View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Servicio de Difusión de la Creación Intelectual

Security and integrity checker for JavaScript dependencies

Short Paper

April 30, 2017

Abstract

Nowadays, most of web-based software includes heavy usage of external dependencies. However, the control over the security aspects of these dependencies is out of control of the developers, because dependency injectors do not check for the security or integrity. Thus, the software built on top of insecure dependencies become vulnerable too. We propose a novel solution by using the PumaScript meta-programming framework and browser capabilities to generate a safe JavaScript dependency injector that help in generation of software solutions less vulnerable.

1 Introduction

Internet technologies have grown fast in the last ten years, with JavaScript becoming one of the hottest hypes in programming languages [1]. The growing number of frameworks, applications and tools, and particularly the emergence of NodeJS [2], a server-side JavaScript development environment based on Google's V8 Javascript runtime, has created a thriving JavaScript ecosystem. This has made JavaScript the first option for several developers to start writing their backend solutions.

Nowadays, most of new web based solutions are created by integrating libraries published by third parties for different purposes [3]. However, this practice is not without risks. In fact, the biggest and most popular publisher of JavaScript third party libraries, the node-package-module (npm) [4], does not ensure any security level as it does not perform safety checks on any of the packages that it publishes. This is a huge problem for the security and integrity of the systems that use this package manager. Making things worse, most key libraries do not provide a really high security level, since they do not perform any safety check [5].

The combination of the JavaScript ecosystem growth and the missed security checks presents an open door for successful cyber-attacks. For example, during the attack on the SCM platforms, an estimated of 4.5 millions of WordPress and Joomla users were hacked by using jQuery, the most popular library in JavaScript, to inject malicious code in their websites [6].

Today, a common practice for web solutions is to use content delivery networks (CDN). Their purpose is to provide webpages or web content, such as JavaScript libraries to end-users with high speed and availability, by applying caching techniques.

Despite the fact that dependencies are a big part of the developed software, there is no a clear unique solution to prevent attacks on CDNs [7] or to avoid including a library containing malicious code. This problem is very common and affects the integrity and security of the systems. It even can be used to produce DDoS attacks [8] on external systems by using reflection and amplification techniques [9]. Several dependency injectors, such as RequireJS [10], help with the task of modularizing and solving dependencies, but they not apply security checks for third party dependencies.

To make thing worse, outdated dependencies, which leave systems vulnerable to exploits by hackers, are a very common problem [11]. The problem has been exacerbated in the last ten years due the proliferation of sources of software dependencies and the inability to make automated upgrades when security problems are discovered. Thus, developers have to keep up to date with the latest news about security issues and continually upgrade the security checks in their software. A solution to this problem would be a tool providing automated checks and upgrades.

We want to tackle the problem of the lack of dependency injectors with built-in security checks for third party dependencies. In particular, our goal is to provide a way to perform security checks for JavaScript libraries used in web solutions. Thus, we extend PumaScript [12], a framework for code rewriting using techniques such as meta-programing and code introspection, to include checks for the integrity of third-party libraries, avoiding malicious code injection.

2 PumaScript Framework

PumaScript is a JavaScript-based framework where code transformation is achieved by replacing JavaScript code by semantically similar code included in PumaScript meta-functions. The process is shown graphically in Figure 1. It produces JavaScript code semantically equivalent to the original, but with some required properties, such as better performance or higher security levels.

A PumaScript program is a JavaScript program plus some meta-functions. Figure 1shows the steps followed to process a PumaScript program to produce a JavaScript program with the desired properties. In the first step, Esprima [13] parses the PumaScript program syntax and produces its Abstract Syntax Tree (AST). This tree is processed in Step 2 to replace calls to meta-functions by the sub-trees representing them. This steps actually replaces the undesirable parts of the JavaScript original program with the meta-functions providing the required properties. The third step is a simple prune phase, where meta-functions definitions are erased from the AST. Finally, the resulting AST is processed by Escodegen [14] to produce standard JavaScript code.

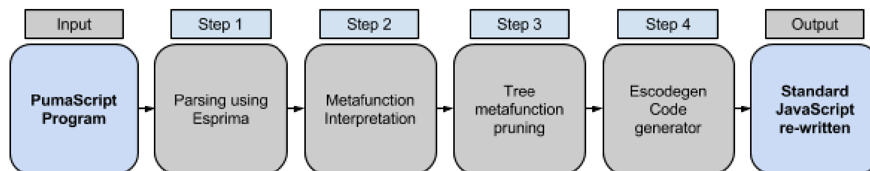

Figure 1: PumaScript program execution workflow and high level modules.

3 Methodology Applied to the Proposed Solution

We propose to use the introspection and rewriting capabilities of PumaScript to detect and fix this kind of security vulnerabilities. The PumaScript meta-functions are used to detect security issues as described in the previous section, allowing developers to use libraries from CDN while avoiding security risks.

Our team developed some "proof of concept" models and iterated on them in order to discover the better approach. The first iteration included the list of dependencies and a fixed CDN as parameters. Meta-functions for PumaScript where hardcoded in the runtime and the user has no ability of configuration. The second iteration included the parametrization of the CDN, while the meta-functions were taken from a meta-function database expressed in a JSON format file. The third and final iteration added integrity checks for external resources as described in the 2016's recommendation on Sub-resource Integrity (SRI) [15]. The system proposed then includes the capability of getting the integrity checks from a PumaScript database and injecting the dependency only if the security and integrity are confirmed.

