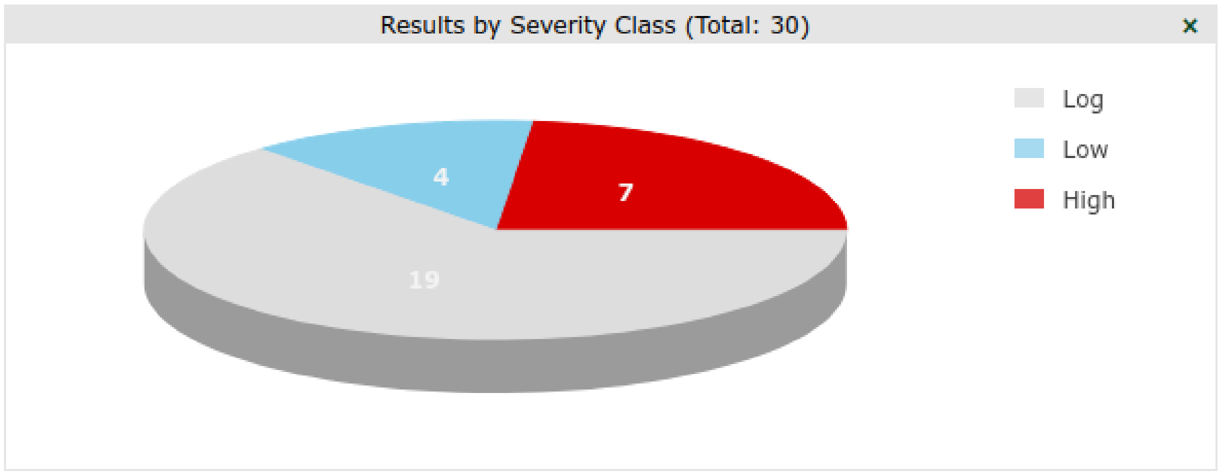


Figure 2: *Results by Severity Class*

Figure 2 presents a pie chart illustrating the severity distribution of the 30 vulnerabilities identified. Seven were classified as high severity, signaling critical risks with the potential to compromise system confidentiality, integrity, or availability. Four findings were considered low severity, representing minor or informational-level issues. The remaining nineteen were log-level alerts, which, while not immediately dangerous, still provide insight into potential misconfigurations or system behaviors worth addressing. The distribution underscores that approximately one-quarter of the vulnerabilities warrant immediate attention.

