Technical Report:

Our Team Members:

Ahmed Mohamed Fouad 2305071.

Ahmed Mohamed Alsafey 2305075.

Mohamed Atief Asmael 2305032.

Executive Summary:

Purpose of the test and brief overview of key findings:

The objective of this web penetration test was to evaluate the security stance of the target websites, APIs, and applications. The aim was to uncover vulnerabilities, analyze their potential impact, and offer practical recommendations to strengthen security. Additionally, we continuously enhance our cybersecurity skills through hands-on challenges, such as Capture the Flag (CTF) competitions.

key findings:

The Assessment Uncovered Several Critical Vulnerabilities That Could Potentially Compromise CIA Of The System. This Report include These vulnerabilities:

1. **Enumeration to Find Admin Path** (★★)

High-level impact:

Attackers can discover administrative entry points, increasing the likelihood of further attacks such as brute force, privilege escalation, or exploitation of unpatched vulnerabilities.

Summary of recommendations:

- Obfuscate or restrict access to sensitive directories such as admin paths.
- Use security measures like CAPTCHA and rate limiting to deter enumeration attempts.
- Implement HTTP response headers to suppress unnecessary information leakage.

2. **Brute Force on Admin Credentials** (★★★)

High-level impact:

Successful brute force attacks could allow unauthorized access to administrative accounts, leading to full control of the application or sensitive data compromise.

Summary of recommendations:

- Enforce strong password policies and multi-factor authentication (MFA).
- Apply account lockout mechanisms after several failed attempts.
- Monitor login attempts and alert for suspicious activity.

3. **Cross-Site Scripting (XSS) in Product Search** (★★★)

High-level impact:

Exploitation could allow attackers to execute malicious scripts in users' browsers, leading to session hijacking, phishing, or data theft.

Summary of recommendations:

- Validate and sanitize all user inputs both on the client and server side.
- Implement Content Security Policy (CSP) headers to restrict script execution.
- Use proper encoding for dynamic content to prevent injection attacks.

4. **SQL Injection in Login Page** (★★★★)

High-level impact:

An attacker could execute arbitrary SQL commands, compromising the database, extracting sensitive data, or altering application functionality.

Summary of recommendations:

- Use parameterized queries or prepared statements to prevent SQL injection.
- Perform input validation and reject suspicious input patterns.
- Limit database privileges for the application to minimize potential damage.

5. **Access Log (Sensitive Data Exposure)** (★★★★)

High-level impact:

Sensitive information such as user credentials, tokens, or internal IP addresses could be exposed, enabling attackers to exploit the application or infrastructure.

Summary of recommendations:

- Mask or exclude sensitive data from logs.
- Secure log files with proper access controls and encryption.
- Regularly review and purge logs to maintain minimal retention of sensitive information.

Vulnerability Findings :

1. **SQL Injection in Login Page** (Critical)

i. Description, risk, and potential impact:

SQL Injection allows attackers to manipulate backend SQL queries by injecting malicious inputs into user fields. This can lead to unauthorized access, data exfiltration, or even full database compromise.

Potential impact:

- Unauthorized access to sensitive user data such as credentials, PII, or financial records.
- Potential for privilege escalation and control over the database or application.

ii. Evidence:

- **Proof of Concept:**
  Injected payload: admin' OR '1'='1 into the login form, bypassing authentication.
- Screenshot: Displays successful login to an admin account without valid credentials.
- Log snippet: Shows SQL query manipulated to bypass checks (SELECT * FROM users WHERE username='admin' OR '1'='1' AND password='...').

iii. Remediation steps:

- Use parameterized queries or prepared statements for database interactions.
- Validate and sanitize all user inputs to reject malicious patterns.
- Implement a web application firewall (WAF) to detect and block injection attempts.
- Regularly conduct code reviews and use static analysis tools to identify SQLi risks.

2. **Access Log (Sensitive Data Exposure)** (Critical)

i. Description, risk, and potential impact:

Access logs were found to contain sensitive information, such as API keys, session tokens, or internal IP addresses. Exposure of this data can enable attackers to perform lateral movement, session hijacking, or unauthorized API usage.

Potential impact:

- Compromise of API keys could lead to misuse of backend services.
- Attackers may gain insight into application infrastructure and operational behavior.

ii. Evidence:

- Screenshot:
- Log snippet: GET /ftb.

iii. Remediation steps:

- Remove or exclude sensitive information from log entries.
- Secure log files with strict access controls and encryption.
- Review logging configurations to ensure adherence to privacy and security guidelines.
- Implement automated log monitoring tools to detect and alert on anomalous activities.

3. **Cross-Site Scripting (XSS) in Product Search** (Critical)

i. Description, risk, and potential impact:

XSS vulnerabilities allow attackers to inject malicious scripts into web pages. These scripts execute in the context of other users, leading to session hijacking, phishing, or data theft.

Potential impact:

- Theft of session cookies could result in unauthorized access to user accounts.
- Malicious scripts may redirect users to phishing sites or inject trojans.

ii. Evidence:

- **Proof of Concept:**
  Payload: <iframe src="javascript:alert(xss)"> injected into the product search bar, causing an alert box to appear for other users viewing the page.
- Screenshot: Alert box displaying on a victim’s browser.
- Network trace: Confirms execution of injected malicious payload.

iii. Remediation steps:

- Sanitize and validate all user inputs on both client and server sides.
- Use output encoding techniques to render input as plain text instead of executable code.
- Deploy a Content Security Policy (CSP) to restrict script execution.
- Conduct regular security testing to identify and patch XSS vulnerabilities.

4. Broken Access Control (Critical)

i. Description:

Broken access control occurs when restrictions on what authenticated users can access are improperly enforced. This can allow unauthorized users or attackers to gain access to sensitive data or restricted functionalities.

Potential impact:

- Unauthorized access to sensitive information or administrative functions.
- Potential for privilege escalation, allowing attackers to perform actions beyond their intended permissions.

ii. Evidence:

- Proof of Concept:
- Direct access to admin-only pages via URL manipulation without proper authentication.
- Bypassed restrictions using forged requests or tokens.
- Screenshot: Displays unauthorized access to an admin page.
- Log entry: Shows access granted to a non-privileged user with manipulated HTTP requests.

iii. Remediation steps:

- Implement strict role-based access control (RBAC) to ensure users only access permitted resources.
- Validate user permissions on the server side for every request, regardless of client-side checks.
- Use secure session management techniques to prevent token manipulation or reuse.
- Regularly test for access control flaws using automated tools and manual penetration testing.

Conclusion:

Summary of security posture:

The assessment revealed critical vulnerabilities, including SQL injection, XSS, and sensitive data exposure, indicating significant security gaps in input validation, authentication, and logging practices. These issues put sensitive data and system integrity at risk.

Overall risk level and next steps:

The overall risk level is **high**, requiring immediate remediation of critical issues, such as fixing the SQL injection and XSS vulnerabilities, securing logs, and enforcing stronger authentication measures. Long-term steps include regular security testing, developer training, and integrating security tools into the development process.

By addressing these gaps, the organization can strengthen its defenses and reduce exposure to potential threats.

GitHub Repository: GitHub - a7medxd123/Cybersecproject