Vulnerability Assessment of Web Applications and Recommendations for Actions: defenselessV1 Penetration Testing Report

Oudjani Seyyid Taqy eddine (2024)

Keywords: web application, vulnerability assessment, penetration testing, information security, cyber-attacks, SQL injection, XSS, RFI, SAST&DAST.

Abstract

The objective of this report is to find web application vulnerabilities of a vulnerable application This report encapsulates the comprehensive vulnerability assessment conducted for the web application named Defenseless. The assessment encompassed the utilization of Static ApplicationSecurity Testing (SAST) and Dynamic Application Security Testing (DAST) tools, alongside penetration testing techniques and manual code review. By integrating these methodologies, a thorough examination of Defenseless was achieved to identify and mitigate potential vulnerabilities.The amalgamation of SAST, DAST, penetration testing, and manual code review facilitated acomprehensive understanding of Defenseless's security posture. The findings of this assessment will serve as a foundation for implementing robust security measures and fortifying the resilience of Defenseless against potential cyber threats.

Introduction

Currently, there is no single system or application that is not vulnerable to attack. In the case of this study, the vulnerable web application is designed with a weak security policies that made it susceptible to most vulnerabilities for penetration testing. This report will cover vulnerable web application penetration test and report. The test report will include four vulnerabilities found, each with an explanation, impact score, the probability of exploitation, and security mechanism to protect the system against the attack. The investigation will cover the working mechanism of SQL injection and a novel way to protect against the attack.

Section 1: Web Application Test Report

Executive Summary

The vulnerability assessment of the web application, named Defenseless, revealed critical security shortcomings stemming from its development without adherence to fundamental security measures. Defenseless primarily facilitates Create, Read, Update, and Delete (CRUD) operations, empowering clients to register, modify their information, manage their accounts, and add clients to their functionality suite. However, the application exhibits numerous vulnerable input points and fields, including forms for GET and POST requests, name fields, passwords,emails, phone numbers, and more.

The application's logic follows a standard user flow: users create an account, log in to the main space, receive a welcome message and interfaces, manipulate their information, add clients as needed, and log out. Additionally, it offers functionality to request a new password, which necessitates sending a valid email, an aspect that was found to be vulnerable during the assessment.

Two of the most perilous threats discovered, which posed the risk of system disclosure, were SQL injection and Remote File Inclusion(RFI). These vulnerabilities could potentially lead to severe data breaches, unauthorized access, and complete system compromise if exploited by malicious actors.

The report underscores the critical importance of integrating robust security measures during the development phase of web applications. The identified vulnerabilities pose significant risks, including data breaches and potential exploitation by malicious actors. Addressing these vulnerabilities is imperative to safeguard sensitive user data, uphold user trust, and maintain the integrity of the Defenseless application. Implementing comprehensive security protocols and adopting a proactive approach to security will be essential in fortifying Defenseless against potential cyber threats and ensuring a secure user experience.

Definitions of Impact and probability of exploitation

Impact of exploitation

Consequences of exploitation	Explanation
Low	Even if the attack is successful, the attacker would not have control over what's being exploited, modification of the contents is not also possible. Hence, the attack impact scope may be limited to only allowing an attacker to access a specific level of only viewing the system content.
Medium	The attack needs some pre-conditions for the attacker to exploit the whole system because the system is configured with some security measures. Therefore, a successful attack may allow little modifications on the specific system contents which may not affect the entire system.
High	There is a total compromise of the whole system, and therefore, loss in system protections, which leads to full system files and configuration disclosure. Moreover, an attacker can modify or control the entire system.

The Probability of Exploitation

Chances of Exploitation	Explanation
Low	There is a need for sophisticated tools to bypass systems restrictions to expose the vulnerability before starting to exploit the system. So, this vulnerability is very unlikely to occur.
Medium	The chance for this kind of vulnerability to occur requires some preconditions. So, this vulnerability occurs moderately.
High	For this vulnerability to be exposed, require only some open available tools with some little knowledge. So, this weakness is very common to occur.

Vulnerability 1

File Inclusion: Remote File Inclusion	
Impact	Probability
Medium	High

Finding

http://127.0.0.1/defenselessV1/PHPMailerAutoload.php?file=%22../../../../../etc/passwd%22 vulnerable to file inclusion attacks. It allows an attacker to execute filesystem commands remotely from the vulnerable web application's URL. This vulnerability was exploited using Linux terminal commands and the URL of the web application; the 'password' and 'group' files of the 'etc' directory of the hosting server were exposed without authorization.

Remote File Inclusion." It is a type of vulnerability commonly found in web applications that allow an attacker to include a remote file (usually containing malicious code) on a server. RFI occurs when a web application dynamically includes external files or scripts, such as PHP includes, without proper validation.

Here's how RFI works:

Insecure File Inclusion: Web applications often include files dynamically to reuse code or fetch data from external sources. If the application doesn't properly validate or sanitize user input used to specify the file to include, it can lead to an RFI vulnerability.

