ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 4, Issue 7, July 2017

Mitigating Malicious Scripting Attacks with a Content Security Policy

K.Veena Devi
MPhil (Research Scholar)
Department of computer science
Prist University, Thanjavur

R.Arivumalar
Research Supervisor
Department of computer science
Prist University, Thanjavur

Abstract: A content security policy (CSP) can help Web application developers and server administrator's better control website content and avoid vulnerabilities to cross site scripting (XSS). In experiments with a prototype website, the authors' CSP implementation successfully mitigated all XSS attack types in four popular browsers. Among the many attacks on Web applications, cross site scripting (XSS) is one of the most common. An XSS attack involves injecting malicious script into a trusted website that executes on a visitor's browser without the visitor's knowledge and thereby enables the attacker to access sensitive user data, such as session tokens and cookies stored on the browser.1 With this data, attackers can execute several malicious acts, including identity theft, key logging, phishing, user impersonation, and webcam activation**.** Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP is designed to be fully backward compatible; browsers that don't support it still work with servers that implement it, and vice-versa. Browsers that don't support CSP simply ignore it, functioning as usual, defaulting to the standard same-origin policy for web content. If the site doesn't offer the CSP header, browsers likewise use the standard same-origin policy. Enabling CSP is as easy as configuring your web server to return the Content-Security-Policy HTTP header. (Prior to Firefox 23, the X-Content-Security-Policy header was used). See Using Content Security Policy for details on how to configure and enable CSP.

Keywords: CSP (content security policy), XSS (Cross site Scripting), Data Attack, Scripting.

I. INTRODUCTION

Researchers have proposed a range of mechanisms to prevent XSS attacks, with content sanitizers dominating those approaches. Although sanitizing eliminates potentially harmful content from untrusted input, each Web application must manually implement it—a process prone to error. To avoid this problem, we use a different technique. Instead of sanitizing harmful scripts before they are injected into a website, we block them from loading and executing with a variation of the content security policy (CSP), which provides server administrators with a white list of accepted and approved resources. The Web application or website will block any input not on that list and thus there is no need for sanitizing. The white list also guards against data exfiltration and extrusion—the unauthorized downloading of data from a website visitor's computer.

Our variation of CSP 1.0, a World Wide Web Consortium (W3C) standard, uses directives such as report-uri, which lets server administrators either save policy violations to a log file or receive them as email. In the report-only mode, this directive lets administrators conduct a dry run of the website with a particular CSP and note XSS attack types— information that can be shared in the security community to inform both practical case implementations and research projects.

Existing defense mechanisms tend to focus on preventing one or two of the three XSS attack types, but our CSP is the first that we know of to mitigate all three. To test its effectiveness, we created a prototype website on a simple Web application server, configured a CSP header with a report-uri directive, and incorporated the header in an .htaccess file (a configuration file that specifies how a webpage should be accessed). We then conducted a series of experiments on our local host machine by injecting XSS vectors into the website. In every case, our CSP prevented XSS attacks, even with 50 unique XSS vectors.

II. RELATED WORK

IN XSS ATTACK PREVENTION others have proposed mechanisms to prevent XSS attacks.4 Noxes, a client- side tool that acts as a Web proxy, disallows requests

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 4, Issue 7, July 2017

that do not belong to the website and thus thwarts stored XSS attacks. Browser-enforced embedded policies (BEEPs) let the Web application developer embed a policy in the website by specifying which scripts are allowed to run.6 With a BEEP, the developer can put genuine source scripts in a white list and disable source scripts in certain website regions. Document Structure Integrity (DSI) is a client-server architecture that restricts the interpretation of un-trusted content.7 DSI uses parser-level isolation to isolate inline untrusted data and separates dynamic content from static content. However, this approach requires both servers and clients to cooperatively upgrade to enable protection.
Blueprint is a server-side application that encodes content into a model representation that the client-side part can process. However, applying Blueprint to Word press increased processing time on average 55 percent; applying it to MediaWiki increased processing time an average 35.6 percent.8 implementing our CSP can help Web application developers specify allowable content type and resource locations and can be an early warning system for any policy violations, which greatly assists system administrators' website control. With little or no modification to application source code, website visitors are assured of protection from the unauthorized downloading of the sensitive data stored in their browsers. The CSP's report-only mode along with the report-uri directive gives server administrators the option to test and configure their applications without breaking website functionalities. Although our CSP has many benefits, it is not intended as a primary defense mechanism against XSS attacks. Rather, it would best serve as a defense in-depth mitigation mechanism. A primary defense involves tailored security schemes that validate user inputs and encode user outputs. So far our work has involved CSP 1.0. In future work, we plan to investigate using directives with CSP 2.0, which as of February 2016 was still a W3C working draft.

