A VULNERABILITY ASSESSMENT OF A SAMPLE WEB APPLICATION

REPORTED

BY

MWANAMISI KASSIM

**Objective:** The goal of this vulnerability assessment is to identify potential security weaknesses in the **VulnEWeb** web application using three widely-recognized security testing tools: **OWASP ZAP**, **Burp Suite**, and **Nmap**. These tools will help us conduct an ethical vulnerability scan, report vulnerabilities, and demonstrate how each tool can be used to assess the security posture of the target application.

### 1. ****Introduction to Tools****

* **OWASP ZAP (Zed Attack Proxy)**: An open-source web application security scanner designed to find security vulnerabilities automatically in web applications. It provides both automated and manual testing features.
* **Burp Suite**: A comprehensive suite for web application security testing, which provides powerful tools for scanning, intercepting, and analyzing web traffic to identify vulnerabilities.
* **Nmap**: A network scanning tool used for discovering hosts and services on a computer network, as well as performing security auditing by identifying open ports, services, and vulnerabilities.

### 2. ****Test Environment and Setup****

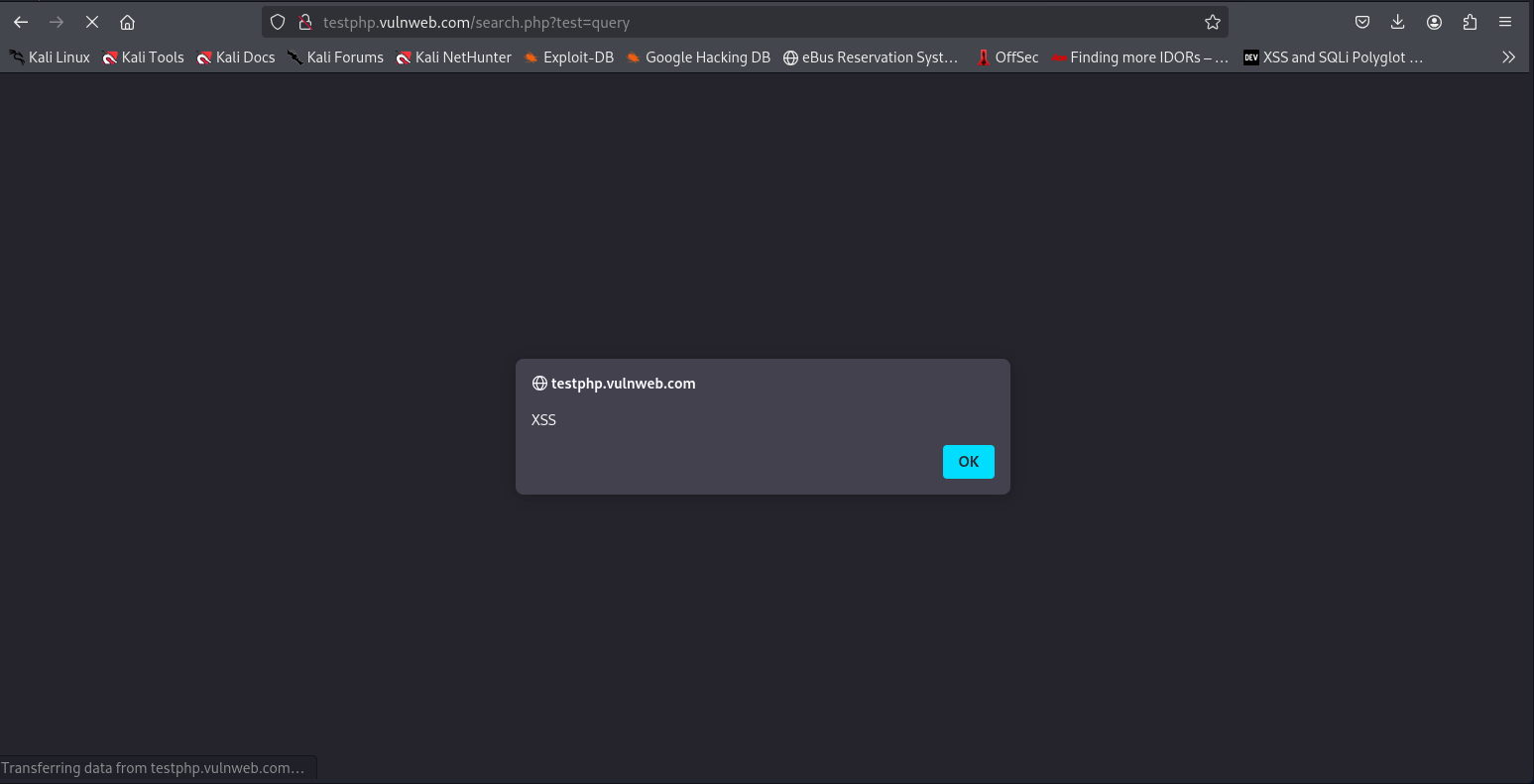
* **Target**: VulnEWeb (a deliberately vulnerable web application used for educational purposes).
* **Scope**: The vulnerability assessment was performed on the public-facing web application (vulnerable version), and all activities were carried out within the boundaries of ethical guidelines.
* **Methodology**: The assessment utilized a combination of automated and manual testing to uncover issues like SQL injection, Cross-Site Scripting (XSS), security misconfigurations, and others.