Exploiting RFI: An attacker can exploit RFI by manipulating the input parameters that specif y the file to include. By providing a URL pointing to a malicious file hosted on a remote server, the attacker can trick the application into including and executing the malicious code.

Consequences: The consequences of a successful RFI attack can vary depending on the nature of the included file and the permissions of the web server process. Attackers can exploit RFI to execute arbitrary code, steal sensitive data, gain unauthorized access to the server, or launch further attacks against other systems.

To prevent RFI vulnerabilities, it's essential to follow secure coding practices such as:

Avoid including files based on user-controlled input.

Use whitelisting to specify allowed files and directories for inclusion.

Sanitize and validate user input to ensure it contains only expected values.

Implement proper access controls to restrict file inclusion to trusted sources.

Keep software and libraries up to date to mitigate known vulnerabilities.

Recommendation

The disabling of the '*allow_url_fopen'* function of PHP allows a filesystem of the remote server to be accessed from the URL. A below PHP code can be used to fix and validate the inputs from the system before execution. Lastly, the use of IPS, IDS, and Web applications firewall to blacklist execution of unexpected malicious commands from the URL could help to secure the system from this attack.

```
<?PHP
 if (is_readable($filename)) {
require $filename;    }
?>
```

Vulnerability 2

SQL Injection : SQL Injection Attack	
Impact	Probability
High	High

Finding

Our comprehensive assessment revealed that all input points across various sections of the site are susceptible to SQL injection attacks due to a fundamental lack of senitization and validation protocols.

This alarming discovery encompasses critical data entry points such as usernames, email addresses, passwords, phone numbers, and locations. The absence of robust sanitization and validation mechanisms across the platform poses a severe risk to the confidentiality, integrity, and availability of sensitive information.

Specifically, vulnerabilitieswere identified in the following URLs:

http://127.0.0.1/defenselessV1/Signup.php,
http://127.0.0.1/defenselessV1/profile-edit.php,
http://127.0.0.1/defenselessV1/clients.php,
http://127.0.0.1/defenselessV1/login.php,
http://127.0.0.1/defenselessV1/passwd-chnge.php.

These URLs represent critical entry points into the system, thereby amplifying the urgency for immediate remediation efforts.
Without prompt action to implement robust input validation and sanitization measures, the system remains highly susceptible to exploitation by malicious actorsseeking unauthorized access to sensitive data. It is imperative that remediation efforts prioritize the implementation of industry-standard security controls to mitigate the risk posed by SQL injection vulnerabilities across the web application. Failure to address these vulnerabilities promptly may result in severe consequences, including data breaches, financial losses, and damage to the organization's reputation. As such, I recommend initiating an immediate action plan to remediate these vulnerabilities and fortify the application's defenses against SQL injection attacks.

SQL injection is a type of security vulnerability that occurs when an attacker is able to manipulate SQL queries in a web application's input fields. This vulnerability arises when user input is not properly sanitized or validated before being passed as part of an SQL query to the database.
Here's how SQL injection works and its consequences:
Injection Points: SQL injection attacks typically target input fields such as login forms, search boxes, and other user-controlled parameters that interact with a database.
Malicious Input: An attacker submits specially crafted input, such as SQL statements or fragments, to the vulnerable input fields of the web application.
Unsanitized Input: If the web application does not properly validate or sanitize user input, the malicious SQL code provided by the attacker can be concatenated with the SQL query executed by the application.
SQL Query Manipulation: As a result of successful injection, the attacker can manipulate the original SQL query in various ways, including:
Retrieving sensitive data from the database.
Modifying or deleting existing data in the database.
Performing unauthorized actions, such as bypassing authentication mechanisms.
Consequences:
The consequences of SQL injection attacks can be severe, ranging from data breaches and leakage of sensitive information to complete compromise of the application and underlying database. In some cases, SQL injection can lead to full control of the web server or even the entire infrastructure.

Recommendation

Prepared Statements and Parameterized Queries: Use parameterized queries and prepared statements provided by database APIs to separate SQL code from user input, preventing injection attacks.

Input Validation and Sanitization: Validate and sanitize all user input before using it in SQL queries. This includes input from forms, URL parameters, and cookies.

Least Privilege Principle: Restrict database user privileges to the minimum required for the application to function properly. Avoid using database accounts with excessive permissions.

Web Application Firewall (WAF): Implement a WAF to monitor and filter web traffic.

Vulnerability 3

Cross-Site Scripting: Stored XSS attack	
Impact	Probability
Medium	High

Finding

The vulnerable page identified as susceptible to Stored Cross-Site Scripting (XSS) poses a severe risk to the integrity and security of the web application. Specifically, the input field labeled "name" within the page d *(http://127.0.0.1/defenselessV1/Signup.php)* has been identified as vulnerable to Stored XSS attacks. This vulnerability allows malicious actors to inject arbitrary scripts into the application, which are then saved in the database and subsequently reflected in the main page, thereby compromising the integrity of the site's content. The potential impact of this vulnerability extends beyond mere data manipulation to encompass a wide range of malicious activities, including session hijacking, phishing attacks, and the unauthorized extraction of sensitive information from unsuspecting users.