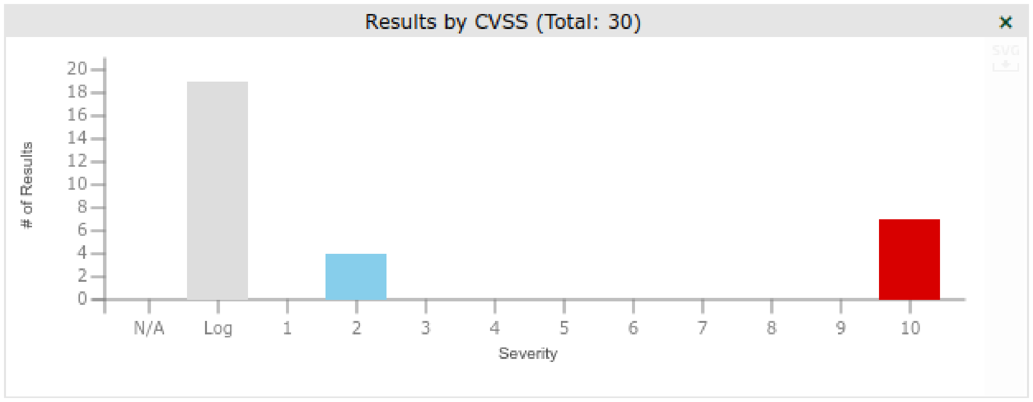
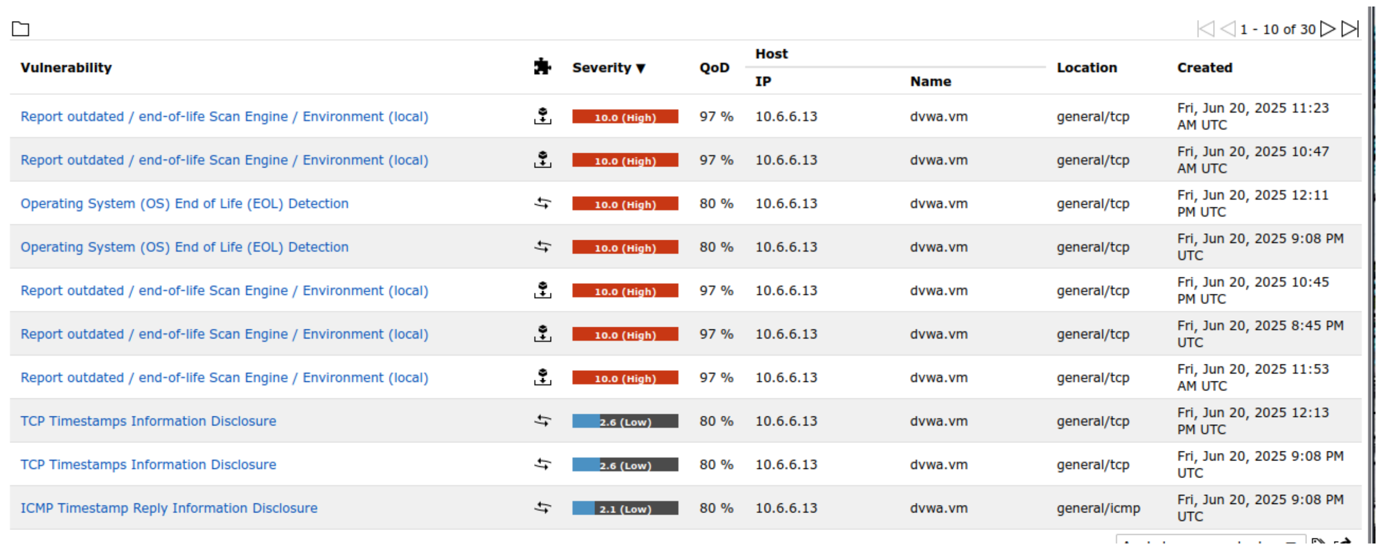


Figure 3: *Results by CVSS Score*

Figure 3 depicts the vulnerability distribution based on CVSS (Common Vulnerability Scoring System) ratings. Notably, several vulnerabilities scored a CVSS of 10 the maximum severity indicating they are both exploitable and capable of causing severe impact if left unaddressed. The presence of these high-scoring vulnerabilities highlights the urgency for prompt remediation efforts.



