Poster: UserCSP- User Specified Content Security Policies

Kailas Patil
National University of Singapore
patilkr@comp.nus.edu.sg

Tanvi Vyas
Mozilla Corporation
tanvi@mozilla.com

Frederik Braun
Mozilla Corporation
fbraun@mozilla.com

Mark Goodwin
Mozilla Corporation
mgoodwin@mozilla.com

Zhenkai Liang
National University of Singapore
liangzk@comp.nus.edu.sg

ABSTRACT

Content Security Policy (CSP) is a browser security mechanism that aims to protect websites from content injection attacks. To adopt CSP, website developers need to manually compile a list of allowed content sources. Nearly all websites require modifications to comply with CSP's default behavior, which blocks inline scripts and the use of the *eval()* function. Alternatively, websites could adopt a policy that allows the use of this unsafe functionality, but this opens up potential attack vectors. When websites do not implement CSP, security savvy users do not have the control to proactively protect themselves. To make adoption of CSP easier, we propose UserCSP, a Firefox extension that uses dynamic analysis to automatically infer CSP policies, facilitates testing, and gives savvy users the authority to enforce client-side policies on websites.

1. INTRODUCTION

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. Content Security Policy (CSP) [1] 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. Web developers at large technology companies may not have direct access to change the CSP header on web servers, making it difficult to iterate over policies.

Developers are primarily focused on user experience and providing rich functionality to end users. The reluctance of developers to adopt CSP, as seen in our experiments, shows that they are unwilling to sacrifice functionality for security because they are worried about losing customers. However, security savvy users may prefer security over rich functionality. Since browsers do not currently expose a policy enforcement mechanism directly to users, users lack control over their own security when websites do not implement CSP. If developers and users do not experiment with CSP, it is difficult for the community to iterate on the CSP specification [2] to come to a more usable solution.

To assist website administrators in constructing Content Security Policies, CSP AiDer [3] uses crawler to crawl all the pages associated with a website and recommends a CSP policy based on the types of content found and the sources of that content. However, CSP AiDer is not open-source and hence not available to most web developers. UserCSP, on the other hand, is an open-source project available for download on the Mozilla Add-on gallery [4] as well as on GitHub [5].

The goals of UserCSP are two-fold: i) to allow security savvy users to specify their own CSP policies, and ii) to allow developers to experiment with CSP policies on their production pages. Moreover, UserCSP assists users and developers in constructing comprehensive CSP policies by providing them automatically inferred Content Security Policies that they can use as a starting point for experimenting with CSP on a website.

In summary, this paper makes the following contributions:

- We design and prototype UserCSP to automatically generate Content Security Policies and then we evaluate compatibility of the inferred security policies on websites.
- We propose an approach for applying security policies on the client-side. Our approach allows savvy users to specify their own custom Content Security Policies.

Our experiments show a lack of Content Security Policy implementations and the necessity for a tool like UserCSP to help promote adoption. UserCSP provides developers with an easy mechanism to create an effective, comprehensive, and strict Content Security Policy that secures their users and does not break website functionality.

2. UserCSP DESIGN

UserCSP helps developers and users write comprehensive policies for websites by providing them with a GUI to add and modify CSP policies. UserCSP monitors the browser's 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.

When users visit a website, UserCSP performs one of the following actions:

- 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 website's 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.

3. ANALYSIS & RESULTS

We tested UserCSP's user defined CSP feature and automatically infer CSP feature with the Alexa Top 100 websites[1]. Manually defined CSP policies are harder to evaluate since they require several rounds of refinement and HTML source code inspection to record content sources. We initially seeded the policies with same-origin restrictions and then expanded them since many websites require content from CDN's and sub-domains.

To test compatibility of the automatically infer CSP feature of UserCSP, the extension inferred policies for each of the Alexa Top 100 websites and then applied the policies onto their respective website home pages (Figure 1 includes an example of automatically inferred policy). Reports were created for each website and examined for CSP violations[2].

```
default-src      'self';
script-src        http://ads1.msads.net
                  http://kaw.stj.s-msn.com;
img-src           http://udc.msn.com
                  http://kaw.stb.s-msn.com
                  http://b.scorecardresearch.com
                  http://c.in.msn.com
                  http://www.bing.com
                  http://kaw.stb01.s-msn.com
                  http://kaw.stc.s-msn.com
                  http://kaw.stb00.s-msn.com;
style-src         http://kaw.stc.s-msn.com;
frame-ancestors   *;
```

Figure 1: Inferred CSP for msn.com

The number of whitelisted origins per-policy ranged from 1 to 33, with a mean of just over 7 origins per-policy and a standard deviation of 6.52. Over 25% of websites required more than 10 origins, indicating that creating a comprehensive and effective CSP policy is a challenging task. When there are more than a handful of resources to whitelist, developers are likely to whitelist everything by including "*" in a directive instead of searching for all the necessary origins; this makes the policy less restrictive than it could be. By providing a mechanism to infer the policy, UserCSP provides a quick, effective, and comprehensive policy for developers to set on their websites.

[1]Three websites containing adult content were excluded from our testing

[2]In order to adhere to the same-origin-only report-uri restriction in Firefox without alerting websites with our custom CSP testing, we used *http-on-modify-request* to capture and then cancel HTTP requests that contained violation reports.

Our tests show that none of the Alexa Top 100 websites have implemented CSP. A recent study revealed that only 79 out of the Alexa Top 1,000,000 websites implement CSP, showing that CSP has a very low adoption rate [6].

After applying UserCSP's inferred policies, all the Alexa Top 100 websites generated CSP violation reports that showed violations for the inline script default restriction. In addition, 11 websites generated CSP violation reports for using *eval()*[3]. This experimental survey implies that websites commonly use inline scripts.

To further test this theory, we scanned the Alexa Top 25,000 websites using the *Scrapy* framework [7]. Of the 23,195 accessible websites, 22,324 (96.2%) were using inline scripts or inline event handlers.

4. CONCLUSION

Content Security Policy has not been widely adopted because of the challenges involved in creating a comprehensive and functional policy. Since adoption is controlled by developers, users lack control over their own security. Users do not have a mechanism to apply Content Security Policies on the websites that they visit and cannot protect themselves from Cross-Site Scripting and Clickjacking attacks.

UserCSP helps break down the challenges involved in adopting Content Security Policy with its feature to automatically infer policies. It also puts control into the users hands by providing them a mechanism to protect themselves with custom policies that they can create and modify. Our analysis and results show that another barrier to Content Security Policy adoption is the use of inline JavaScript. To overcome this, we would like to experiment further with the proposed *script-nonce* and *script-hash* directives that are under discussion for inclusion in the CSP 1.1 specification [8].

Acknowledgments

We thank Sid Stamm for his help on the implementation of UserCSP. We thank the anonymous reviewers for their valuable comments.

5. 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, and Fredrik Braun. Usercsp. github. https://github.com/patilkr/userCSP.

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

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

[8] W3C Editor's Draft. Content security policy 1.1. `https://dvcs.w3.org/hg/content-security-policy/raw-file/tip/csp-specification.dev.html`.

[3]Websites that generated CSP violation reports for the use of *eval()*: `www.youtube.com`, `www.qq.com`, `bbc.co.uk`, `adobe.com`, `sohu.com`, `aol.com`, `youku.com`, `cnn.com`, `dailymotion.com`, `imgur.com`, `neobux.com`