****

**""**

**Presented by**

Abdelrahman Albeshtawi:2100906014

Mahammad Alzoubi:2100906042

Ahmad Shwaiyat:2100906056

Yousef Hjooj:2100906076

Supervisor: Dr.

Submitted in Partial Fulfillment of the Requirements for Bachelor Degree in Cybersecurity

**Prince Al Hussein bin Abdullah Faculty of Information Technology**  
Al al-Bayt University

**Al-Mafraq- Jordan**

**DECLARATION**

As part of the requirements, for a Bachelor’s degree in Cybersecurity, we Ehab Abu Alia & Rashed Al Omari, state that the project titled “Active Directory Attack & Detect with Splunk” is our creation. We confirm that all data, sources, and information used in this project have been appropriately referenced and acknowledged.

Furthermore, we declare that this project has not been previously submitted for credit, towards another program or test at any institution. We understand that providing information or omitting facts may lead to measures, including the revocation of our degree. We take accountability for the content and accuracy of this assignment.

Signed: Mahammad Alzoubi, Ahmad Shwaiyat, Yousef Hjooj,abdalrahman Albeshtawi

Student No.: 2100906042, 2100906056,2100906076,2100906014.

**TABLE OF CONTENTS**

**CHAPTER.1 Introduction**

**1.1** Project Problem

**1.2** Project Goals

**1.3**What is an automated web vulnerability?

**1.4** Why are vulnerability scanners Important

**1.5** What the project covers?

**1.6** Beneficiaries

**1.7** SDLC Phases

**CHAPTER.2 Overview of Target vulnerabilities  
 2.1** Server-side Request Forgery (SSRF)  
 **2.2** Server-side Template Injection (SSTI)   
 **2.3** Path Traversal  
 **2.4** Cross-Site Scripting (XSS)

**CHAPTER.3 Building the Vulnerability Scanner**

**3.1** How the Scanner works?

**3.2** Choice of Programming and Libraries

**3.3** User Graphic interface (GUI)

**3.4** Comparisons with other vulnerability

**CHAPTER.4 Detection Methodologies**

**4.1** SSRF Detection Technique   
 **4.2** SSTI Detection Technique   
 **4.3** Path Traversal Detection Technique  
 **4.4** XSS Detection Technique   
  
**CHAPTER .5 Conclusion**

5.1 Summary of Achievements

5.2 Potential Enhancements

5.3 References

* 1. **Project Problem**

Modern web applications are increasingly targeted by cyberattacks due to vulnerabilities such as SQL injection, cross-site scripting (XSS), insecure APIs, and misconfigured servers. Manual vulnerability detection is time-consuming, error-prone, and requires specialized expertise. Many organizations, especially small-to-medium enterprises (SMEs), lack the resources to implement robust security practices, leaving their systems exposed to breaches. This project addresses the critical need for an automated, accessible, and efficient web vulnerability scanner to identify and mitigate risks proactively.

* 1. **Project Goals**

The main goals of this project are:

1. To develop an automated web vulnerability scanner that identifies and reports common security weaknesses.
2. To improve the security posture of web applications by enabling early detection and remediation of vulnerabilities.
3. To provide a user-friendly tool that can be used by developers, security teams, and organizations regardless of their technical expertise.

**1.3 What is an automated web vulnerability?**

An **automated web vulnerability scanner** is a software tool designed to systematically identify security weaknesses in web applications, APIs, and servers by combining predefined rules, machine learning (ML), and simulated attack patterns. It eliminates the need for manual penetration testing, enabling rapid, scalable, and repeatable security assessments.

**Key Vulnerabilities Detected**

These tools focus on critical flaws, including:

1. **Injection Attacks**:
   * **SQL Injection (SQLi)**: Exploits unsensitized input fields to execute malicious SQL queries.
   * **Command Injection**: Injects OS commands (e.g., ; rm -rf /) via vulnerable parameters.
2. **Broken Authentication**: Weak session management (e.g., predictable cookies) or brute-forceable login endpoints.
3. **Sensitive Data Exposure**: Unencrypted transmission of passwords, tokens, or PII.
4. **Cross-Site Scripting (XSS)**: Injects client-side scripts (e.g., <script>alert(document. Cookie)</script>) to hijack user sessions.
5. **Security Misconfigurations**: Default settings, open ports, or exposed debug interfaces.