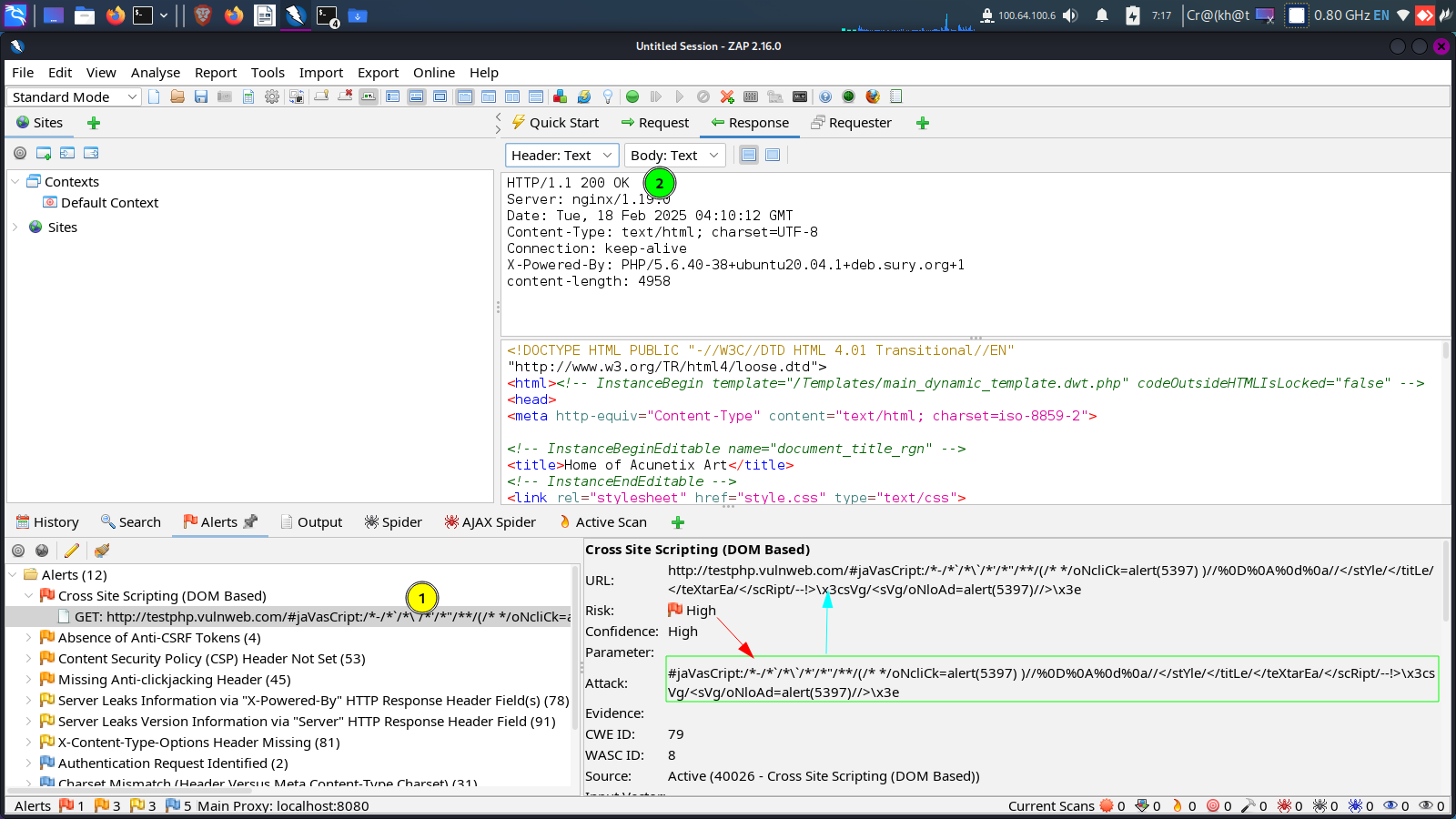
### 3. ****Vulnerability Assessment Using Tools****

