Penetration Testing Report

Executive Summary:

The penetration testing was conducted on the web application hosted on HackThisSite, focusing on the Basic 1-11 challenges. These challenges encompassed various aspects of web security, including HTML source code analysis, URL manipulation, encryption, steganography, SQL injection, cross-site scripting (XSS), cookie manipulation, and HTTP header manipulation. The testing revealed several vulnerabilities that could potentially compromise the confidentiality, integrity, and availability of the application. Recommendations have been provided to enhance the security posture of the web application.

Scope of Web Application Tested:

The scope of the penetration testing included the Basic 1-11 challenges hosted on the HackThisSite platform. The testing primarily targeted the functionality and security of the web application, focusing on identifying vulnerabilities and providing recommendations for remediation.

Vulnerability Description and Key Findings:

Basic 1:

- Vulnerability: Lack of input validation and insufficient access controls.

- Description: The hidden message within the HTML source code could be easily accessed, potentially exposing sensitive information.

- Recommendation: Implement proper access controls and input validation mechanisms to prevent unauthorized access to sensitive information.

Basic 2:

- Vulnerability: Insecure direct object references.

- Description: Manipulating URL parameters allowed access to restricted pages.

- Recommendation: Implement proper authorization mechanisms and validate user inputs to prevent unauthorized access to sensitive resources.

Basic 3:

- Vulnerability: Weak encryption.

- Description: The encryption method used was susceptible to brute-force attacks, compromising the confidentiality of the message.

- Recommendation: Implement stronger encryption algorithms and key management practices to protect sensitive data.

Basic 4:

- Vulnerability: Lack of steganography protection.

- Description: The hidden message within the image was easily discoverable, potentially leading to unauthorized access.

- Recommendation: Enhance steganography techniques and consider additional layers of security for hiding sensitive information.

Basic 5:

- Vulnerability: Insecure authentication mechanism.

- Description: The JavaScript code contained the password for authentication, making it susceptible to unauthorized access.

- Recommendation: Implement secure authentication methods such as hashing and salting passwords to mitigate the risk of unauthorized access.

Basic 6:

- Vulnerability: SQL injection.

- Description: The application was vulnerable to SQL injection attacks, allowing attackers to bypass authentication.

- Recommendation: Implement parameterized queries and input validation to prevent SQL injection attacks.

Basic 7:

- Vulnerability: Insecure cookie handling.

- Description: Cookies were manipulated to gain unauthorized access to restricted areas.

- Recommendation: Use secure and HTTPOnly flags for cookies, implement proper session management, and validate cookie data to prevent unauthorized access.

Basic 8:

- Vulnerability: Insecure password storage.

- Description: The password was stored in plaintext within the JavaScript code, posing a security risk.

- Recommendation: Hash and salt passwords before storage to prevent exposure in case of a breach.

Basic 9:

- Vulnerability: Cross-site scripting (XSS).

- Description: The application was vulnerable to XSS attacks, allowing attackers to execute arbitrary JavaScript code.

- Recommendation: Implement input sanitization and output encoding to mitigate XSS vulnerabilities.

Basic 10:

- Vulnerability: Insecure form handling.

- Description: Hidden form fields were manipulated to bypass authentication.

- Recommendation: Validate form submissions on the server-side and implement proper access controls to prevent unauthorized access.

Basic 11:

- Vulnerability: Lack of HTTP header security.

- Description: HTTP headers were manipulated to bypass authentication and gain unauthorized access.

- Recommendation: Implement secure HTTP headers and enforce strict security policies to prevent header manipulation attacks.

Recommendations:

- Implement a robust web application firewall (WAF) to monitor and filter incoming traffic for malicious activities.

- Regularly update and patch the web application and its underlying components to address known vulnerabilities.

- Conduct regular security assessments, including penetration testing and code reviews, to identify and remediate security flaws.

- Provide security awareness training to developers and users to educate them about common security risks and best practices.

- Implement a bug bounty program to incentivize security researchers to report vulnerabilities responsibly.

- Regularly conduct comprehensive security assessments, including penetration testing, to identify and mitigate vulnerabilities.

- Implement secure coding practices and perform code reviews to prevent common security issues such as injection attacks and information disclosure.

- Monitor web application logs and network traffic for signs of suspicious activity and potential security breaches.

By addressing these recommendations and implementing proper security measures, the web application can enhance its resilience against potential cyber threats and safeguard the confidentiality, integrity, and availability of its data and services.