**1.4 Why are vulnerability scanners Important**

1. Cost Efficiency: Reduce expenses associated with manual security audits.
2. Proactive Defense: Identify vulnerabilities before attackers exploit them.
3. Compliance: Meet regulatory standards (e.g., GDPR, PCI-DSS).
4. Reputation Protection: Prevent data breaches that damage organizational trust.
5. Continuous Monitoring: Enable real-time scanning in DevOps pipelines (shift-left security).

**1.5 What the project covers?**

1.5.1 Path Traversal

* Definition: Exploits improper input sanitization to access unauthorized files (e.g., /../../etc./passwd).
* Impact: Data theft, system compromise.
* Detection: Inject traversal sequences (e.g., ../, %2e%2e%2f) and analyze server responses for file disclosures.

1.5.2 Cross-Site Scripting (XSS)

* Definition: Injects malicious scripts into web pages (e.g., <script>alert(1)</script>).
* Types: Stored (persistent), Reflected (URL-based), DOM-based (client-side).
* Detection: Submit payloads and check for unencoded output in HTML/JS contexts.

1.5.3 Server-Side Request Forgery (SSRF)

* Definition: Forces a server to make unauthorized internal requests (e.g., to AWS metadata endpoints).
* Impact: Internal network reconnaissance, cloud credential theft.
* Detection: Send URLs with internal IPs (e.g., http://169.254.169.254) and monitor responses.

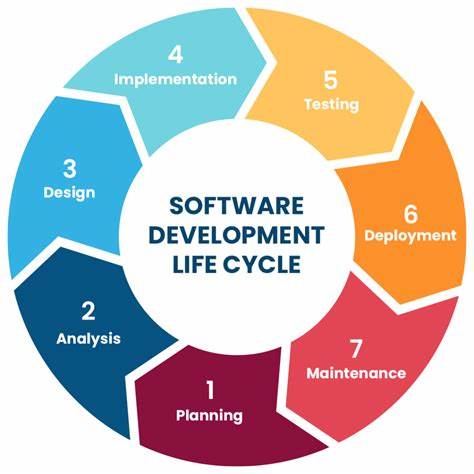
1.5.4 Server-Side Template Injection (SSTI)

* Definition: Injects malicious code into templating engines (e.g., Jinja2, Smarty).
* Impact: Remote code execution (RCE), data leaks.
* Detection: Test with template syntax (e.g., {{7\*7}} → 49 indicates vulnerability).

**1.6 Beneficiaries**

* Developers: Integrate security into CI/CD pipelines.
* Penetration Testers: Accelerate vulnerability discovery.
* Organizations: Reduce breach risks and audit costs.
* End Users: Safeguard personal data from exploits.

**1.7 SDLC Phases**

****

1. Planning
   * Define scope (path traversal, XSS, SSRF, SSTI).
   * Choose tools: Python, Requests, Beautiful Soup.
   * Set up test environments (Docker, Kali Linux).
2. Analysis
   * Study OWASP Top 10 patterns.
   * Map attack vectors (e.g., ../ for path traversal).
3. Design
   * Modularize components:
     + Scanner Engine (payload injection).
     + Reporting Module (PDF/HTML outputs).
4. Implementation
   * Code payload generators (e.g., SSTI {{7\*7}}).
   * Build response parsers (regex, DOM analysis).
5. Testing
   * Validate with OWASP Juice Shop.
   * Benchmark false positives/negatives.
6. Deployment
   * Package as CLI tool (Python Installer).
   * Publish on GitHub.
7. Maintenance
   * Update payload databases (CVE tracking).
   * Add new vulnerability checks (e.g., Log4j).

**Test**

**Test2**

**Test3**

**Hello World!**