Volume 3, Issue 3, March-2016, pp. 123-128 ISSN (O): 2349-7084
International Journal of Computer Engineering In Research Trends
Available online at: www.ijcert.org

Smart Content Security Policy for Mozilla Firefox

[1]Dr. Kailas R. Patil, [2]Madhulika Jhawar, [3]Rashida Mumin, [4]Aashay Mokadam and [5]Kunal Kharsadia

[1]*Professor, Vishwakarma Institute of Information Technology, Pune*
[2,3,4,5] *B.E. Computer Engineering, Vishwakarma Institute of Information Technology, Pune.*

Abstract: - Use of Internet is very common these days. There are a lot of websites containing different kinds of content over the internet. An average person visits atleast 8-10 web pages per day. The web page sometimes contains malicious contents or some inline scripts which the user is not aware of. The scripts try to force display contents on user's screen or try to steal information from the user without the user's awareness. This makes the user vulnerable. Content Security Policy is a way to defeat these types of attacks. If CSP is enforced on the web browser, content will be displayed from trusted sources only and all other contents will be blocked. UserCSP was implemented to facilitate this purpose. In UserCSP, users could specify the policies they want to enforce. However, this had a drawback too. An average user is not familiar with the concept of CSP. So, we are making SmartCSP, an add-on for Mozilla Firefox. This will facilitate the users who are not aware of CSP by inferring the policies based on the structure of HTML pages loaded.

Keywords – Content Security Policy, Code Injection Attacks, UserCSP, SmartCSP

1. INTRODUCTION

The Websites suffer from different kinds of malicious attacks. Most of the end users and website developers are not really aware of the process of enforcing content security policies. Smart CSP has been inspired from user's inability to enforce policies. Smart CSP infers the policies automatically, if not specified by users, depending on the structure of page loaded and different elements and sources.

The web browser security model is rooted in the same-origin policy (SOP), which isolates one origin's resources from other origins. However, attackers can subvert the SOP by injecting malicious content into a vulnerable website through attacks such as Cross-Site Scripting (XSS). According to the OWASP vulnerability assessment in 2013, XSS attacks are among top five vulnerabilities. The root cause of code injection problem on websites is that browsers are unable to distinguish between legitimate and maliciously injected content in a web application. [12]

To mitigate threats of XSS attacks, Mozilla proposed Content Security Policy (CSP) a defense-in-depth. CSP has become a part of W3C specification and CSP 1.0 is in the state of Candidate Recommendation. It aims to solve this problem by providing a declarative content restriction policy in an HTTP header that the browser can enforce. CSP defines directives associated with various types of content that allow developers to create whitelists of content sources and instruct client browsers to only load, execute, or render content from those trusted sources. However, writing an effective and comprehensive CSP policy for websites is laborious. A policy can break website functionality if legitimate content is overlooked during policy generation. [9]

Introduction to UserCSP helped developers and users write comprehensive policies for websites by providing them with a GUI to add and modify CSP policies.

UserCSP monitors the browsers internal events (including HTML parsing, HTTP requests, and XHR requests triggered by scripts running in the JS engine). It then dynamically analyzes the content type loaded by a webpage and the source of that content. This information is useful to automatically infer the policy for a webpage. Major problem with UserCSP was that the group of end users who are not aware about the need and implementation of content security policy and thus weren't able to give necessary specifications in UserCSP add-on to eliminate malicious content.

In this paper, we have introduced SmartCSP, which is like an enhancement over UserCSP. Even if the user doesn't specify content security policies, he'd still be protected with the auto-inferring capability of SmartCSP.

2. RELATED WORK

A. Content security policy

Content security policy(CSP) is a computer security standard introduced to prevent cross-site scripting (XSS), clickjacking and other code injection attacks resulting from execution of malicious content in the trusted web page context. CSP provides a standard method for website owners to declare approved origins of content that browsers should be allowed to load on that website.

B. UserCSP

The goal of UserCSP is to allow users to specify and apply security policies on web content. UserCSP helps developers and users write comprehensive policies for websites by providing them with a GUI to add and modify CSP policies.