1 - 10 of 30

Vulnerability	Severity ▼	QoD	Host IP	Host Name	Location	Created
Report outdated / end-of-life Scan Engine / Environment (local)	10.0 (High)	97 %	10.6.6.13	dvwa.vm	general/tcp	Fri, Jun 20, 2025 11:23 AM UTC
Report outdated / end-of-life Scan Engine / Environment (local)	10.0 (High)	97 %	10.6.6.13	dvwa.vm	general/tcp	Fri, Jun 20, 2025 10:47 AM UTC
Operating System (OS) End of Life (EOL) Detection	10.0 (High)	80 %	10.6.6.13	dvwa.vm	general/tcp	Fri, Jun 20, 2025 12:11 PM UTC
Operating System (OS) End of Life (EOL) Detection	10.0 (High)	80 %	10.6.6.13	dvwa.vm	general/tcp	Fri, Jun 20, 2025 9:08 PM UTC
Report outdated / end-of-life Scan Engine / Environment (local)	10.0 (High)	97 %	10.6.6.13	dvwa.vm	general/tcp	Fri, Jun 20, 2025 10:45 PM UTC
Report outdated / end-of-life Scan Engine / Environment (local)	10.0 (High)	97 %	10.6.6.13	dvwa.vm	general/tcp	Fri, Jun 20, 2025 8:45 PM UTC
Report outdated / end-of-life Scan Engine / Environment (local)	10.0 (High)	97 %	10.6.6.13	dvwa.vm	general/tcp	Fri, Jun 20, 2025 11:53 AM UTC
TCP Timestamps Information Disclosure	2.6 (Low)	80 %	10.6.6.13	dvwa.vm	general/tcp	Fri, Jun 20, 2025 12:13 PM UTC
TCP Timestamps Information Disclosure	2.6 (Low)	80 %	10.6.6.13	dvwa.vm	general/tcp	Fri, Jun 20, 2025 9:08 PM UTC
ICMP Timestamp Reply Information Disclosure	2.1 (Low)	80 %	10.6.6.13	dvwa.vm	general/icmp	Fri, Jun 20, 2025 9:08 PM UTC

Figure 4: *Summary Table of Detected Vulnerabilities*

The table below summarizes major findings from the GVM scan. Each entry includes descriptions, associated risks, recommended mitigations, and relevant reference sources.

Vulnerability Description	**Priority**	**Potential Impact**	**Recommended Mitigation**	**References**
End-of-life Scan Engine Detected (local) – Report 1	High	Legacy environments may contain unpatched critical vulnerabilities.	Upgrade to a supported version of the scan engine or replace the system**.**	Greenbone Feed
End-of-life Scan Engine Detected (local) – Report 2	High	Systems may lack updates and support, increasing exploitability.	Transition to actively maintained software versions	CVE Database / GVM Feed
Operating System (OS) End of Life (EOL) Detection – 1	High	Unsupported OS could be targeted using known exploits.	Upgrade to a supported and updated OS version.	OS Vendor Docs
Operating System (OS) End of Life (EOL) Detection – 2	High	Leaves server exposed to privilege escalation and remote code execution.	Apply OS-level patches or migrate to a newer OS version.	CVE Database
End-of-life Scan Engine Detected	High	Increased likelihood of exploitation due to	Replace the outdated scan	Greenbone Feed

(local) – Report 3		outdated scanning environment.	engine with a maintained version.	
End-of-life Scan Engine Detected (local) – Report 4	High	May introduce platform-level vulnerabilities into broader system infrastructure.	Decommission legacy components and deploy secure alternatives.	CVE Database
End-of-life Scan Engine Detected (local) – Report 5	High	May enable attackers to bypass security controls.	Review system architecture and replace deprecated components.	Vendor Security Advisories
TCP Timestamps Information Disclosure – Report 1	Low	Enables OS fingerprinting and potential attack planning	Disable TCP timestamps in kernel settings.	RFC 1323
TCP Timestamps Information Disclosure – Report 2	Low	Minor info disclosure, could assist in uptime estimation.	Adjust sysctl or firewall to suppress timestamp responses.	GVM Detection Info
ICMP Timestamp Reply Information Disclosure	Low	Can assist attackers in mapping network latency and device availability.	Filter ICMP timestamp replies using a firewall	RFC 792

These vulnerabilities illustrate the risks posed by unmaintained software environments and insufficient configuration hardening. The following subsection offers a detailed analysis and remediation strategy based on the GVM scan results.

5.2.1 Analysis and Recommendations

The Greenbone Vulnerability Manager (GVM) scan identified several high-risk vulnerabilities primarily related to deprecated services and unsupported operating system environments. These outdated components significantly increase the likelihood of successful exploitation due to the lack of ongoing security updates and the presence of publicly documented CVEs. The repeated detection of end-of-life scan engines and operating systems points to broader systemic issues in software lifecycle management and patch governance. In addition to these critical findings, the scan revealed low-severity information disclosure vulnerabilities, including TCP and ICMP timestamp responses. While these do not pose immediate threats on their own, they can aid attackers in conducting reconnaissance or crafting timing-based exploits when used alongside more critical vulnerabilities.

