JavaScript static security analysis made easy with JSPrime

Nishant Das Patnaik & Sarathi Sabyasachi Sahoo
nishant.dp@gmail.com & sarathisahoo@gmail.com

JavaScript is the lingua-franca of Web 2.0 and, recently, with the advent of mobile SDKs JavaScript seen the light of being used as a mainstream programming language for 'hybrid' mobile applications which combine web applications fluidity and with native capabilities. However as with any other programming languages we have seen that developers are not very cautious about writing secure applications in JavaScript, be it rich web applications or mobile applications which leads to vulnerabilities like DOM-based Cross-Site Scripting attacks, and the situation becomes even more critical when the application is written on Node.JS or its variants which may lead to arbitrary JavaScript code injection in the context of the server.

The following section summarizes the outline of this Whitepaper:

Introduction

Insecure JavaScript programs in the client-side applications can easily lead to script injections that can run with the same privileges that the application is running. Script injection is browser-based applications due to is often known as DOM-based Cross Site Scripting attacks. This variant of XSS is in no way different from it's siblings in the context of its impact. However traditional tools that rely on pattern-match, in HTTP request-response, do not detect these vulnerabilities. Hence, lots of modern applications are still vulnerable to this attack. This is not just limited to browser-based applications; this type of code injection might also be a risk when developing Node.JS based applications. However in those cases it impact would be more lethal: from completely taking over servers to creating denial-of-service attacks against your application. It is not essentially a fact that developers are not aware of this type of vulnerability nor they are aware of its mitigations. The problem lies when organizations become agile in releasing code to the consumers. This is when the demand for fast shipping, leads to security issues in the code. Hence we need better and automated tools that can help developers to write secure code right from their IDE. This would save a lot time and money for organizations.

The Problem

Until now it seems to be a very serious problem and the solution seems to be very obvious. You might be thinking either of two approaches to tackle this problem: 1. Build a dynamic scanner for Black-box testing for the security team, or 2. Build a static code analyzer for the development team. If you are thinking either of these two, you are on the right track. However let us explain you the challenges in both of these approaches. Dynamic Black-box scanner is a good option for a very limited type of architecture of the application, and what we mean by that is today the applications have become too much complex and it's virtually impossible for an automated tool to scan these applications with no manual intervention, as each and every application is so different. And the moment a tool depends upon manual effort it efficiency becomes proportional to it's user and secondly humans are meant to work on smart problems (think of business logic flaws and similar issues); repetitive tasks should be automated. That's our honest thought.

Now coming to the second option of building a static security analyzer is the option we preferred. Since we already said the problem we are discussing is all about JavaScript, I would like to introduce you to this beautiful and amazing language. JavaScript is a dynamic language, which is why it is very loosely typed. It Object-based where properties can be created on demand. It fully supports Prototype-based inheritance, First-class functions i.e. functions can be passed as arguments to other functions, function can return another function, function can be stored in data structures etc. It also supports closure, which is considered to be an advanced feature of the JavaScript language, and it's understanding is essential in mastering the language. Runtime type casting and coercions are other challenges that are essential for achieving high-degree of correctness of the analysis. So by now we hope you might realize why manual code review of JavaScript applications can be intimidating. Also developers are seldom writing code in pure JavaScript i.e. without using any 3rd party JavaScript libraries like YUI or jQuery or MVC frameworks. In such scenario it becomes difficult for a code reviewer to learn the insecure coding practices of these libraries and keeping himself updated up till the latest version might be really challenging at times.

Introducing JSPrime

JSPrime is a lightweight source code scanner for identifying security issues using static analysis. It is written purely in JavaScript to analyze JavaScript. Uses the open-source ECMAScript parser: http://www.esprima.org JSPrime is mostly a developer centric tool. It can aid code reviewers for identifying security issues in 1st pass. Security professionals may find it useful during penetration testing engagements.

Following is a brief description of the working of JSPrime:

- Feed the code to Esprima, to generate the AST.
- Parse the JSON AST, to locate all sources (including Objects, Prototype) and keeping track of their scopes
- Parse the AST, to locate all assignment operations related to the sources, while keeping track of their scopes
- Parse the AST to locate sinks and sink aliases, again keeping track of their scope.

- Parse AST to locate functions (including closures, anon functions) which are fed with sources as arguments and while tracking down their return values.
- Once all the sources, source aliases are collected we check for any filter function on them, rejected if found.
- Remaining sources, source aliases are tracked for assignments or pass as argument operations to the collected sinks or sink aliases.
- We repeat the same process in reverse order to be sure that we reach the same source when we traverse backwards, just to be sure.
- Once we confirm that, we then extract the line numbers and their statement and put it in the report we generate with different color-coding.

JSPrime has some really good features that appreciated by developers. It is capable of following code execution order and handle first-class functions. It has the ability to analyze Prototype-based inheritance and while understanding type-casting. It has knowledge of sources and sinks of pure JavaScript, YUI & jQuery and is also aware of context-based filter functions which has to be manually supplied for each library, though and can be updated at will. Variable, Objects, Functional scoping is kept in track during the analysis to lower false-positives rate. Control-flow analysis & data-flow analysis is a integral part of the tool. However, it can't detect 100% of the issues, like any other tool, though the intelligence will only grow with time. It can't learn sources and sinks automatically nor handle obfuscated JavaScript. It also can't report issues in minified JavaScript, unless beautified. It can't analyze dynamically generated JavaScript using 'eval' or similar methods. Having said that presently the tool handle up to 1500 lines-of-code, in a single scan and a Node.JS port is available for server-side web service like setup in enterprises. It's robustness is largely dependent on Esprima parser, can be the 1st point failure. You can find a list of the test cases that JSPrime is able to analyze here: http://goo.gl/ju6oq You can add your own test cases and help us improve the tool.

Conclusion

This is project is actively work-in-progress with a promising roadmap. We plan to work on improving the performance and stability of the stability of the tool for making it even more enterprise ready. Future releases would see multiple file scanning capabilities with full Node.JS project scanning capability. A much requested feature that is coming its way is an IDE Plugin (Notepad++, WebStorm,??). Added support for even more 3rd party libraries along with string manipulation function simulation for lesser false negative rate. And of course your suggestions.

Credits

Aria Hidayat, Esprima.org
Paul Theriault, Mozilla Security Team
Bishan Singh - @b1shan
Rafay Baloch – rafayhackingarticles.com