UserCSP:User Specified Content Security Policy

Kailas Patil, Tanvi Vyas
patilkr24@gmail.com, and tanvi@mozilla.com

[1] National University of Singapore
[2] Mozilla

Abstract. According to OWASP top vulnerability list, cross-site scripting (XSS) is among the top five web application vulnerabilities. It allows attackers to inject malicious code or resources from attacker domains into the DOM of a vulnerable web page. Browsers are not able to distinguish between legitimate and malicious content. Therefore, Content Security Policy (CSP) is a mechanism that enables the browser to identify potentially malicious injected content in web pages.
By-default CSP doesn't allow inline scripts and eval, which are used by almost all website. To use CSP, websites are required to change their code or allow these potential attack vectors, and hence mitigate the effectiveness of CSP. The code change requirement is hindering the adaptation of CSP by web applications (websites). However, there are savvy users who prefer security over usability. In addition, website developers need a tool to test different CSP rules for their website that secure their users and also achieve usability. To address these issues we propose UserCSP and prototype our approach in a Firefox extension using the JetPack framework. UserCSP allows savvy users to specify CSP rules to particular websites or to specify general CSP rules that are enforced on each and every website a user visits. Moreover, it allows website developers to try out different CSP rules and iterate to achieve the best suited CSP policy for their website.

Keywords: content security policy, web security, security policy, content restrictions

1 Introduction

The web browsers security model is rooted in the same-origin policy (SOP) [6], which isolates resources of one origin from others. However, attackers can subvert the SOP by injecting malicious contents into a vulnerable website; this is known as Cross-site scripting (XSS) attacks [1]. Cross-site Scripting (XSS) attacks in web applications are considered a major threat. XSS allows an attacker to conduct a wide range of potential attacks, such as session hijacking, stealing sensitive data and passwords, creation of self-propagating JavaScript worms, etc. To mitigate XSS attacks, Content Security Policy (CSP) [8] provide website administrators with a way to enforce content restrictions at the client side.

Content Security Policy is a declarative policy that restricts what content can be loaded on a web page. Its primary purpose is to mitigate Cross-Site Scripting vulnerabilities. The core issue exploited by Cross-Site Scripting (XSS) attacks is that web browsers lack the knowledge to distinguish between content that's intended to be part of a web application, and content that's been maliciously injected into a web application. To address this problem, CSP defines the `Content-Security-Policy` HTTP header that allows web application developers to create a whitelist of trusted content sources, and instruct the client browsers to only execute or render resources from those sources. However, it is often difficult for developers to write a comprehensive Content Security Policy for their website. They may worry about their page breaking because unanticipated but necessary content is blocked. They may not be able to easily change the headers their site is sending when these situations occur, which makes it difficult for them to try different policies until they find one that is the most restrictive for their page without breaking site functionality.

UserCSP changes this! A developer or user can now view the current policy set by their site and add their own policy. They can choose to apply their custom policy on the site, or even combine their policy with the websites existing policy. When combining policies, they have an option to choose from the strictest subset of the two, or the most lax subset. They can locally test their site with the custom policy applied and tweak the policy until they have one that works.

UserCSP also allows automatic inference of a Content Security Policy for a website. Automatically inferred policies for a website help web developers figure out what CSP rules to set for their site by giving them the strictest possible policy they could apply without breaking the current page. To infer the CSP policy for a website, UserCSP analyzes the content on the current web page and recommends a CSP based on the content types and content sources. UserCSP

provides this inferred policy to developers in the proper syntax for the CSP header, so all a developer needs to do is start serving this policy for their site via the CSP header. Furthermore, UserCSP allows savvy users to voluntarily specify their own Content Security Policy for websites that may not have implemented CSP.

In summary, this paper makes the following contributions:

- We present an automated approach, UserCSP, for writing a Content Security Policy for a website.
- UserCSP allows savvy users to voluntarily specify their own Content Security Policy for websites that may not have implemented CSP.
- We implemented a prototype of our approach in a Firefox extension.

2 Background

Content Security Policy (CSP) helps to detect and mitigate attacks such as Cross-site scripting(XSS) and Clickjacking.