Based on these observations, several key remediation strategies are recommended. First, the affected systems should be migrated to supported operating systems that receive regular vendor updates. Second, any deprecated services or outdated scanning components must be replaced with secure and actively maintained alternatives to reduce the attack surface. Network configurations should also be updated to suppress timestamp responses, thereby limiting exposure to passive reconnaissance techniques. Furthermore, regular vulnerability scanning and patch auditing should be integrated into the system's maintenance routine to ensure ongoing threat visibility and proactive risk mitigation.

It is important to acknowledge that the GVM scan was not fully completed due to hardware limitations on the host machine. Several scan segments were either interrupted or left pending, which may have resulted in undetected vulnerabilities that would have emerged under extended scan duration or improved computational resources. Nonetheless, the available scan results were sufficiently detailed to support a meaningful and actionable assessment of the system’s current security posture, and they provide a solid foundation for implementing an effective remediation plan.

5.3 Comparative Analysis of Nikto and GVM Scan Results

To gain a comprehensive understanding of the security posture of the target server (10.6.6.13), two complementary vulnerability assessment tools were employed: Nikto and Greenbone Vulnerability Manager (GVM). Each tool offers unique strengths and analytical depth, contributing to a more holistic evaluation of system vulnerabilities across different layers.

Nikto is a lightweight, command-line-based web server scanner designed for rapid detection of common web application misconfigurations. It excels in identifying issues at the application layer, such as outdated web server software, missing HTTP security headers, directory indexing, and the presence of publicly accessible files like README or configuration documentation. Its simplicity and speed make it ideal for quickly identifying surface-level issues that attackers might leverage during initial reconnaissance.

In contrast, GVM is an enterprise-grade vulnerability management platform that performs in-depth scans of the host system. It leverages the Greenbone Security Feed, incorporates Common Vulnerabilities and Exposures (CVE) identifiers, and provides detailed CVSS-based scoring to assess risk. GVM’s broader scope includes operating system vulnerabilities, network protocol weaknesses, and deprecated services or software components. It also provides a graphical user interface for advanced report generation and visualization, making it well-suited for professional security assessments.

The findings from both tools reflect their respective strengths. Nikto primarily identified web-layer issues, such as the use of an outdated Apache version (2.4.10), absence of essential security headers like X-Content-Type-Options and X-Frame-Options, exposed directories (/config/, /docs/), and publicly accessible default files. These represent typical application-layer misconfigurations that can expose the server to client-side attacks and information disclosure.

GVM, on the other hand, revealed deeper host-level vulnerabilities. It detected multiple instances of end-of-life operating systems and scanning engines, which introduced significant risks due to the absence of ongoing security support. Furthermore, GVM highlighted information disclosure vectors such as TCP and ICMP timestamp responses, which, although low in severity, can assist attackers in network mapping or timing-based exploits. Warnings related to deprecated scan engines and legacy software components further emphasized the need for comprehensive system upgrades and proper patch management.

While Nikto provides a rapid, low-overhead view of application security misconfigurations, GVM delivers a more in-depth and structured analysis of systemic weaknesses, complete with severity classification, CVE mapping, and remediation guidance. Together, these tools complement each other by covering both surface-level and systemic vulnerabilities. Their combined usage in this assessment enabled a more nuanced and thorough understanding of the overall threat landscape affecting the target server