#### ****Tool 1: OWASP ZAP (Zed Attack Proxy)****

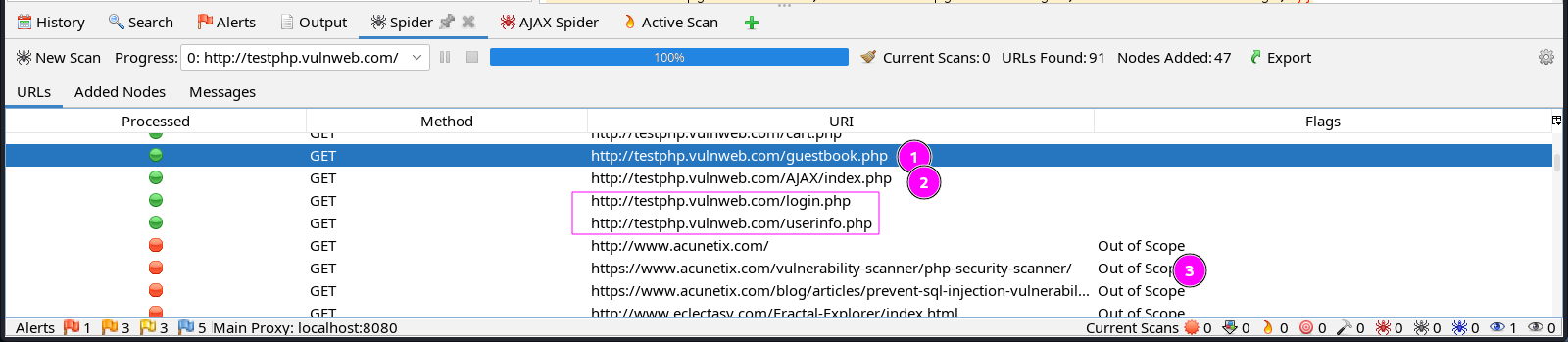
**Steps Taken**:

1. **Passive Scanning**: We began by using ZAP’s **passive scanning** feature to observe incoming and outgoing web traffic without actively probing the application. This is useful for identifying low-hanging fruits like information leaks, insecure HTTP methods, etc.
   * **POC (Proof of Concept)**: ZAP flagged sensitive data being sent in plain text (e.g., passwords) via HTTP requests. This was identified in the response headers and content, suggesting a need for HTTPS enforcement.
2. **Active Scanning**: We then moved to **active scanning**, where ZAP automatically tested for vulnerabilities such as SQL Injection, Cross-Site Scripting (XSS), and security misconfigurations.
   * **POC**: ZAP identified a **reflected DOM Based XSS** vulnerability in a user input field. A proof of concept payload (<script>alert('XSS')</script>) injected into the input field triggered an alert, indicating that the input wasn't properly sanitized.





1. **Spidering**: ZAP was used to **crawl** the VulnEWeb application to discover all the available endpoints and resources.
   * **POC**: The spider revealed several hidden endpoints, one of which was an **user login page** that lacked proper authentication mechanisms.