3.1 Design of the solution

The design of the concept could be developed, and demonstrated that PumaScript framework could be used as part of the overall solution.

Given a determined list of security features we want PumaScript to test is the list of meta-functions we need to include. The PumaScript injector will look for those in the Meta-function database automatically and adds them in the application to test.

These days the DDoS attacks are very common, and with those the usage of CDN to generate one. For this reason that W3C created a new document with the guide to include resources from third party and avoid the problem by adding integrity checks. The problem is that still today there is no automated tool that includes and compares the SHA signatures and is responsibility of the developer to include these.

To complete the puma injector with the missing features we created a new iteration that starts from a simple puma.json file that contains the descriptive

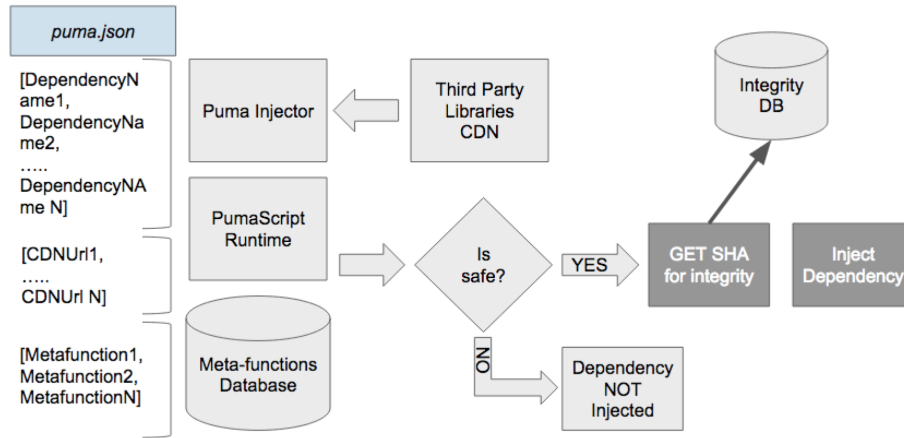

Figure 2: PumaScript program execution workflow and high level modules.

configuration and delegates the process of dependency loading. Takes all the information on the dependencies needed, the list of CDN that can be used and the Meta-functions that need to be applied.

Once taken this the process starts by downloading the raw code of the library, pass to PumaScript runtime and observe the result. If the dependency was marked as safe by Puma the process will continue with the integrity checks.

The integrity checks has complementary operations, first it retrieves from a database the integrity value for the specific library and version. This library has a white listed list containing dependencies names with the integrity values for each one.

4 Conclusions and next steps

After testing the design we could demonstrate that the solution is possible to generate. The design also allow to determine if a third party library does not comply with the security checks or the integrity checksum before load. This help to generate safer solutions by detecting security errors on loading avoiding possible security exploits that generate DDoS attacks. Although we found improvements areas on the PumaScript runtime to improve some parser cases on big libraries, the POC was consider successful. In near future the job must continue on extending and improving the injector to resolve as easy as possible for the developers this problem that today none of the dependencies injectors can resolve.

References

[1] StackOverflow. Survey 2016. http://stackoverflow.com/insights/survey/2016.

[2] NodeJs Foundation. Nodejs. https://nodejs.org/en/foundation/.

[3] OWASP. Third party management. https://www.owasp.org/index.php/3rd_Party_Javascript_Management_Cheat_Sheet.

[4] NPM. Npm introduction. https://docs.npmjs.com/getting-started/what-is-npm.

[5] ZDNET. Js hackers playground. http://www.zdnet.com/article/an-insecure-mess-how-flawed-javascript-is-turning-web-into-a-hackers-playground/.

[6] AVAST. Wordpress and jommla hacked by facked libraries. https://blog.avast.com/wordpress-and-joomla-users-get-hacked-be-aware-of-fake-jquery.

[7] IETF. Rfc cdn. https://tools.ietf.org/html/rfc6770.

[8] OWASP. Denial of service definition. https://www.owasp.org/index.php/Denial_of_Service.

[9] Fastly. Ddos definitions. https://www.fastly.com/sites/default/files/Fastly-Bizety%20DDoS%20White%20Paper.pdf.

[10] RequireJS. Requirejs. http://requirejs.org/.

[11] Sajjad Arshad William Robertson Christo Wilson Tobias Lauinger, Abdelberi Chaabane and Engin Kirda. Thou shalt not depend on me: Analysing the use of outdated javascript libraries on the web. http://www.ccs.neu.edu/home/arshad/publications/ndss2017jslibs.pdf.

[12] Nestor Navarro Emanuel Ravera Ricardo Medel, Alexis Ferreyra. Plataforma de meta-programación para javascript. In *CONAIISI 2015*, Buenos Aires, Noviembre 2015.

[13] Esprima. Esprima js parser. http://esprima.org/.

[14] Escodegen. Js code generator escodegen. https://github.com/estools/escodegen.

[15] W3C. Sub-resource integrity best practices. https://www.w3.org/TR/SRI/.