ijecs open access www.ijecs.in
International Journal Of Engineering And Computer Science
Volume 13 Issue 12 December 2024, Page No. 26677-26681
ISSN: 2319-7242 DOI: 10.18535/ijecs/v13i12.4950

Analysis of Vulnerabilities in The Content Security Policy Standard to Enhance Website Security

Okhonko Pylyp [1]

[1] Application Security Engineer, Tential Rockville, Maryland, United States

Abstract

This article analyzes the vulnerabilities of the Content Security Policy (CSP) standard with the aim of improving website security. It highlights that this standard has vulnerabilities that enable attackers to successfully execute attacks by injecting malicious code into web pages. The relevance of studying CSP vulnerabilities to enhance website protection is substantiated. The primary aspects of implementing web resource protection using the CSP standard are discussed. The article outlines the most well-known techniques for bypassing the directives of this standard. It describes the process of executing an XSS attack using the "unsafe-inline" directive. It is revealed that such an attack organization allows the interception of user data without detection by users or security administrators. The conclusion is made that the use of the unsafe-inline directive by developers does not provide an adequate level of protection against XSS attacks. As an alternative, the implementation of a more effective CSP policy, configured in accordance with the recommendations of information security specialists, is proposed.

Keywords: *information security, Content Security Policy, Cross-Site Scripting, website protection, security standard vulnerabilities.*

1. Introduction

Ensuring data protection within the operation of web resources has long been a critical issue. This type of security is implemented through various means, one of the most relevant being a distinct layer known as Content Security Policy (CSP). This mechanism is designed to detect and mitigate attacks based on cross-site scripting and the injection of malicious data into transmitted requests. It operates using a set of directives that enable flexible configuration of protection mechanisms.

Despite its widespread popularity, this standard for securing web services and applications has several shortcomings. Studying these vulnerabilities will help identify ways to address them for a broad range of specialists. In practice, however, despite the existence of numerous directives aimed at providing diverse protections against code injection, attackers continue to find ways to bypass them. For this reason, this article explores these bypass methods and introduces a query redirection technique that has not been previously discussed.

2. Materials and Methods

The article begins by defining Stored XSS attacks, a vulnerability in web systems that allows attackers to embed malicious code (commonly JavaScript) into a webpage [1]. Upon successful execution of such an attack, an attacker may intercept login credentials, personal user data, or other sensitive information [2]. Essentially, XSS attacks exploit a browser's trust in the content displayed, as retrieved from a server platform, leading to the execution of all received scripts.

Modern web services have been developed to eliminate the possibility of XSS vulnerabilities. Browser developers have also contributed by implementing tools for input validation and policies such as Same-Origin Policy (defining access permissions for scripts) and Content Security Policy (specifying trusted script sources). However, vulnerabilities persist, and this article examines such weaknesses within the CSP standard. Content Security Policy (CSP) is an additional layer of protection for web services and applications designed to prevent script injection attacks and the execution of unauthorized code. CSP achieves protection by specifying a list of trusted resource sources [3]. If a resource or script originates from an unlisted source, the browser simply does not load it. This prevents the execution of third-party code and mitigates the risks of malicious code injection, which can lead to a range of outcomes—from data interception to website malfunction [4].

The CSP standard has multiple versions that maintain compatibility, except for the second version, which has some inconsistencies with other versions. However, browsers can still operate with servers using various versions of the standard. If a CSP header is not provided by the server, the browser applies a default policy for that source [5, 6].

To define trusted sources, a CSP-compliant web page must include a specialized `Content-Security-Policy` header and a set of directives, such as:

- `img-src` – specifies sources for image loading;
- `media-src` – specifies sources for media content (videos, animations, audio);
- `script-src` – specifies sources for loading scripts for the web page;
- `frame-src` – specifies sources for web elements;
- `default-src` – a fallback directive. If one or more of the above directives lacks arguments, the browser uses the sources specified in this directive [3, 6].

Using these directives, developers can define the domains, subdomains, or individual pages permitted for loading, as well as set interaction rules. For this, two values can be specified: `"none"`, which blocks access, and `"self"`, which permits the use of resources from the specified source. In addition to general rules, individual directives have specific rules. For example, the `unsafe-inline` rule allows the use of JavaScript scripts directly embedded in a webpage's code. If this rule is not specified, any script not sourced from a trusted origin will be blocked. The `unsafe-eval` rule enables or blocks (by default) the execution of dynamic code during runtime [7].

To implement CSP policies, developers must configure the appropriate header to be returned by the web page. For testing CSP policies, the `Content-Security-Policy-Report-Only` directive is available. When used, scripts are not blocked, but all violations and deviations are logged in reports sent to the system administrator.