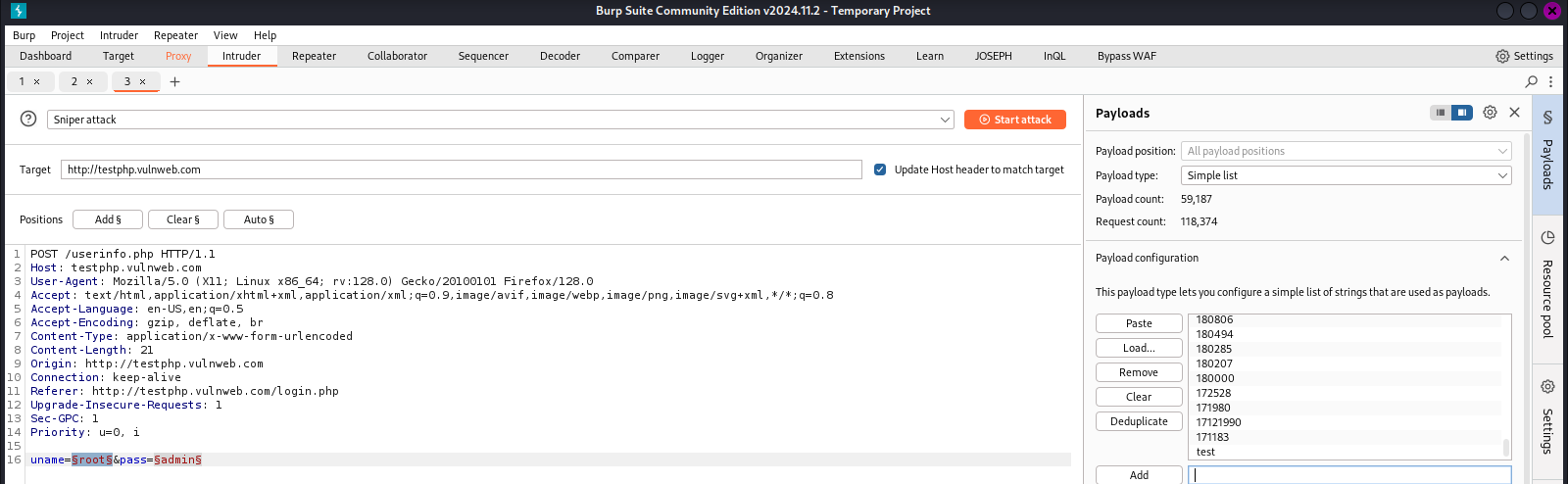


1. **Authentication Testing**: ZAP was configured to test for authentication weaknesses, identifying the login process's flaws, such as weak session management and insufficient password complexity policies.

