|  |  |
| --- | --- |
|  |  |
| Application Attacks |  |
|  |  |
|  | CODE REVISION: 1.0.0.0DATE: 8/26/2024REPORT: AAVR2024 |
|  | AUTHOR: SEIF ELDIN MOHAMED GABER |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | Application Attacks Vulnerability ReportAAVR1.0.0 -- SQL Injection in URL -- High****Vulnerability Exploited:**** SQL Injection in URL****Severity:**** High****System:**** AAVR (Application Attacks) **Vulnerability Explanation:**  A SQL Injection vulnerability was identified in the product category filtering functionality by manipulating the category parameter in the URL, an attacker can inject SQL code to bypass application filters and retrieve hidden or unauthorized data from the database. This type of vulnerability allows attackers to access sensitive information, potentially leading to data breaches, unauthorized access  Vulnerability Walk-thru:  . Click on any product category, such as "Gifts," to filter products by that category.  . The URL will change to something like:  https://your-lab-url.com/product?category=Gifts  . Modify the category parameter value by adding a single quote (') to test if the application is vulnerable to SQL injection:  https://your-lab-url.com/product?category='  . Use SQL injection to exploit the vulnerability and retrieve hidden data. Modify the URL to:  <https://your-lab-url.com/product?category=>' or 1=1—      . As you see the application displays all products from the database, not just those from the "Gifts" category. This demonstrates that the filter was bypassed using SQL injection.  Recommendation:  . Always use prepared statements and parameterized queries to ensure that user inputs are treated as data, not executable code. This method effectively prevents SQL injection attacks.  . Implement a WAF to detect and block SQL injection attempts based on known attack patterns. This adds an additional layer of security to the application.  . Validate and sanitize all user inputs on both client and server sides. Implement strict input validation rules and remove or escape special characters that could be used for SQL injection   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  | |  | |  | Application Attacks Vulnerability ReportAAVR2.0.0 -- SQL Injection Login bypass -- Critical****Vulnerability Exploited:**** SQL Injection Login bypass****Severity:**** Critical****System:**** AAVR (Application Attacks) **Vulnerability Explanation:**  The login form of the web application is vulnerable to SQL injection. By injecting SQL commands into the input fields, an attacker can manipulate the SQL query executed by the application. This vulnerability allows unauthorized users to bypass authentication and access restricted areas of the web application  Vulnerability Walk-thru:  . Access the login page of the web application  . In the **username** field, enter the following payload  administrator’--  . password will be ignored due to the injection so you can fill it with any text or leave it empty  . the injection is successful, you will gain access to the restricted areas of the application, effectively bypassing authentication      . As you see the injection and it log me in successfully    Recommendation:  . **Use Prepared Statements and Parameterized Queries:** These prevent the application from treating user input as executable code, effectively neutralizing SQL injection attacks.  . **Input Validation and Sanitization:** Implement strict input validation to ensure only allowed characters and inputs are processed by the application  . **Update and Patch Software:** Keep all software components up to date with security patches | | |  | |  |  |  | | |  |
|  | Application Attacks Vulnerability ReportAAVR3.0.0 -- Stored Cross-Site Scripting (XSS) -- High****Vulnerability Exploited:**** stored Cross-Site Scripting (XSS)****Severity:**** High****System:**** AAVR (Application Attacks) **Vulnerability Explanation:**  Stored Cross-Site Scripting (XSS) vulnerability occurs when malicious scripts are injected into a trusted website. In this case, user inputs (comments) are stored and then reflected back on the web page without proper sanitization or encoding. As a result, an attacker can execute arbitrary JavaScript code in the victim's browser when they view the affected page, leading to potential theft of cookies, session tokens, or other sensitive data  Vulnerability Walk-thru:  . Access the page containing the comment input field (typically found on a blog post, feedback form, or a dedicated comment section).  . Locate the comment input field where users can submit their comments  . Enter the script in the comment field:  <script>alert(123 )</script> . After submitting the comment, navigate back to the page where comments are displayed.. You should see a pop-up alert saying "123", indicating that the script has been executed in the browser. | |  |
|  |  | |  |
|  |  | |  |
|  | Recommendation:  . Implement strict input validation on all user-supplied data. Only allow input that strictly matches expected patterns (e.g., alphanumeric characters only).  . Utilize libraries or frameworks that automatically handle output encoding and provide built-in protection against XSS attacks.  . Implement a Content Security Policy to limit the execution of scripts to trusted sources  . Regularly test the application for vulnerabilities using automated tools and manual penetration testing to identify and remediate XSS and other vulnerabilities | |  |
|  |  | |  |
|  | Application Attacks Vulnerability ReportAAVR4.0.0 -- Multi-Factor Authentication bypass -- High****Vulnerability Exploited:****Multi-Factor Authentication (MFA) Bypass****Severity:**** High****System:**** AAVR (Application Attacks) **Vulnerability Explanation:**  This vulnerability allows an attacker to bypass the Multi-Factor Authentication (MFA) mechanism implemented in the web application. By exploiting this flaw, the attacker can gain access to user accounts without needing the second authentication factor (e.g., a one-time password), potentially accessing sensitive information or performing unauthorized actions on behalf of the user.  Vulnerability Walk-thru:   Navigate to the login page of the application.   Enter valid credentials (e.g., wiener :peter).  . observe the HTTP request made to the server and capture the request made after the first authentication step . Instead of entering the 2FA code, replay the initial request to the server or access a protected URL directly: Change in URL : “login2” to “my-account”, gained from your URL . resubmitting the login POST request or accessing a dashboard URL resulted in successful login without requiring the 2FA code.. Verify that you have accessed the user dashboard or other protected areas. | |  |
|  |  | |  |
|  |  | |  |
|  | Recommendation:  . Ensure that the application properly validates the second authentication factor before granting access to protected resources. The session should not be fully established until the 2FA step is completed successfully  . Use separate session tokens for the first and second authentication factors. The final session token should only be issued after successful 2FA validation.  . Implement logging for all login attempts, especially failed 2FA attempts.  . Use anomaly detection to identify and respond to unusual login patterns.  . Invalidate the initial session or token immediately after the user is prompted for 2FA to prevent replay attacks | |  |
|  |  | |  |