Third-party JavaScript Implementation and Security on the Internet

J. López[1], T. Wang[2]
University of Puerto Rico, Mayagüez, Mayagüez, PR 00682[1]
New Mexico State University, Las Cruces, NM 88003[2]

Keywords: JavaScript, Crawler, libraries, Python, security

In the web development process, it has become common process to include client-side, third-party JavaScript libraries to facilitate their websites' development. These libraries include uses such as advertisements, location services, social media integration, and support functionality. Examples of common JavaScript libraries used by developers include Google Analytics and jQuery. However, there are some third-party libraries, even outdated versions of reliable libraries, that contain potentially dangerous scripts. When a developer includes a third-party library to their website, the security of the website will heavily rely on the library's server. These privileges can cause the user to experience web attacks, of which the most common one is cross-site scripting (XSS). It is essential to investigate the security issues regarding third-party JavaScript libraries, and understanding how the libraries are included and deployed will help developers build a better security policy and better websites for their users, especially in a time where society uses the Internet for their daily lives.

The intention of this project is to extract the JavaScript files from a list of websites via a web crawler and look for potential vulnerabilities in those files. These files contain important metrics and information, such as response headers and Cache-control, that can demonstrate how secure are the websites. Important security headers include Strict-Transport-Security, a header that enforces the use of an encrypted HTTPS connection instead of the regular HTTP. Also, there is Content-Security-Policy, a header that bring protection against XSS attacks by only allowing assets either from the local origin of the website or a source that a developer chooses to implement. Cache-control determines the amount of time it takes for a copy of a resource to expire as well as who can cache the resource.

The web crawler was developed using the Python programming and some of its libraries. The crawler includes the BeautifulSoup library for extracting the HTML and XML files, the requests library used to extract HTTP requests, the Selenium library used for automated testing of web applications, the tqdm library used as a terminal time progress bar, and the Tk library used for Graphical User Interfaces (GUI). The crawler also uses the Chrome browser for the automated testing of the websites. The crawler accepts from a user a .txt file containing the list of websites that the user wants to be crawled. The crawler will then extract the JavaScript files and its data and store it in a .csv file. The .csv file will contain the following data: the websites crawled by the program, the amount of JavaScript files per website, the files of the websites, the full headers of each file, the Cache-control, Last-Modified, and Expires of each file header. The crawler will also generate another .csv file containing the files that were repeatedly used among the list of crawled websites.

For this research project, the web crawler obtained the data of the Alexa Top 500 US websites list. Then, the data was analyzed to find potentially vulnerable sites and libraries on the list. From the list of websites, the web crawler obtained data from 98% of the sites and obtained a total of 12,508 JavaScript files. Among the sites that did not contain data, 8 sites had 0 files and 2 had expired domains. And among the 8 sites that had 0 files, the reasons for having no files were either that the sites didn't have any files to begin with or that the files were not loaded by the crawler because of the request timeout of the respective sites.

From the list of websites used, the overall usage of security headers in the analyzed files is low, with the highest being Strict-Transport-Security at 22.07%, followed by X-Content-Type-Options: 'nosniff' at 21.33%. The lowest was the default and not recommended option of X-Frame-Options, ALLOWALL, at 0.11%. On the Cache-control side, 10,149 files, or 81.1% of the total files found, contained Cache-control. Most of the files use the public metric with 38.56% of them using it. But, if the file works with sensitive information, it is recommended that it includes the private metric, which 16.82% of the files have it. For JavaScript files, it is recommended to have the max-age set from 6 months up to one year and the list of 500 websites show that the average max-age for the files surpasses that estimate.

The results obtained by the web crawler developed for this research project with the Top 500 US website list show that a few libraries and websites implemented the recommended security headers. And, in a time where our information is more vulnerable than ever, developers need to implement security headers, use reliable third-party JavaScript libraries, and update the libraries when required to or be able to implement the security headers on their own projects and not fully depend their security on a third-party.

References:

Yue, C., & Wang, H. (2013). A Measurement Study of Insecure JavaScript Practices on the Web. https://doi.org/10.1145/2460383.2460386

DOM Based XSS Software Attack | OWASP Foundation. (n.d.). OWASP. https://owasp.org/www-community/attacks/DOM_Based_XSS

Alassmi, S., Zavarsky, P., Lindskog, D., Ruhl, R., Alasiri, A., & Alzaidi, M. (2012). An Analysis of the Effectiveness of Black-Box Web Application Scanners in Detection of Stored XSSI Vulnerabilities.

Musch, Marius & Steffens, Marius & Roth, Sebastian & Stock, Ben & Johns, Martin. (2019). ScriptProtect: Mitigating Unsafe Third-Party JavaScript Practices. 391-402.
. 10.1145/3321705.3329841

Johns, M. (2011). Code-injection Vulnerabilities in Web Applications — Exemplified at Cross-site Scripting. It - Information Technology, 53(5), 256–260.
https://doi.org/10.1524/itit.2011.0651

Raman, P (2008). Jaspin: JavaScript Based Anomaly Detection Of Cross-Site Scripting Attacks https://doi.org/10.22215/etd/2008-08774

Yue, Chuan & Wang, Haining. (2009). Characterizing insecure JavaScript practices on the Web. WWW'09 - Proceedings of the 18th International World Wide Web Conference. 961-970. 10.1145/1526709.1526838.

Johns, Martin. (2008). On JavaScript Malware and related threats. Journal in Computer Virology. 4. 161-178. 10.1007/s11416-007-0076-7.

Jackson, B. (2019). *X-Frame-Options- How to Combat Clickjacking* https://www.keycdn.com/blog/x-frame-optionsX-Frame-Options -

Cross Side Scripting Prevention Cheat Sheet OWASP Cheat Sheet Series https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

Yue, C. & Wang, H. (2013) A Measurement Study of Insecure JavaScript practices on the Web https://www.eecis.udel.edu/~hnw/paper/tweb.pdf

Banach, Z. (2020, June 5). *HTTP security headers: An easy way to harden your web applications*. Netsparker | Web Application Security Solution. https://www.netsparker.com/blog/web-security/http-security-headers/

What is cache-control and how HTTP cache headers work | CDN guide | Imperva. (2019, December 30). Learning Center. https://www.imperva.com/learn/performance/cache-control/