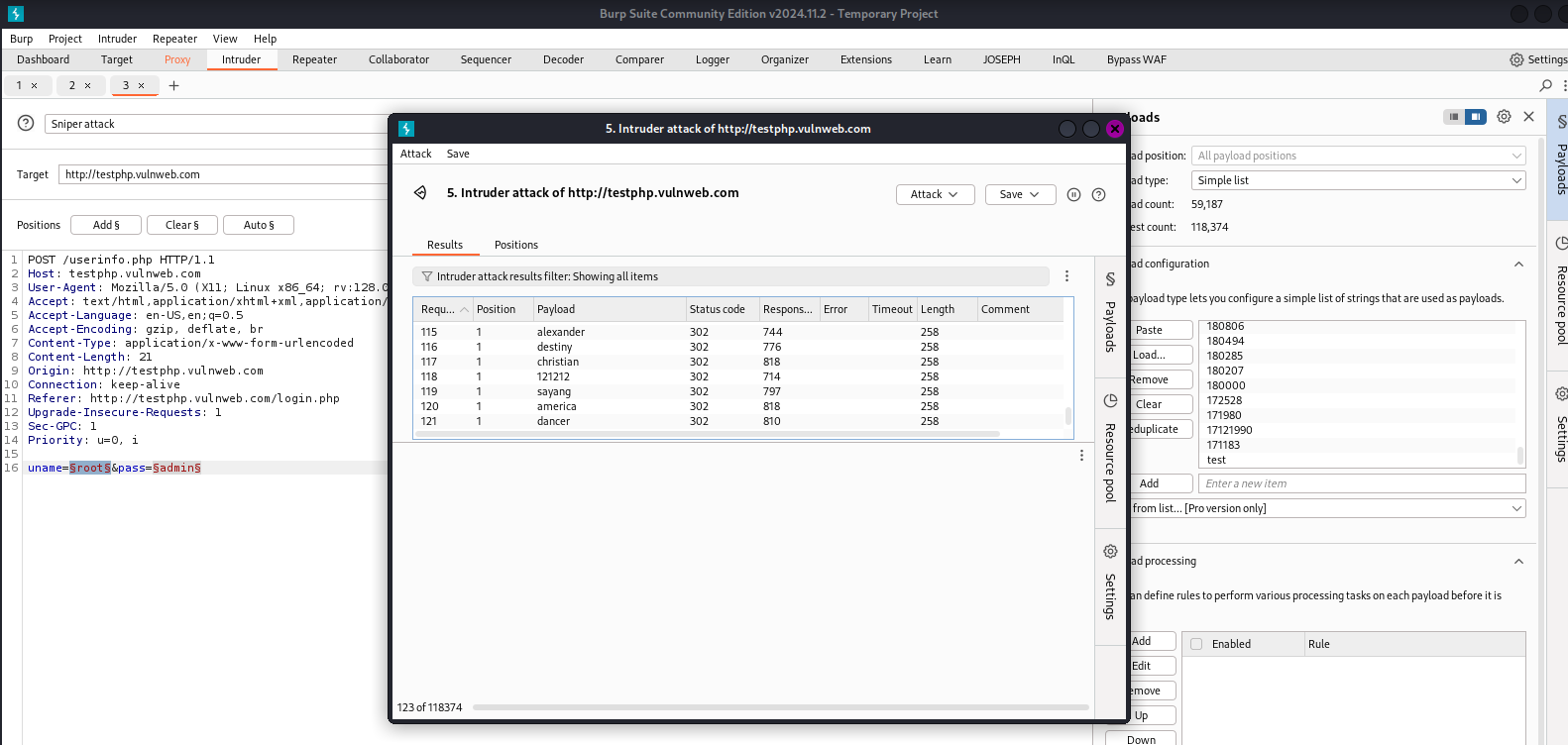
#### ****Tool 2: Burp Suite****

**Steps Taken**:

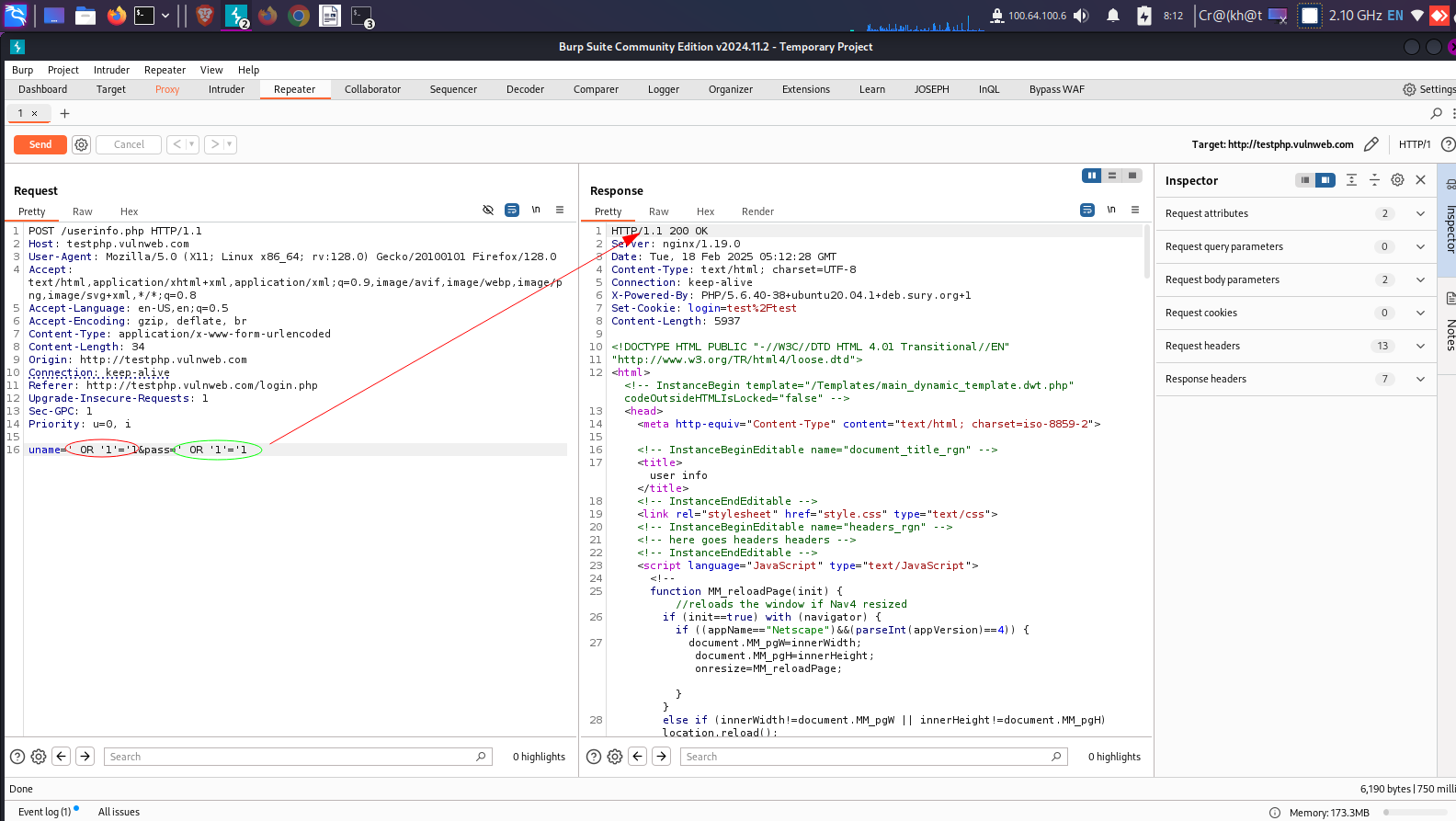
1. **Proxy Interception**: Burp Suite's **proxy** was set up to intercept all HTTP/HTTPS traffic between the browser and the VulnEWeb application. This allowed for manual inspection of all requests and responses.