2.1 Cross-site Scripting (XSS)

Cross-site Scripting (XSS) or script injection is well known vulnerability in web applications. According to Web Hacking Incident Database (WHID) 2010 semiannual report, XSS was in the top in the list of application security risks [9]. In XSS attack malicious script is injected into an otherwise trusted web page violating integrity of infected web application and causing an unexpected behavior such as stealing sensitive information or modifying server side state of a user. When a user visits a web page of trusted site that contains malicious injected script, the injected script is executed in user's browser with the principle of trusted site. Therefore, injected scripts have full control over the session and thus can send arbitrary requests with valid session and security tokens.

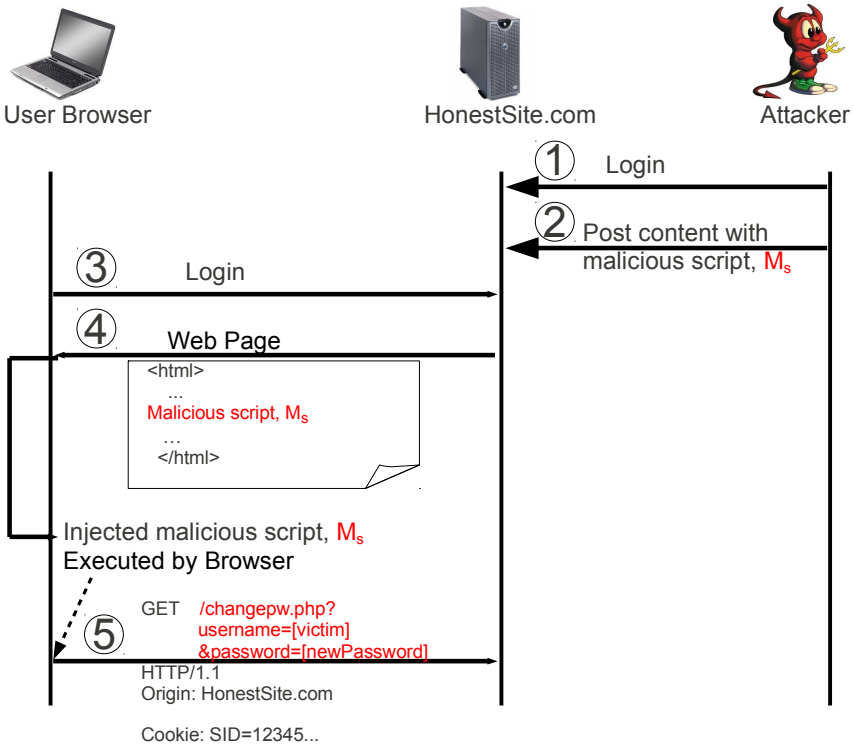

Fig. 1. Illustration of Cross-Site Scripting

An Example of XSS Attack. If user visits a web page of vulnerable trusted web site, say *HonestSite.com* in which attacker has injected inline malicious script, say M_s. Malicious script, M_s, is executed in the context of web site and can send state modifying request to *HonestSite.com*

As shown in Figure 1, attacker first login to a web site, *HonestSite.com* (Step 1). On the vulnerable web page of the web site, attacker submits malicious script, M_s (Step 2). When user login to the web site (Step 3), and visits the vulnerable web page (Step 4), injected malicious script, M_s is executed in the context of web site. When injected script executed in the context of web site it will compromise the integrity of web site by sending malicious request to modify user state of the user on the web site (Step 5). Request is generated by injected script running with the principle of web site, therefore web site processes the malicious request generated by injected script. Moreover, injected script when executed with the principle of web site it can also steal sensitive information such as username/password, cookies, etc and send them to attacker's domain.

Mitigation of Cross-site Scripting(XSS). Content security policy allows website administrators to eliminate their XSS attack surface. To achieve this goal, CSP allows website administrators specify which domains the browser should treat as valid sources of script. Moreover, the web browser will only execute script in source files from the white-listed domains and will disregard everything else, including inline scripts.