Given the critical nature of this security flaw, immediate action is warranted to remediate the vulnerability and mitigate the risk of exploitation. Failure to address this issue promptly may result in severe consequences, including reputational damage, loss of user trust, and legal liabilities. I recommend implementing stringent input validation and output encoding mechanisms to sanitize user inputs effectively and prevent the execution of malicious scripts within the application. Additionally, conducting thorough security testing and code reviews on a regular basis can help identify and mitigate similar vulnerabilities in the future, ensuring the continued resilience and security of the web application against evolving threats.

Here's how XSS works and its consequences:

-Injection Points: XSS attacks typically target input fields, such as search boxes, comment sections, or form fields , where user-controlled data is displayed without proper validation or sanitization.
-Malicious Script Injection: An attacker injects malicious scripts, usually written in JavaScript,into the vulnerable input fields of the web application.
-Execution in the Victim's Browser: When other users visit the affected web page, their browsers unknowingly execute the injected malicious scripts alongside legitimate content.

Consequences: XSS attacks can have various consequences, including:
Cookie Theft: Attackers can steal users' session cookies, allowing them to impersonate the victims and perform unauthorized actions.
Data Theft: Attackers can extract sensitive information from the web page, such as login credentials, personal data, or payment information.
Session Hijacking: Attackers can hijack users' active sessions, allowing them to perform actions on behalf of the victims.
Defacement: Attackers can deface the web page by modifying its content or layout, causing reputational damage to the organization.

Recommendation

Input Validation and Sanitization: Validate and sanitize all user input before displaying it on web pages. This includes input from forms, URL parameters, cookies, and HTTP headers.

Output Encoding: Encode user input and dynamic content using appropriate encoding techniques (e.g., HTML entities encoding, JavaScript escaping) to prevent browsers from interpreting them as executable scripts.

Content Security Policy (CSP): Implement CSP headers to restrict the sources from which browsers can load content, helping to mitigate XSS attacks by blocking inline scripts and unauthorized external scripts.

Vulnerability 4

Cross-Site Scripting : Reflected XSS attack	
Impact	Probability
High	Medium

Finding

Our thorough assessment has revealed a significant vulnerability exposing the application to Reflected Cross-Site Scripting (XSS) attacks, thereby jeopardizing the integrity and security of the system. Specifically, the page located at *http://127.0.0.1/defenselessV1/clients.php* has been identified as vulnerable to Reflected XSS. This vulnerability allows malicious actors to inject and execute arbitrary scripts within the application, posing a serious threat to the confidentiality and integrity of user data.

Upon submission, all input fields within the page are susceptible to exploitation, as the injected script is reflected back to the user in the same page, thereby facilitating the execution of malicious code. The potential consequences of this vulnerability are grave and include unauthorized data disclosure, session hijacking, and the compromise of user accounts. Immediate action is warranted to remediate this vulnerability and safeguard the application against exploitation. I recommend implementing robust input validation and output encoding mechanisms to sanitize user inputs effectively and prevent the execution of malicious scripts within the application. Additionally, conducting thorough security audits and penetration testing on a regular basis can help identify and mitigate similar vulnerabilities in the future, ensuring the continued resilience and security of the web application against emerging threats. Failure to address this vulnerability promptly may result in severe repercussions, including reputational damage, financial losses, and legal liabilities. It is imperative that remediation efforts prioritize the implementation of industry-standard security controls to mitigate the risk posed by Reflected XSS attacks across the web application.

Recommendation

Input Validation and Sanitization: Validate and sanitize all user input before displaying it on web pages. This includes input from forms, URL parameters, cookies, and HTTP headers.

Output Encoding: Encode user input and dynamic content using appropriate encoding techniques (e.g., HTML entities encoding, JavaScript escaping) to prevent browsers from interpreting them as executable scripts.

Content Security Policy (CSP): Implement CSP headers to restrict the sources from which browsers can load content, helping to mitigate XSS attacks by blocking inline scripts and unauthorized external scripts.

Secure Coding Practices : Follow secure coding practices, such as using framework-specific security features, escaping user input in templates, and avoiding the use of eval() and innerHTML functions.

Regular Security Audits: Conduct regular security audits and vulnerability assessments to identify and remediate XSS vulnerabilities in web applications.

Conclusion :

Among the most critical vulnerabilities identified are Remote File Inclusion (RFI), SQL injection, Reflected Cross-Site Scripting (XSS), and Stored Cross-Site Scripting (XSS). These vulnerabilities expose the application to a range of malicious activities, including data theft , unauthorized access, and the execution of arbitrary code.

The vulnerable URLs where these vulnerabilities were identified include:

http://127.0.0.1/defenselessV1/Signup.php (SQL, XSS Stored)

http://127.0.0.1/defenselessV1/profile-edit.php (SQL)

http://127.0.0.1/defenselessV1/clients.php (SQL, XSS Reflected)

http://127.0.0.1/defenselessV1/login.php (SQL)

http://127.0.0.1/defenselessV1/passwd-chnge.php (SQL)

http://127.0.0.1/defenselessV1/PHPMailerAutoload.php (RFI)