- If the website has defined a CSP policy, but the user hasn't, then UserCSP does not interfere with the website defined policy. However, it does allow the user the option to amend the websites policy.
- If a user has specified a CSP policy for a website, but the website administrator hasn't, then the user's policy is enforced.
- If both a user specified CSP policy and a website defined policy exist, then the user has a choice to either apply their own policy or adopt the website defined policy. Moreover, users can choose to combine their custom policy with an existing website policy by selecting a strict (intersection) or loose (union) combination policy.
- If neither the user nor the website specify a CSP policy, but the user has specified a global policy that can be used for websites that do not have site-specific policies defined, then UserCSP will apply the global policy.

If neither the user nor the website specify a CSP policy, and there is no global policy, then UserCSP does not affect the content loading on the website. To allow automatic policy inference for websites, UserCSP uses dynamic analysis to monitor content loaded by a webpage and recommends a CSP policy based on the content types and content sources included in the webpage. It also monitors the resources dynamically added to the webpage by JavaScript.[5]

3. PROPOSED METHOD

A. Introduction to SmartCSP

The goals of Smart CSP are three-fold:

- To allow security savvy users to specify their own CSP policies.
- To allow developers to experiment with CSP policies on their production pages.
- To infer the pattern and block unwanted sites and contents automatically.

SmartCSP is like an enhancement over UserCSP. It provides facility of inferring the policies itself if the user or web developer hasn't specified them. This will facilitate the users, who are unaware of content security policy and its implementation, to secure their web content.

Not all attacks are successfully overcome with SmartCSP, but SmartCSP is extensively used for stopping code injection attacks.

B. Code Injection Attacks

Code injection is the exploitation of a computer bug that is caused by processing invalid data. Injection is used by an attacker to introduce (or "inject") code into a vulnerable computer program and change the course of execution. The result of successful code injection is often disastrous (for instance: code injection is used by some computer worms to propagate).

Injection flaws occur when an application sends untrusted data to an interpreter. Injection flaws are very prevalent, particularly in legacy code. Injection can result in data loss or corruption, lack of

accountability, or denial of access. Injection can sometimes lead to complete host takeover.[10]

Code injection techniques are popular in system hacking or cracking to gain information, privilege escalation or unauthorized access to a system. Code injection can be used malevolently for many purposes, including:

- Install malware or execute malevolent code on a server, by injecting server scripting code (such as PHP or ASP).
- Privilege escalation to root permissions by exploiting Shell Injection vulnerabilities in a setuid root binary on UNIX, or Local System by exploiting a service on Windows.
- Attacking web users with HTML/Script Injection (Cross-site scripting).

a. *Cross-site Scripting*: Cross-site Scripting ("XSS") is a type of injection attack, in which malicious scripts are introduced into the trusted websites. These attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. A cross-site scripting vulnerability may be used by attackers to bypass access controls such as the same-origin policy. Their effect may range from a petty nuisance to a significant security risk, depending on the sensitivity of the data handled by the vulnerable site and the nature of any security mitigation implemented by the site's owner.[10]

b. *Object injection*: PHP allows serialization and deserialization of whole objects. If untrusted input is allowed into the deserialization function, it is possible to overwrite existing classes in the program and execute malicious attacks.[10]

c. *Shell Injection Attacks*: It is also called "OS Command Attacks". This class of attacks exploits applications which use input to formulate commands that are executed by the OS. The user supplies all or part of malformed OS command through a web interface. If the web interface is not properly sanitized the input is vulnerable to this exploit. With the ability to execute OS commands, the user can inject unexpected and dangerous commands, upload malicious programs or even obtain passwords directly from the operating system.[10]

d. *Remote File Inclusion*: A very sneaky method of running malicious software on a victim's server is by simply asking it to go somewhere else on the Internet to find a dangerous script, and then run it from that location. This scary scenario is called a Remote File Inclusion (RFI) attack. An RFI can occur when functions are improperly crafted, allowing users to modify the URL parameters when web apps are launching components for their own purposes. By changing the intended process in order to activate a faraway malicious payload sitting on a public server, the attacker may be able to activate a piece of code that will give them a shell through a held connection between the victim site and the remote server that holds the designated file. Including a script in this way opens up a number of dangerous options that a hacker can use against the user.[10]