2.2 Clickjacking

In Clickjacking attacks [2, 7], attackers exploit the layout feature introduced by iFrames. Specifically, they load a victim web page into an iFrame on the top and make it transparent. Then they load a deceptive page in another iFrame at the bottom layer to attract users to click.

```
<-- Page from www.Websitename.com -->
<html>
...
<iframe id="victim" src="http://example.com" scrolling="no"
     width="600px" height="600px" style="opacity:0;
     position:absolute; left:10px; top:10px;">
</iframe>
...
<div style = "position:absolute; top:Ypx;left:Xpx;">
     <a href= "http://example.com">Click Here</a>
</div>
...
</html>
```

Fig. 2. Clickjacking using transparent iFrame and overlay objects.

An Example of Clickjacking. Figure 2 shows an example of a Clickjacking attack. The front page of `http://example.com` is loaded inside a transparent iFrame (zero opacity value to make it transparent). To lure users to click at a particular location of the page loaded inside the transparent iFrame, the attacker creates a link in the visible bottom layer, which is located exactly at the same position where the attacker wants users to click in the top layer. As shown in Figure 2, the attacker specifies the location of a link by setting its X and Y coordinates. When users try to click on the link, they actually click on the transparent layer of the iFrame loaded with the page from `example.com`. An illustration of such a Clickjacking attack is presented in Figure 3.

Mitigation of Clickjacking. Content Security Policy enables a protected website `S` to specify which websites can embed `S`. That is, a protected website `S` can decide what other websites it trusts to embed it.

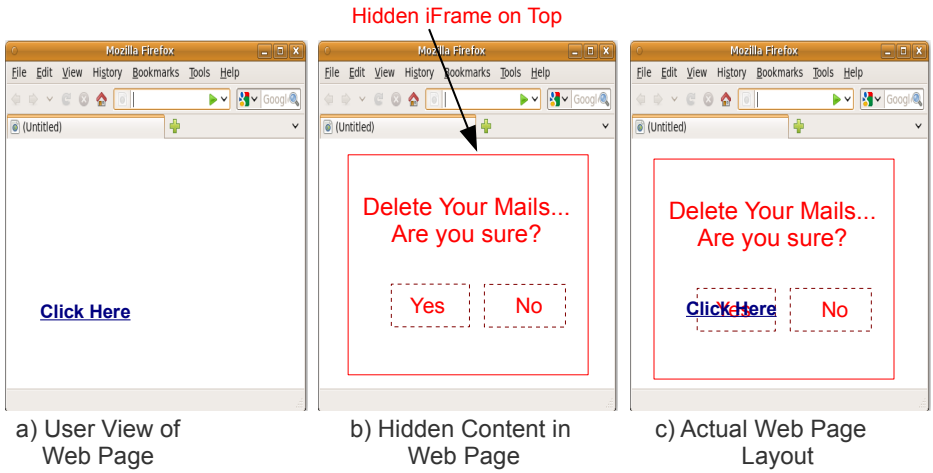

Fig. 3. Illustration of Clickjacking using transparent iFrame and overlay objects

3 UserCSP Design

The goal of our approach, UserCSP, is to help web site administrators and website users to write comprehensive CSP rules for the website.

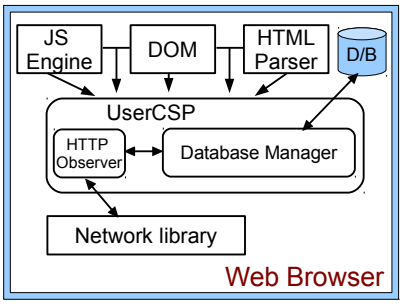

Fig. 4. UserCSP Architecture

Figure 4 illustrates the architecture of UserCSP. To enforce user specified Content Security Policy as well as to infer CSP policy for a website, UserCSP monitors web browsers internal events including HTML parsing, HTTP requests, XHR, etc., and analyzes the type of content loaded by a web page and source of that content. Database manager component of the UserCSP is responsible for storing user specified CSP rules for websites into local database and retrieves the corresponding CSP rules for the website when user loads the website.