III. System Analysis

EXISTING SYSTEM

Thus system will send request with identity. After that all the collected information will be send to collection database server. It not only protects clients from XSS attacks but also inform the vulnerable web servers.

This mechanism requires minimal effort and low performance overhead. Also, it will prevent all the types of XSS attacks.

Disadvantages

- How to use the collected information in database is not addressed.
- How to make system deployed universally has also not been addressed.
- It requires modifications in the frameworks or installation of additional frameworks.
- Approved scripts have to be identified by the website.

PROPOSED SYSTEM

A client-side tool that acts as a Web proxy, disallows requests that do not belong to the website and thus thwarts stored XSS attacks. Browser-enforced embedded policies (BEEPs) let the Web application developer embed a policy in the website by specifying which scripts are allowed to run. With a BEEP, the developer can put genuine source scripts in a white list and disable source scripts in certain website regions. Document Structure Integrity (DSI) is a client-server architecture that restricts the interpretation of untrusted content. DSI uses parser-level isolation to isolate inline untrusted data and separates dynamic content from static content. However, this approach requires both servers and clients to cooperatively upgrade to enable protection.

IV. Implementation

An XSS attack can be persistent or nonpersistent, or it can be based on a document object model (DOM).

1) MODULES

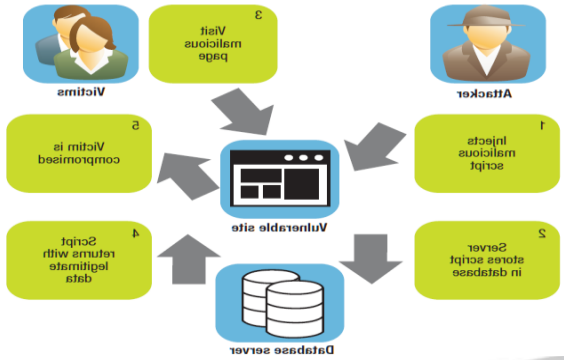
1. Persistent XSS
2. Non-persistent XSS
3. DOM-based XSS

Persistent XSS

A persistent XSS attack does not need a malicious link for successful exploitation; simply visiting the webpage will compromise the user. Persistent XSS is often difficult to detect and is considered more harmful than the other two attack types. Because the malicious script is rendered automatically, there is no need to target individual victims or lure them to a third party website.

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

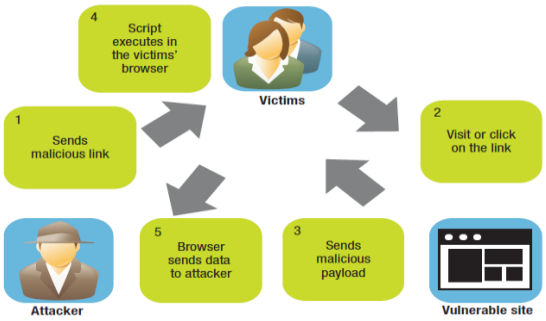
International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 4, Issue 7, July 2017



Non-persistent XSS

A non-persistent, or reflected, XSS attack, which occurs when a website or Web application passes invalid user inputs. Usually, an attacker hides malicious script in the URL, disguising it as user input, and lures victims by sending emails that prompt users to click on the crafted URL.