However, despite the seemingly straightforward implementation and the flexibility in configuring trusted sources for web resources, CSP has vulnerabilities that attackers can exploit to execute XSS threats [8]. One identified vulnerability involves the use of the `unsafe-inline` rule in the `script-src` directive. While this directive is intended to limit the ability to transmit confidential information to external sources, if not included, developers must manually compile an extensive list of resources where such data might be transmitted. Misconfigurations of this directive can limit application functionality. For this reason, developers often rely on this directive, assuming it provides protection against data transmission to other resources. However, in practice, this assumption does not always hold true.

3. Results and Discussion

The conducted study revealed that employing the redirect method enables the acquisition of all necessary parameters. This is achieved by constructing a URL containing the required parameters within a GET request. An example of such a request is shown in Figure 1:

```
const e = encodeURIComponent;
const url = `http://attacker-server.com/?u=${e(location.href)}&c=${e(document.cookie)}
&d=${e(document.documentElement.innerHTML)}&l=${e(JSON.stringify(localStorage))}`;
document.location = url;
```

Figure 1 - URL construction process and transmission of confidential data to the intercepting server

The URL above allows the retrieval of session cookies; however, this method has a significant drawback: the attack redirects the user to the intercepting server, potentially disrupting the operation of the targeted website. To address this limitation, the data interception method must be refined. The intercepting server should be configured to accept the data without returning a response, keeping the connection open. Then, a redirection to the intercepting server with the required parameters is initiated, followed by the execution of the `window. stop` procedure, which cancels the redirection and renders the attack invisible to the user. The visual process of data interception using this method is depicted in Figure 2 and can be described in three stages.

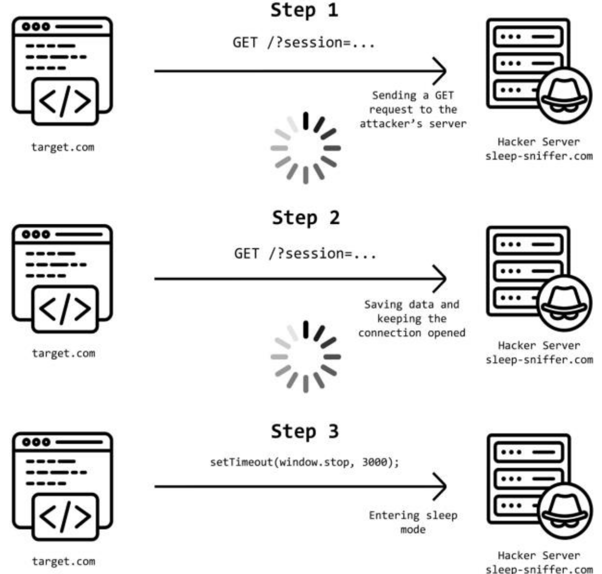

Figure 2 - Implementation process of an XSS attack on a page using the CSP "unsafe-inline" directive

In the first stage, a redirection to the specified server, sleep-sniffer.com, occurs, along with the transmission of the session attributes. During this time, a loading indicator is displayed on the page, leading the user to believe that the page is loading as expected.

In the second stage, the intercepting server receives the request, saves the transmitted parameters, and holds the connection open without sending a response. A key aspect of this method is that while waiting for a response, the browser perceives the page as functioning normally, executing all its scripts. The user sees the loading indicator and continues to wait for the process to complete.

After three seconds, a `set Time out` is triggered, transitioning to the third stage. The execution of the `window. stop` method cancels the loading of all resources, including multimedia, fonts, and styles. Any incomplete requests are simply terminated. The webpage then continues to function as usual, all necessary data is transmitted to the sleep-sniffer.com server, and the user remains unaware of the attack. To implement the data interception method using redirection, a utility was developed. The interface of this utility is shown in Figure 3 [9]. It allows specifying the parameters to be transmitted in the request, as well as the addresses of the targeted webpage and the intercepting server.

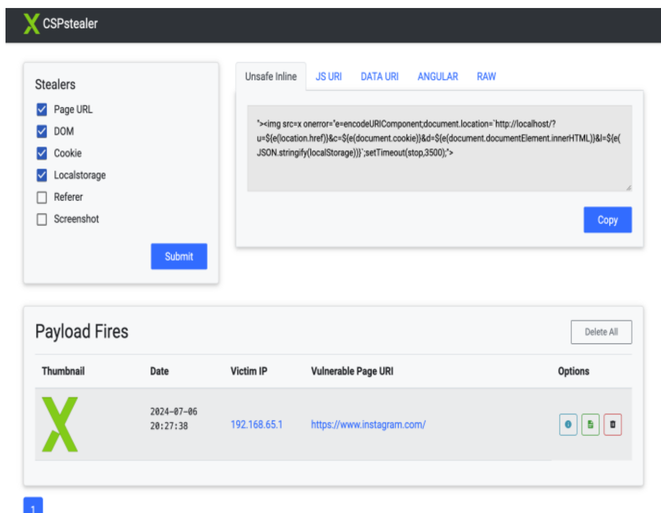

Figure 3 - Utility interface for executing redirect requests with various data sets [9]