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

- If website has defined CSP, but the user hasnt specified a CSP policy for that website, then UserCSP doesnt interfere with the website defined CSP rules. However, it allows the user the option to amend the website's CSP.
- If a user has specified CSP rules for a website, but the website administrators hasnt defined a CSP for their website, then the user specified CSP policy will be enforced by the UserCSP.
- If a user specified CSP exists as well as a website defined CSP, then users have a choice either to apply their own policy or adopt the website defined policy. Moreover, users can also select a strict or loose set of CSP rules by combining their own policy with the website defined policy. For example, if a website sets script-src www.example.com; and user specifies script-src www.example.com www.abc.com; then the strict policy would be script-src www.example.com; whereas the loose policy would be script-src www.example.com www.abc.com;.

– If neither user nor website administrators specifiy a CSP for a website then UserCSP doesnt interfere with the content loading on the website.
– To allow automatic inference of a CSP for websites, UserCSP monitors content loaded by web pages and recommends CSP rules based on the types of content and sources of that content. It also monitors the resources dynamically added to the web page by JavaScript.

4 Implementation of UserCSP

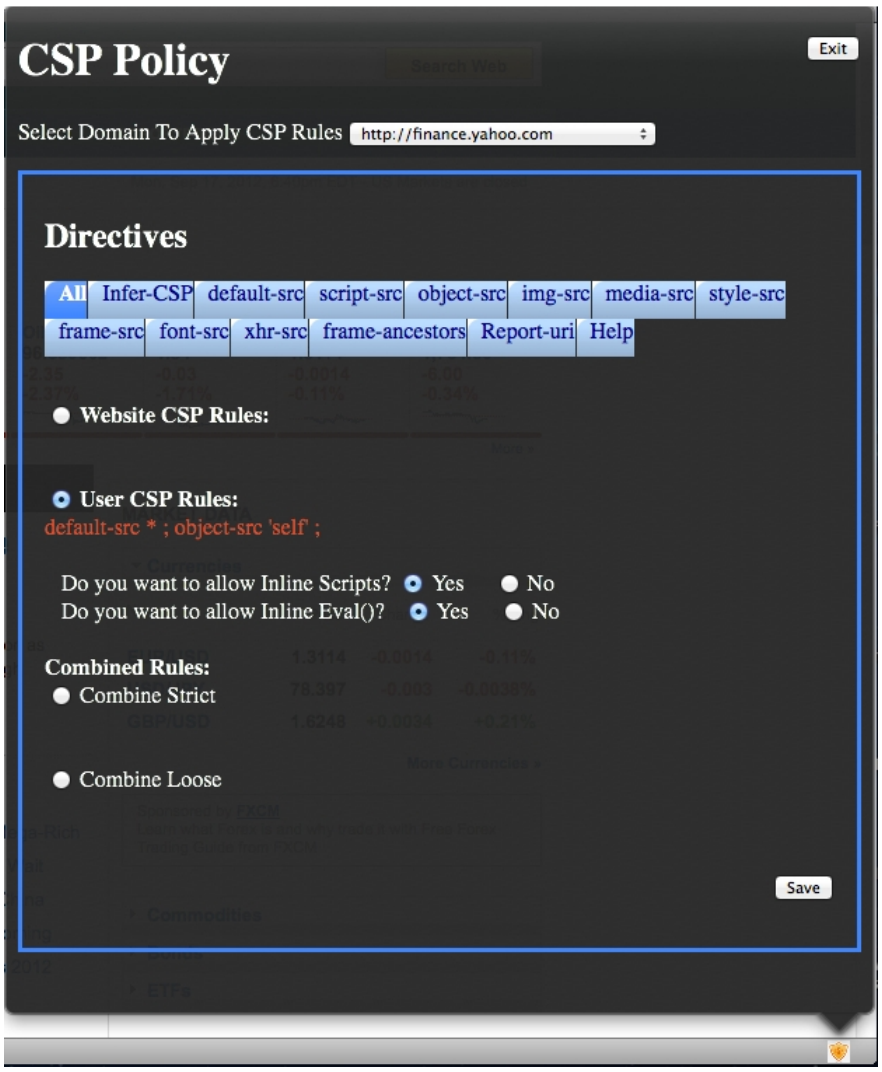

Fig. 5. UserCSP UI

We implemented UserCSP [1] in Firefox extension using the Jetpack SDK v1.9 [5]. UserCSP intercepted various events, including the *READY*, *ACTIVATE*, and *CLOSE* events on tab. The READY event is used to retrieve a list of open websites in a user's Firefox web browser. The ACTIVATE event is used to select the currently active domain in the web browser. The CLOSE event is used to remove the domain name from the UI if a user closes the tab. UserCSP uses a sqlite database to store user specified CSP rules for websites.