DOM-based XSS



The attack occurs when the victim's browser executes the malicious code from the modified DOM. On the client side, the HTTP response does not change but the script executes maliciously. This exploit works only if the browser does not modify the URL characters. A DOM-based XSS attack is the most advanced type and is not well known. Indeed, much of the vulnerability to this attack type stems from the inability of Web application developers to fully understand how it works.

V. Conclusion

CSP can't (not intended to) fix everything At best, a backup tool to help mitigate issues that arise. Remember: If an attacker can modify the web app files, they can modify the CSP. Security is hard .Content Security Policy allows for fine-grain access control over resource origin for web applications. The implementation of a basic form of Content Security Policy can be easily accomplished on any website. The Content-Security-Policy-Report-Only header gives the developers/administrators a process for adding Content Security Policy to a web application without breaking the functionality. The removal of inline script, inline style, and insecure functions can be a large task, so the Content Security Policy can be configured to allow these until the application code can be properly modified. If the prerequisite work is done, Content Security Policy can mitigate common content injection vulnerabilities. The best practices proposed above give the developers a guideline for designing a secure policy and avoiding some common missteps in the process.

VI. Future Enhancements

The project has covered almost all the requirements. Further requirements and improvements can easily be done since the coding is mainly structured or modular in nature. Improvements can be appended by changing the existing modules or adding new modules. One important development that can be added to the project in future is secure the data from the External Scripting and also on cross site Attacks and also we can implement new technology to promote and secure the data

Implementation is the stage of the project when the theoretical design is turned out into a working system. Thus it can be considered to be the most critical stage in achieving a successful new system and in giving the user, confidence that the new system will work and be effective.

References

[1]. Stefano Calzavara, Alvise Rabitti, and Michele Bugliesi. Content security problems?: Evaluating the effectiveness of content security policy in the wild. In Proceedings of the 2016 ACMSIGSAC Conference on Computer and Communications Security, pages 1365–1375. ACM, 2016.

[2]. WorldWide Web Consortium. Content security policy level 3,https://www.w3.org/TR/CSP/, 2016.

[3]. Google. Reshaping web defenses with strict content securitypolicy, https://security.googleblog.com/2016/09/reshapingweb-defenses-with-strict.html, 2016.

[4]. Engin Kirda, Christopher Kruegel, Giovanni Vigna, and Nenad Jovanovic. Noxes: a client-side solution for mitigating cross-sitescripting attacks. In Proceedings of the 2006 ACM symposium on Applied computing, pages 330–337. ACM, 2006.

[5]. Hao Chen Matthew Van Gundy. Noncespaces: Using randomizationto defeat cross-site scripting attacks. El Sevier.

[6]. Kevin Spett. Cross-site scripting. SPI Labs, 1:1–20, 2005.

[7]. Sid Stamm, Brandon Sterne, and Gervase Markham. Reining in the web with content security policy. In Proceedings of the 19th

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 4, Issue 7, July 2017

international conference on World Wide Web, pages 921–930. ACM, 2010.

[8]. Michael Hicks Trevor Jim, Nikhil Swamy. Defeating script injection attacks with browser-enforced embedded policies (beep).ACM.

[9]. Lukas Weichselbaum, Michele Spagnuolo, Sebastian Lekies, and Artur Janc. Csp is dead, long live csp! on the insecurity of whitelists and the future of content security policy. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pages 1376–1387. ACM, 2016

[10]. I. Yusof and A.-S.K. Pathan, "Preventing Persistent Cross-Site Scripting (XSS) Attack by Applying Pattern Filtering Approach," Proc. 5th IEEE Conf. Information and Communication Technology for the Muslim World (ICT4M14), 2014, pp. 1−6.

[11]. L.K. Shar and H.B.K. Tan, "Defending against Cross-Site Scripting Attacks," Computer, vol. 45, no. 3, 2012, pp. 55−62.

[12]. E. Kirda et al., "Noxes: A Client-Side Solution for Mitigating Cross-Site Scripting Attacks," Proc. 21st Ann. ACM Symp. Applied Computing (SAC06), 2006, pp. 330−337.