CSP AiDer: An Automated Recommendation of Content Security Policy for Web Applications

Ashar Javed
PhD Student
Hamburg University of Technology (TUHH)
Hamburg, Germany
justashar@gmail.com

Abstract—Unintended cross-domain content flows are a major security weakness of current web design. Cross Site Scripting (XSS) and Cross Site Request Forgery (CSRF) are widely recognized symptoms of this problem. In the span of just a few years, dozens of high-profile attacks against websites using Cross Site Scripting and Cross Site Request Forgery for the purposes of compromising private data, bring down essential systems, malware planting, clickjacking or otherwise wreak havoc on our lives. Content Security Policy (CSP) is a Mozilla initiative to provide website administrators with a way to specify how content interacts on their web sites—a security mechanism pressingly needed by the uncontrolled Web. The main goal of Content Security Policy is to prevent malicious code from being injected into a website and executed within the context of that site.

In this work, we present the first automated approach for the construction of content security policies in web applications. To the best of our knowledge, no tools have been developed using the standard technology offered by the Mozilla development environment: a mix of Javascript and loading additional resources, a browser extension (i.e., plugin) extracts the content. The browser extension has finished parsing the DOM, executing the clientside scripts, waits until the target page is loaded. After the browser has been presented to date for the recommendation of CSPs of more than 10000 web sites, We informed a number of major web sites about the CSPs we identified, and our findings were confirmed by mainstream web sites such as Twitter.

I. INTRODUCTION

Brandon Sterne et al. proposed the Content Security Policy (CSP). Content Security Policy is a Mozilla initiative to provide website administrators with a way to specify how content interacts on their web sites—a security mechanism desperately needed by the uncontrolled Web. The first component is an instrumented browser that is responsible for fetching the webpages and rendering the content. The instrumented browser in CSP AiDer first waits until the target page is loaded. After the browser has finished parsing the DOM, executing the clientside scripts, and loading additional resources, a browser extension (i.e., plugin) extracts the content. The browser extension has been developed using the standard technology offered by the Mozilla development environment: a mix of Javascript and XML User Interface Language (XUL). The XUL is flexible and extensible.

The second component is a crawler that communicates with the browser through a bidirectional channel. This

Twitter has implemented CSP on its mobile website which is aimed at thwarting cross-site scripting (XSS) attacks.

The second component is a crawler that communicates with the browser through a bidirectional channel. This
channel is used by the crawler to inform the browser on the URLs that need to be visited. Furthermore, the channel is also used to retrieve the collected information from the browser. We used the Heritrix public domain Web crawler to gather a crawl of over 10000 Internet Web sites.

Every time the crawler visits a page, it passes the extracted information to the scanner so that it can be analyzed. Similar to other scanners, it would have been possible to directly retrieve web pages without rendering them in a real browser. However, such techniques have the drawback that they cannot efficiently deal with dynamic content that is often found on Web pages (e.g., Javascript). By using a real browser to render the pages we visit, we are able to analyze the page as it is supposed to appear after the dynamic content has been generated. The ability to deal with dynamic content is a necessary prerequisite to be able to construct content security policy for web mashups. The scanner is responsible for analyzing the page to determine the types of content it finds on the page and the sources of that content. The scanner also takes into account resources that are dynamically added to the page by JavaScript.

The last component in our CSP AiDer system is generator. CSP generator recommends CSP based on the information that are coming from scanner. In other words generator turns a list of sources into proper CSP syntax. All the collected information about CSP is stored in a database that is later analyzed by a statistical component that groups together information and generates a report. The general architecture of the system is summarized in figure 1.

![Diagram of CSP AiDer Tool](image)

**Figure 1.** The architecture of the CSP AiDer Tool.

**A. Implementation**

Our implementation is the JavaScript library used by both the 'scanner' and 'generator' component. Our JavaScript library has 455 lines (including comments).

**B. Example Policies Constructed by CSP Aider**

**Constructed CSP of Technorati:**

X-Content-Security-Policy: default-src 'self';
img-src 'self' scm-l3.technorati.com i.ytimg.com content.yieldmanager.com content.yieldmanager.edgesuite.net aidps.atdmt.com tmstats.technoratimedia.com t.skimresources.com;
object-src content.yieldmanager.edgesuite.net;
frame-src ib.adnxs.com;
sty-le-src scm-l3.technorati.com;

**Constructed CSP of Mobile.Twitter.com**:

X-Content-Security-Policy: default-src 'self';
img-src si0.twimg.com;
script-src ;
sty-le-src si0.twimg.com;

**III. EVALUATION**

To do this, we conduct a large-scale outward-looking study by crawling the Web, downloading content from a large number of sites, and then analyzing it to determine CSP. In an experiment, we collected 7,000 unique URLs from the public database of Alexa. In particular, we extracted the top ranked sites from each of the Alexa's categories. Each website was considered only once even if it was present in multiple distinct categories, or with different top-level domain names such as www.google.com and www.google.de. In addition, we crawled award winning mashups from http://mashupawards.com/winners/site and mashups directory available at http://www.programmableweb.com/mashups/directory.

**IV. CONCLUSION**

Web applications are not what they used to be ten years ago. Popular web applications have now become more dynamic, interactive, complex, and often compose content from multiple web sites. Unfortunately, as the popularity of a technology increases, it also becomes a target for criminals. As a result, most attacks today are launched against web applications. CSP provides not only an ability for web sites to specify what types of content may be loaded (and from where), but also some protection from cross-site scripting and cross-site request forgery by preventing inappropriate or unauthorized cross-domain communication.

In this work, we present the first automated approach for the construction of CSPs in web applications. Our prototype implementation called CSP AiDer is able to crawl websites and recommend CSP. In order to determine the feasibility of our approach, we analyzed more than 10000 popular websites and have contributed in the recommendation of their CSPs. We informed the sites for which we could obtain contact information, and some of these sites wrote back to us and confirmed our findings. We hope that this work will help in raise awareness about CSP.