[1] UserCSP Source Code: https://github.com/patilkr/userCSP

The *http-on-examine-response* observer notification is used to intercept the HTTP response. In the intercepted response, the domain that initiated the request is checked against the sqlite local database to determine whether user defined rules were set. If there are no rules associated with the website, the response is processed without any change. However, if user defined CSP rules exist, the `Content- Security-Policy` header is added to the response with the rules specified by the user. If user has opted to enforce their own rules then UserCSP replaces the existing `Content-Security-Policy` header if it is already set by the website.

Figure 5 shows graphical user interface of UserCSP add-on we developed as an extension to Mozilla Firefox. The userCSP add-on UI contains drop-down list for domain selection. The domain selection list contains the websites that the user has opened in the browser. For example, `http://finance.yahoo.com/` is shown in the Figure. In addition to this, domain selection drop-down list also contains an entry `* (Every Website)`. The `* (Every Website)` option is used to allow users to specify general rules for all websites the users visits that do not have a website or user CSP policy set. If a user has set a policy for website and also set a policy for `* (Every Website)`, then the user policy set for the website takes precedence over the `* (Every Website)`. If a website has set a CSP policy in their header and the user has set a policy for `* (Every Website)`, then the policy set by the website takes precedence over the `* (Every Website)`.

Moreover, for each domain there are total 14 tabs provided to user as shown in the Figure 5 namely, `All`, `Infer-CSP`, `default-src`, `script-src`, `object-src`, `image-src`, `media-src`, `style-src`, `frame-src`, `font-src`, `xhr-src`, `frame-ancestors`, `report-uri`, `Help`. Except for the 'All', 'Infer Policy', and 'Help' tab, the other tabs are CSP directives used in Firefox. They are used to allow a user to specify a CSP rule for that CSP directive. The 'All' tab is used to display the complete website defined CSP policy, as well as complete user defined CSP policy. It also allows users to calculate the Strictest Policy and Loosest Policy from the user defined CSP and the website defined CSP. Moreover, it also allows user to select a policy for a website from the four possible values - Website CSP rules, User CSP rules, Combine Strict Rules or Combine Loose Rules. By-default website CSP rules are selected. In addition to this, when the User CSP rules are selected, the 'All' tab also allows users to enable or disable inline scripts and inline evals.

Combine Strict CSP: If both website and user defined CSP rules for a website are available then this feature allow users to apply the strictest subset CSP policy which is calculated from the website defined CSP and the user defined CSP. For example, when you strictly combine website specified policy `img-src 'self'` with user specified policy `img-src '*'`, the combined policy `img-src 'self'` is set.

Combine Loose CSP: If both website and user defined CSP rules for a website are available then this feature allow users to apply the loosest subset CSP policy which is calculated from the website defined CSP and the user defined CSP. For example, when you loosely combine website specified policy `img-src 'self'` with user specified policy `img-src "*"`, the combined policy `img-src '*'` is set.

5 Evaluation

We implemented our approach in Firefox v14.0 using JetPack SDK v1.9. We used Alexa Top 10 Sites [2] to test our approach against user defined CSP policies as well as automatically inferred CSP policies. Automatically inferred CSP policies don't break websites whereas manually defined CSP policies required several rounds of refinement and web page source code inspection to record content sources. The reason for that is initially we set CSP rules for a website to load resources from its own domain only, but websites were loading content from CDN's or sub-domains. Appendix 8 shows examples of inferred CSP policies.

5.1 Effectiveness of Automatically Infer CSP

We tested effectiveness of UserCSP with Alexa TOP 10 Sites listed in Appendix 8.