1. **Intruder**: The **Intruder** tool was used to automate attacks such as brute-forcing login credentials or testing for SQL Injection via payload injection.
   * **POC**: The Intruder tool was used to test different variations of input for different password payloads to test brute-force attack , which confirmed that the application was vulnerable to brute-force attack, after successful login using **test** as a user name and **test** as a password.



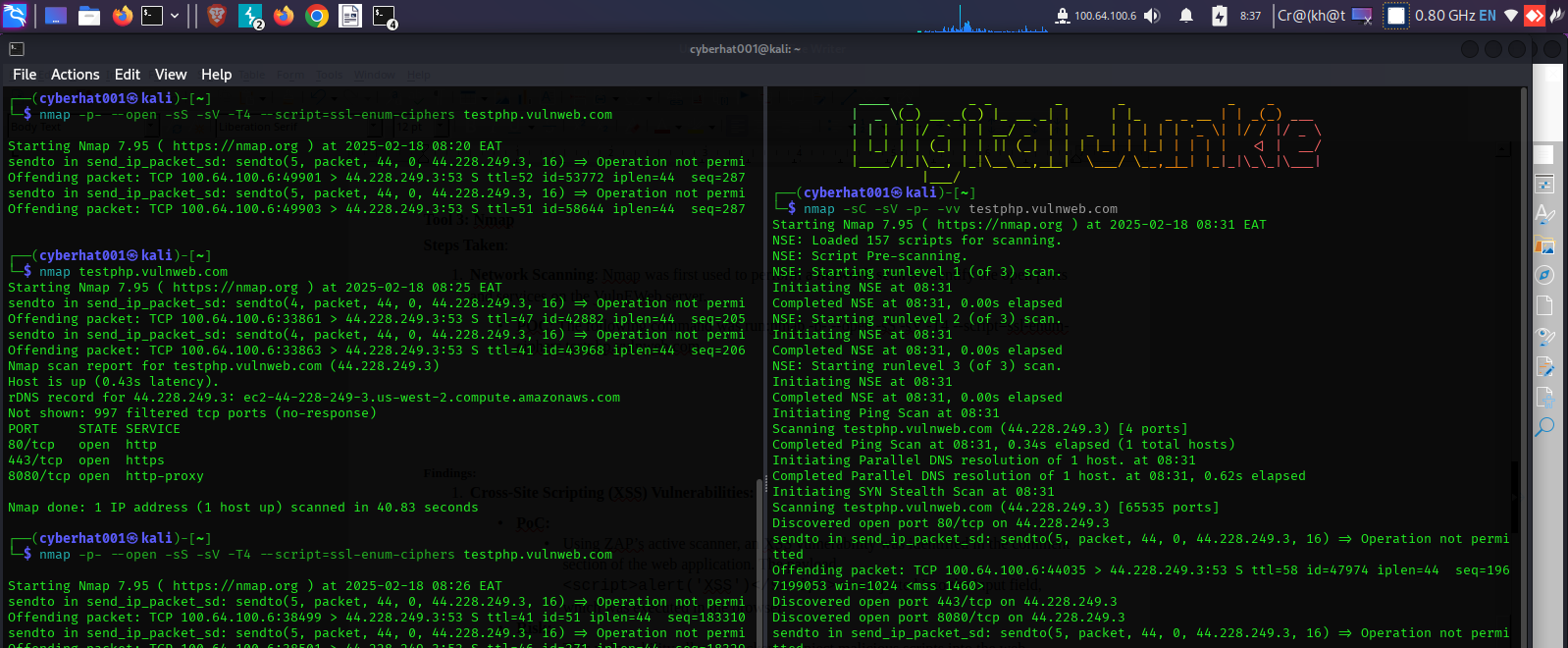
1. **Repeater**: The **Repeater** tool was employed to manually replay requests and modify them in real-time to analyze different responses or further exploit a vulnerability.
   * **POC**:
   * During a SQL injection test, the Intruder tool injected payloads such as ' OR '1'='1 into a vulnerable login field. The application returned a successful login even when the incorrect credentials were used, confirming the SQL injection vulnerability.