e. *Clickjacking*: Clickjacking (User Interface redress attack, UI redress attack, UI redressing) is a malicious technique of tricking a Web user into clicking on something different from what the user perceives they are clicking on, thus potentially revealing confidential information or taking control of their computer while clicking on seemingly innocuous web pages. It is a browser security issue that is a vulnerability across a variety of browsers and platforms. A clickjack takes the form of embedded code or a script that can execute without the user's knowledge, such as clicking on a button that appears to perform another function. A clickjacked page tricks a user into performing undesired actions by clicking on a concealed link. On a clickjacked page, the attackers load another page over it in a transparent layer. The users think that they are clicking visible buttons, while they are actually performing actions on the hidden/invisible page. The hidden page may be an authentic page; therefore, the attackers can trick users into performing actions which the users never intended. There is no way of tracing such actions to the attackers later, as the users would have been genuinely authenticated on the hidden page.[10]

4. IMPLEMENTATION OF PROPOSED WORK

A. Automatic Inference

To allow automatic policy inference for websites, SmartCSP uses an algorithm that performs dynamic analysis to monitor content loaded by a webpage and recommends a CSP policy based on the content types and content sources included in the webpage. It also monitors the resources dynamically added to the webpage by JavaScript. To record new content introduced by websites at run-time, SmartCSP during learning phase continuously monitor websites even after website is completely loaded. It records inferred policy into local database. Next time, when the user visits the same site SmartCSP takes previously inferred policy and combine it with the currently inferred policy. Due to rotating advertisements that change periodically may lead to a load request from different origins, therefore, the continuous inferring process of SmartCSP helps users to detect changes in the resource origin and apply changed domain to reload resources. Our inferred policy derives strict CSP policy, which blocks in- line scripts, styles, eval, and event handlers. However, SmartCSP also provides features to users to allow inline scripts and eval on their favorite websites manually. In modern browsers, extensions are high privilege than web- sites and run with the privileges of the browser. Browser extensions are used to enhance user experience and provide new functionality. Therefore, SmartCSP honors content included by extensions into web pages and includes them into inferred policy. [12]

B. Constraints and Challenges

1) *Websites using inline script:* One of the primary security benefits of CSP is that it disallows unsafe coding practices such as inline scripts and the eval() function. However, inline scripting is a popular practice among today's websites. Many use inline scripts as part of their core functionality, and UserCSP blocks such resources. While removing all inline scripts from websites is possible, it requires significant commitments of time and effort by the developers, and may also increase load latency of webpages by introducing additional resources. Hence the UserCSP add-on faces compatibility issues with websites which rely on inline JavaScript.

   The use of proposed script-nonce and script-hash directives may present a solution for such cases.[11]

2) *Risk of sub-optimal policies that break functionality:* An inferred policy for a website may block resources important to the usability and functionality of the website. This could be due to an incorrectly inferred policy, or because the resource in question doesn't fulfil the security criteria of the policy. The architecture of a website can determine whether it is possible to create an effective policy for its content. Websites may use CDNs (Content Delivery Networks) to host content which are far too extensive to precisely enumerate and incorporate into a secure, viable policy. Websites may also allow their users to post externally hosted content, making security policies meaningless. In such cases, the implementation of UserCSP might deliver a "secure" rendering of the website, which is not usable due to broken or blocked functionality. This can cause more damage to the organization than the occasional content-based attacks.[11]