Automatically infer CSP policy feature of UserCSP helps a developer to derive a usable and secure policy for their web site. By looking at the content the website loads, the add-on determines the strictest set of CSP rules it can apply to the site without breaking the current page. Web users can also protect themselves by disabling content such as

[2] Alexa top 10 sites used for evaluation were: facebook.com, google.co.in, youtube.com, yahoo.com, baidu.com, wikipedia.org, live.com, twitter.com, qq.com, and amazon.com

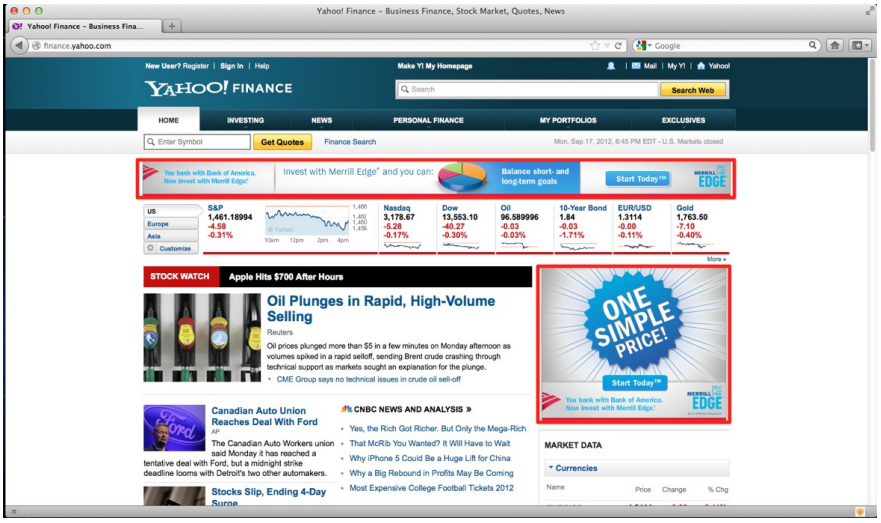

Fig. 6. Before any Content Security Policy is applied to *http://finance.yahoo.com/* web site

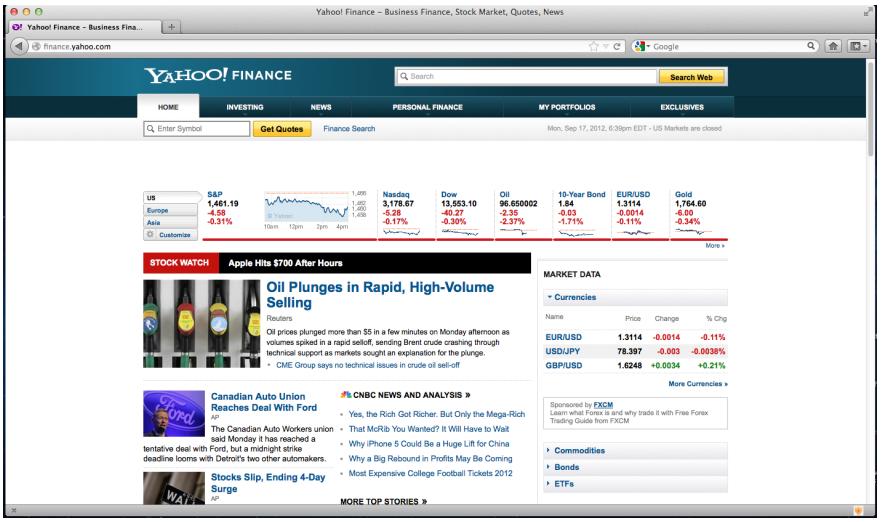

Fig. 7. After `object-src:'self'` Content Security Policy is applied to *http://finance.yahoo.com/* web site

objects on personal finance sites or frames and third party JavaScript for their web-email. Figure 6 shows an example of `http://finance.yahoo.com/` web site, which used third-party flash objects. In the Figure, third-party flash objects are marked with red rectangle. Figure 7 shows an example, where user specified Content Security Policy that sets `object-src` directive to `'self'` so that third-party plugins won't load. All third-party flash object used for an advertisement in `http://finance.yahoo.com/` were blocked by our UserCSP add-on after the user applied a custom policy.