#### ****Tool 3: Nmap****

**Steps Taken**:

1. **Network Scanning**: Nmap was first used to perform a **network scan** to identify the open ports and services on the VulnEWeb server.
   * **POC**: The following command was run:nmap -p- --open -sS -sV -T4 --script=ssl-enum-ciphers testphp.vulnweb.com.



##### ****Findings:****

1. **Cross-Site Scripting (XSS) Vulnerabilities:**
   * **PoC:**
     + Using ZAP’s active scanner, an XSS vulnerability was identified in the comment section of the web application. The payload <script>alert('XSS')</script> was injected into the input field, which was executed in the browser.
   * **Risk:**
     + This vulnerability allows attackers to inject malicious scripts into the web application, potentially compromising user data and session information.
   * **Recommendation:**
     + Sanitize user inputs and escape special characters in the output. Utilize a web application firewall (WAF) for additional protection. Implement proper content security policies (CSP).
2. **SQL Injection:**
   * **PoC:**
     + The active scanner detected a potential SQL injection vulnerability in the login form. By submitting the payload ' OR '1'='1, the application returned a successful login without valid credentials, indicating that SQL queries were not properly parameterized.
   * **Risk:**
     + Attackers could manipulate SQL queries to access, modify, or delete data in the database.
   * **Recommendation:**
     + Use parameterized queries (prepared statements) or ORM frameworks to prevent direct manipulation of SQL queries.
3. **Sensitive Data Exposure (HTTP headers):**
   * **PoC:**
     + ZAP revealed that sensitive data, such as session cookies, was transmitted over HTTP rather than HTTPS, making it vulnerable to interception via man-in-the-middle attacks.
   * **Risk:**
     + Without encryption, session cookies and user credentials can be stolen by attackers.
   * **Recommendation:**
     + Implement HTTPS (SSL/TLS) across the entire web application and enforce HTTP Strict Transport Security (HSTS) headers.

#### ****Recommendations:****

1. **Input Validation and Sanitization:** Always validate user inputs and escape characters to prevent injection attacks like XSS and SQL Injection.
2. **Encryption:** Ensure all sensitive data is encrypted in transit using HTTPS, and always use strong SSL/TLS configurations.
3. **Authentication and Authorization:** Strengthen authentication mechanisms, implement proper session management, and enforce access controls.
4. **Security Patches:** Regularly update and patch all software components, services, and libraries.
5. **Network Security:** Close unnecessary ports and services to minimize the attack surface. Use firewalls and intrusion detection/prevention systems.

**Conclusion**

The security testing revealed critical vulnerabilities, including XSS, SQL Injection, brute-force attacks ,and sensitive data exposure to highlight a few, which could compromise the application’s integrity and user data. Network-level issues, such as open ports and insecure SSL/TLS configurations, were also identified, increasing the attack surface. By implementing recommended mitigations, such as input validation, secure authentication, and regular updates, the security of the application can be significantly improved.