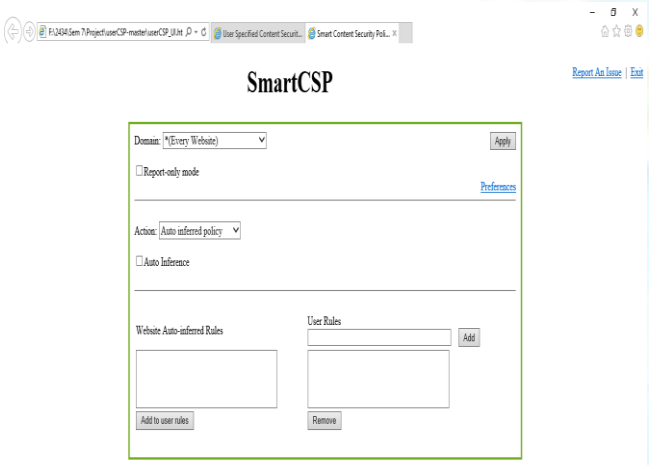
3) *Compatibility problems with Browser Extensions:* Many browser extensions modify the DOM of webpages by injecting additional third-party resources such as images, scripts, libraries (eg. Ajax tools and Google Analytics), and even resources from large CDNs, in order to fulfil functionality. These resources aren't included in the original webpage, and hence are not whitelisted in the inferred policy. While some extensions are able to modify the CSP headers to whitelist their sources, using CSP can cause most browser extensions to stop functioning. Whitelisting of browser extension resources is an option; however, the number of third party resources included by extensions is also unbounded and unpredictable. CSP in its current form is not an adequate mechanism for browser extension security. Furthermore, implementation of CSP over browser extensions gives rise to numerous complications. Due to the above reasons, not enforcing content security rules for browser extensions is the most advantageous approach.[11]

C. User Interface

SmartCSP add-on displays the following User Interface. The main window of UI of the SmartCSP is divided into three different sections:-

- The lowermost section comprises of website auto-inferred rules and user rules along with the option to add or remove these rules.

- The middle section is dedicated to the type of action that needs to be selected along with a checkbox for Auto-Inference. Action dropdown involves following options:
  - **Website defined policy**
  - User defined policy
  - Auto-inferred policy
  - Do nothing
- The topmost section has a dropdown for selecting domain, a checkbox for report only mode, an apply button and a preferences link.



Preferences link on the main will lead the user to the next window.

- In case when no policy is found, one of the following options needs to be selected:
  - Define Policy
  - Auto-inferred policy
  - Do nothing
- This dropdown is followed by two checkboxes, one for 'Apply website policy if available' and the next one for 'auto infer policies for websites'.
- In case of conflicts between user defined and website defined policy occurs, one of the following options from the dropdown needs to be selected:
  - Website Policy
  - User defined policy
  - Coupled (loose)
  - Coupled (strict)

5. CONCLUSION

Growing need for Content Security Policy inspired us to develop an add-on for Mozilla Firefox that could infer the policies itself and guide the user about enforcing the policies. This would facilitate web security for an average user who is unaware of Content Security Policy. For users and web developers who are aware with the concept of Content Security Policy, they can specify and enforce the policies as per the requirements. For users who are not aware of Content Security policy, Smart CSP provides a feature of inferring the policy itself and guiding the user as to which recommended policies should be enforced and how as per the usage pattern of the user.

REFERENCES

[1] Sid Stamm, Brandon Sterne, and Gervase Markham. Reining in the web with content security policy. In Proceedings of the 19th International Conference on World Wide Web, 2010.

[2] W3C Candidate Recommendation. Content security policy 1.0. http://www.w3.org/TR/CSP/.

[3] Ashar Javed. Csp aider: An automated recommendation of content security policy for web applications. In IEEE Oakland Web 2.0 Security and Privacy (W2SP 2012), 2012.

[4] Kailas Patil, Tanvi Vyas, and Fredrik Braun. Usercsp:: Add-ons for firefox. https://addons.mozilla.org/en-US/firefox/addon/newusercspdesign/.

[5] Kailas Patil, Tanvi Vyas, Fredrik Braun, Mark Goodwin and Zhenkai Liang Poster: UserCSP- User Specified Content Security Policies, soups'13

[6] Kailas Patil, Tanvi Vyas, and Fredrik Braun. Usercsp. github. https://github.com/patilkr/userCSP.

[7] Isaac Dawson. Security headers on the top 1000000 websites. http://www.veracode.com/blog/2012/11/security-headers-report/.

[8] ScrapyProject. Scrapy: An open source web scraping framework for python. http://scrapy.org/.

[9] W3C Editors Draft. Content security policy 1.1. https://dvcs.w3.org/hg/content-securitypolicy/raw-file/tip/cspspecification.dev.html.

[10] https://en.m.wikipedia.org/wiki/Code_injection/

[11] Michael Weissbacher, Tobias Lauinger, and William Robertson. 'Why is CSP failing? Trends and challenges in CSP adoption'.