6 Related Work

Content Security Policy(CSP) [8] provides content restriction enforcement that allows web sites to specify which sites are trusted and then rely on the users browser to forbid loading resources from untrusted sites or from the sites not in the list specified by the web application. We extended CSP mechanism to allow users to specify CSP policy for a web site, and then enforce it.

BEAP [4] mitigates the CSRF attack at client side. It infers whether a request reflects user's intention using the heuristics derived from analyzing real-world web applications. If a sensitive request does not reflects users' intention, BEAP strips authentication token from it. Whereas CSP blocks all requests except for those that the web site has explicitly granted.

Browser-enforce Embedded Policies (BEEP) [3] aims to allow web applications to specify which scripts can run on its web site. As compared to our approach, BEEP is limited to restricting scripts that can run on a web site, whereas other contents such as images, frames, style sheets, etc are not restricted.

7 Conclusion

In complex web application attackers find various ways to inject malicious content in a website. Content Security Policy (CSP) provides a content restriction mechanism to allow website administrators to client side control of the types of content that can be loaded and the sources they can be loaded from. It also provides the ability for web browsers to distinguish web site intended contents and injected content. In this work, we present an approach, UserCSP, that helps web administrators and users write compressive CSP rules for websites. It also automatically infers CSP rules for websites. Our prototype was implemented in the Firefox extension using the JetPack SDK.

References

1. cgisecurity. The cross-site scripting (xss) faq. http://www.cgisecurity.com/xss-faq.html.
2. Robert Hansen and Jeremiah Grossman. Clickjacking. http://www.sectheory.com/clickjacking.htm, 2008.
3. Trevor Jim. Defeating script injection attacks with browser-enforced embedded policies. In *In World Wide Web*, 2007.
4. Ziqing Mao, Ninghui Li, and Ian Molloy. Defeating cross-site request forgery attacks with browser-enforced authenticity protection. In *Financial Cryptography and Data Security, 13th International Conference*, 2009.
5. Mozilla. Add-on sdk documentation. https://addons.mozilla.org/en-US/developers/docs/sdk/latest/.
6. Mozilla. Same origin policy for javascript. https://developer.mozilla.org/En/ Same_origin_policy_for_JavaScript.
7. Nex. The clickjacking meets xss: a state of art. http://www.milw0rm.com/papers/265, 2008.
8. 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.
9. WHID. Web hacking incident database 2010 semi annual report. http://projects.webappsec.org/w/page/29620941/Web-Hacking-Incident-Database-2010-Semi-Annual-Report.

8 Appendix: Examples of inferred CSP policy by UserCSP

CSP policies that were automatically inferred by UserCSP for `www.facebook.com` and `in.yahoo.com` are given below:

Inferred CSP policy for www.facebook.com Figure 8 shows an inferred CSP policy by UserCSP for `www.facebook.com`.

```
default-src 'self';
script-src http://static.ak.fbcdn.net;
img-src http://photos-g.ak.fbcdn.net
        http://photos-c.ak.fbcdn.net
        http://photos-a.ak.fbcdn.net
        http://photos-b.ak.fbcdn.net
        http://secure-us.imrworldwide.com
        http://static.ak.fbcdn.net
        http://profile.ak.fbcdn.net;
style-src  http://static.ak.fbcdn.net;
```

Fig. 8. Inferred CSP for www.facebook.com

Inferred CSP for in.yahoo.com/?p=us Figure 9 shows an inferred CSP policy by UserCSP for `in.yahoo.com`.

```
default-src 'self';
script-src http://bs.serving-sys.com
           http://l.yimg.com
           http://mi.adinterax.com;
object-src http://mi.adinterax.com ;
img-src http://l.yimg.com
        http://l1.yimg.com
        http://d.yimg.com
        http://ads.yimg.com
        http://ad.yieldmanager.com
        http://ds.serving-sys.com
        http://b.scorecardresearch.com
        http://row.bc.yahoo.com;
style-src http://l.yimg.com;
frame-src http://ads.yimg.com
          http://ad.yieldmanager.com;
```

Fig. 9. Inferred CSP for in.yahoo.com