Report 05

URL: www.enviso.io

Cross-site Request Forgery

MEDIUM

CSRF (Cross-Site Request Forgery) is a type of security vulnerability that allows an attacker to manipulate a victim into unknowingly performing malicious actions on a web application in which the victim is authenticated. The vulnerability occurs due to the web application's failure to adequately verify the source of incoming requests. In a CSRF attack, the attacker tricks the victim into visiting a malicious website or clicking on a malicious link. When the victim interacts with the attacker's website, their web browser automatically sends a request to the vulnerable web application, including any relevant authentication cookies.

The web application, assuming the request is legitimate, processes the request and performs the requested action on behalf of the victim. This can lead to unauthorized actions being executed, such as changing account settings, making fraudulent transactions, deleting data, or performing other actions available to the victim's authenticated session.

Affected URL : https://www.enviso.io/contact/index.html

Affected Components

The affected components in a CSRF vulnerability typically include the web application's functionality and features that require user authentication and session management.

- User Authentication: The process of logging in and managing user sessions can be vulnerable to CSRF attacks if proper measures are not in place to validate the authenticity of requests.

- User Account Settings: Actions related to user account settings, such as changing passwords, updating personal information, or modifying communication preferences, can be targeted by CSRF attacks.

- Data Modification: Any functionality that allows users to create, update, or delete data within the application, such as submitting forms, posting comments, or making purchases, may be affected.

- Actions with Side Effects: Actions that have significant consequences or perform irreversible actions, such as deleting user accounts, initiating financial transactions, or altering critical system configurations, are potential targets for CSRF attacks.

- API Endpoints: If the web application exposes APIs that perform actions on behalf of authenticated users, these endpoints should also be protected against CSRF attacks to prevent unauthorized actions through API calls.

Impact Assessment

The impact assessment of a CSRF (Cross-Site Request Forgery) vulnerability can vary depending on the specific actions that can be performed by the targeted web application.

- Unauthorized Actions: An attacker can trick a victim into unknowingly performing unauthorized actions on the web application. This can include actions such as changing account settings, making fraudulent transactions, submitting forms with malicious content, or deleting important data.

- Data Manipulation: CSRF attacks can lead to the manipulation or unauthorized modification of sensitive data within the application. This can result in the compromise of user information, financial data, personal records, or other confidential data stored or processed by the application.

- Account Takeover: By exploiting CSRF vulnerabilities, an attacker can potentially gain control over a victim's account. This allows them to access the victim's personal information, perform actions on their behalf, or even escalate privileges within the system.

- Reputation Damage: Successful CSRF attacks can negatively impact the reputation and trustworthiness of the affected web application or organization. Users may lose confidence in the security of the application, leading to a loss of business, customer trust, and potential legal or regulatory repercussions.

- Malware Injection: CSRF attacks can be used as a vector to inject malicious content or scripts into the web application, potentially leading to the installation of malware on the victim's device or compromising the security of other users accessing the application.

Steps to Reproduce

- Identify the target: Choose a web application that you suspect may have CSRF vulnerabilities. This can be your own application for testing purposes or a publicly accessible website that you want to evaluate.

- Analyze the target application: Explore the functionality of the application and identify actions that require authentication or perform sensitive operations. Look for forms,

actions, or requests that modify data, change settings, or perform actions on behalf of the user.

- Craft a malicious webpage: Create a webpage or HTML document that contains malicious code. This can be a simple HTML form or an AJAX request that mimics a legitimate action within the target application.

- Set up the attack scenario: Host the malicious webpage on a separate server or a local environment accessible to your browser. Ensure that the victim (authenticated user) and the attacker (you) have separate browser instances or devices.

- Initiate the attack: Trick the victim into visiting the malicious webpage by sending them a link or embedding the link in an email, message, or other communication. The victim should be logged in to the target application in their browser session.

- Exploit the vulnerability: The victim's browser will automatically send the HTTP request to the target application when they visit the malicious webpage. This request will contain any relevant authentication cookies or session information. The target application, assuming the request is legitimate, will process the action requested by the attacker.

Proposed Mitigation or Fix

✓ Implement CSRF tokens: Include a unique and unpredictable token in each form or request that performs sensitive actions. This token should be validated on the server-side to ensure that the request originated from a legitimate source. Tokens should be resistant to guessability and should expire after a certain period or after being used.

- ✓ Same-Site cookies attribute: Set the Same-Site attribute on session cookies to restrict their scope to the same origin. This prevents the browser from sending cookies in cross-site requests, mitigating the risk of CSRF attacks.

- ✓ Cross-Origin Resource Sharing (CORS): Implement CORS on the server-side to control which domains are allowed to make requests to the application. By defining specific allowed origins, you can prevent unauthorized cross-origin requests from being processed.

- ✓ Double Submit Cookies: Generate a random CSRF token and store it as a cookie in the user's browser. This token should also be included as a parameter in each request. The server can then compare the token in the cookie with the token in the request to verify its authenticity.

- ✓ Referer Header Check: Validate the Referer header in incoming requests to ensure that they originate from the same domain. While this method is not foolproof and can be bypassed in some cases, it adds an additional layer of protection.

- ✓ Secure-by-Design Practices: Follow secure coding practices, such as input validation, output encoding, and proper session management, to minimize the risk of vulnerabilities that can be exploited for CSRF attacks.