To eliminate the described possibility of redirecting with the required parameters, the CSP standard introduced the "navigate-to" directive. This directive allows specifying a list of external

sources to which redirection from the current page is permitted. On one hand, it enables defining a list of domains or addresses where requests can be executed. On the other hand, creating such a list requires considerable effort in its formation and constant updates to maintain its relevance. Therefore, the only way to ensure a high level of security is to avoid using the `unsafe-inline` directive and to implement a more effective CSP policy, regardless of the labor and financial costs involved.

The CSP standard includes numerous directives and their configurations, which in some cases may create vulnerabilities that attackers can exploit to execute XSS attacks [8]. Additionally, other known methods for bypassing this standard include:

- Injecting scripts through file upload mechanisms.
- Injecting scripts using public CDNs.
- Injecting scripts through Third-Party EndPoints combined with JSONP.
- Injecting scripts through Angular procedures.

However, these methods only address code injection into a webpage and do not solve the problem of transmitting confidential data from the page to the attacker's server. As with the enabled `unsafe-inline` option, the restriction allowing communication only with trusted web resources remains, preventing data transmission from the page to the attacker's server. In this case, the aforementioned method can be combined with other known CSP bypass techniques to circumvent this restriction.

Table 1 - Example of payload generated using the CSP Stealer utility and the CSP bypass method via the Third-Party End Points + JSONP mechanism

```
"><script
src="/api/jsonp?callback=(function(){e=encodeURIComponent;document.location=`http://attacke
r-
server.com/?u=${e(location.href)}&c=${e(document.cookie)}&d=${e(document.documentElem
ent.innerHTML)}&l=${e(JSON.stringify(localStorage))}`;setTimeout(stop,3500);})();//"></scrip
t>
```

4. Conclusion

The exploitation of any of the vulnerabilities mentioned above allows attackers to intercept various types of information, including authentication data. Combining these methods broadens the range of potential XSS attack techniques, thereby increasing the workload for security specialists. In conclusion, it is essential to emphasize that despite the existence of the CSP mechanism aimed at protecting against script injection into webpages, its effectiveness is often insufficient, and attackers continuously find new ways to bypass it. This study highlighted the critical risk associated with using the `unsafe-inline` directive. Therefore, the only way to ensure a high level of security is to avoid using the `unsafe-inline` directive and to implement a more robust CSP policy, regardless of the required labor and financial investments.

References

1. Singh, M., & Chauhan, A.S. (2016). DOM BASED REFLECTED XSS ATTACK USING SOP.
2. Jasmine M. S., Devi K., George G. Detecting XSS based web application vulnerabilities //International Journal of Computer Technology & Applications. – 2017. – T. 8. – No. 2. – pp. 291-297.
3. Yusof I., Pathan A. S. K. Mitigating cross-site scripting attacks with a content security policy //Computer. – 2016. – T. 49. – No. 3. – pp. 56-63.
4. Golinelli M., Bonomi F., Crispo B. The Nonce-nce of Web Security: An Investigation of CSP Nonces Reuse //European Symposium on Research in Computer Security. – Cham: Springer Nature Switzerland, 2023. – pp. 459-475.
5. Content Security Policy (CSP) [Electronic resource].URL: https://developer.mozilla.org/ru/docs/Web/HTTP/CSP (Date of access: 08/15/2024)
6. Weichselbaum L., Spagnuolo M., Janc A. Adopting strict content security policy for XSS protection //2016 IEEE Cybersecurity

Development (SecDev). – IEEE, 2016. – pp. 149-149.
7. Kerschbaumer C., Stamm S., Brunthaler S. Injecting CSP for fun and security //International Conference on Information Systems Security and Privacy. – SCITEPRESS, 2016. – T. 2. – P. 15-25.
8. Weissbacher M., Lauinger T., Robertson W. Why is CSP failing? Trends and challenges in CSP adoption // Research in Attacks, Intrusions and Defenses: 17th International Symposium, RAID 2014, Gothenburg, Sweden, September 17-19, 2014. Proceedings 17. – Springer International Publishing, 2014. – pp. 212-233.
9. GitHub - sysmustang/csp-stealer: Tool to retrieve web-page secrets and bypass Content Security Policy [Electronic resource].URL: https://github.com/sysmustang/csp-stealer