Comparative Analysis of Nikto and GVM Vulnerability Scanning Tools

Aspect	**Nikto**	**Greenbone Vulnerability Manager (GVM)**
Scope	Web server misconfigurations and file exposure	Full system: OS, network services, and CVE-based vulnerabilities
Tool Type	Lightweight, CLI-based	Enterprise-grade, GUI-based
Output Format	Plain text with flagged entries	Structured reports with severity graphs and CVSS scores
Exploit Mapping & CVE Support	No CVE mapping	Fully mapped to CVEs and CVSS scores
Risk Prioritization Method	Basic manual review	Detailed priority levels and

		likelihood-based classification
Reporting	Minimal CLI output	Rich visual dashboards, logs, and exportable reports

This table summarizes the methodological and functional differences between Nikto and GVM as used in the vulnerability assessment of the target system (IP: 10.6.6.13). While Nikto focuses on lightweight, surface-level web server scanning, GVM provides comprehensive system-wide coverage, CVE mapping, and advanced severity classification**.**

7. Conclusion

This report has presented a comprehensive evaluation of system vulnerabilities through two integrated tasks: the analysis of professional penetration testing reports (Task 1) and the practical application of automated vulnerability scanning tools (Task 2). The objective was to simulate a real-world penetration testing process while developing a clear understanding of industry-standard methodologies, risk classification, and remediation strategies.

In Task 1, three professional penetration testing reports from Cure53 were reviewed and compared. Each report focusing on CaseBox, Dovecot, and cURL offered distinct insights into vulnerability discovery, risk prioritization, and remediation clarity across different software architectures. This analytical exercise highlighted critical differences in testing methodologies (black-box vs. white-box), severity assessment models (impact-based vs. preventive design), and the maturity of codebases. Key takeaways included the significance of secure development practices, the effectiveness of defense-in-depth strategies, and the importance of clearly documented, actionable recommendations tailored to developers and system administrators.

Task 2 focused on the practical aspect of vulnerability identification using two scanning tools Nikto and Greenbone Vulnerability Manager (GVM) on a target web server (IP: 10.6.6.13). Nikto was effective in detecting application-layer misconfigurations such as outdated Apache versions, missing HTTP security headers, and exposed directories. GVM, in contrast, provided deeper host-level insights, identifying critical vulnerabilities stemming from unsupported operating systems, end-of-life scan engines, and unnecessary information disclosure via network protocols. Although hardware constraints prevented the GVM scan from completing fully, the results obtained were sufficient to perform a meaningful risk assessment and guide targeted remediation efforts.

Together, these tasks offered a well-rounded understanding of both the theoretical and practical dimensions of vulnerability assessment. The findings underscored the need for timely software updates and operating system migrations to reduce exposure to known CVEs, along with the implementation of security headers and web server hardening to minimize surface-level risks. They also emphasized the importance of conducting regular vulnerability scans using specialized tools to proactively manage emerging threats.

In conclusion, this assessment demonstrated the value of combining structured analysis with automated tools to simulate professional penetration testing workflows. By applying these integrated methods, cybersecurity practitioners can more effectively detect, prioritize, and address vulnerabilities thereby strengthening the overall resilience and trustworthiness of digital systems.

8. References

Cure53. (2014). *Security Assessment of CaseBox Web Application*. Retrieved from https://github.com/santosomar/public-pentesting-reports

Cure53. (2016). *Security Audit of Dovecot Mail Server – Version 2.2.26.0*. Retrieved from https://github.com/santosomar/public-pentesting-reports

Cure53. (2022). *Security Assessment of cURL and libcurl (Version 7.83.1)*. Retrieved from https://github.com/santosomar/public-pentesting-reports

Mozilla Developer Network. (n.d.). *Set-Cookie - HttpOnly*. MDN Web Docs. Retrieved June 21, 2025, from https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie

Mozilla Developer Network. (n.d.). *X-Content-Type-Options*. MDN Web Docs. Retrieved June 21, 2025, from https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Content-Type-Options

Greenbone Networks. (n.d.). *Greenbone Vulnerability Manager (GVM)*. Retrieved June 21, 2025, from https://www.greenbone.net/en/product