Raiding Scripts the Right Way

By Paul Pemberton

Email: Paul.pemberton@tufts.edu

Mentor: Joseph Pemberton

Contents

0. Abstract

As internet browsers become more and more customizable, there is a correlated demand for browser add-ons and extensions. The top four internet browsers (Google Chrome, Mozilla Firefox, Internet Explorer, and Safari) have all developed extension modules that allow users to add customized functionalization to their browsers. One of these added functions is JavaScript blocking. This technique (JavaScript blocking) presents a dangerous road to securing users from the dangers of JavaScript based web attacks. By providing users with either "blocked" or "non-blocked" JavaScript components, these tools cause web pages to become completely unusable. With many pages left non-functional by script blocking tools, the web becomes a tedious task. Users tend to ignore blocked scripts and allow scripts to run without assessing the actual risks involved. A binary system of scripts being "on" (non-blocked) or "off" (blocked) does not provide a good enough solution to the risk of web based script attacks. There is a clear and present need for a tool that presents users with information or allows them to see the risk levels associated with certain scripts. Without a tool that allows users to learn from blocked scripts, there will likely be more and more social engineering attacks due to user ignorance of risks associated with JavaScripts. While the issue presented is non-trivial to solve, more thought must be invested in protecting internet users from script attacks.

1. Introduction

Currently, Google Chrome and Mozilla Firefox support the leading extensions for script blocking. NoScript, which is run in Firefox, utilizes simple JavaScript detection and then stops all

JavaScripts from running on the current web page (Faziri). NoScript can be customized by the user to block/not block certain pages or scripts. NotScripts, Google Chrome's version, incorporates a few extra components, such as an explicit white list for sites as well as password protection (so that the extension cannot be overridden easily) (Lifehacker). Both of these tools are very effective at blocking JavaScript, but the tools exhibit a fundamental misunderstanding of the user as well as the issue with JavaScript based attacks.

1.1. Browser Equipment (The Binary World)

Currently, JavaScript defenses utilize a binary classification system. The extensions completely block scripts from running or allow them to run. The existing binary system does not provide adequate information to the user. When determining whether or not to run a script, the user must blindly decide if a piece of JavaScript can be safely run on their browser. Binary delegations are not good enough for the world of internet security; it is too dynamic in nature for an "on/off" switch to be used to protect users from some of the most dangerous web attacks (Mookhey).

1.2. The Common User

In addition to the shortcomings of a binary system for JavaScript control, the current extensions exhibit a complete lack of regard for the everyday computer user. Most of the specific settings (in terms of white-lists and JavaScript behaviors to allow) are difficult for a competent computer scientist to navigate. The existing tools render almost any JavaScript website (i.e. any website) partially or completely unusable unless

the preferences for the tool are set to be less stringent. The user is still presented with either “on” or “off” scripts and has not real idea of why certain scripts are blocked. Again, the tools currently in use neglect the needs of the everyday user. How can a non-computer scientist be trusted to make an autonomous determination about what scripts are “safe” or “dangerous.”

The US-CERT (United States Computer Emergency Readiness Team) advises everyday users to “pay attention to the URL of a website” and “take advantage of any anti-phishing features offered” (US-CERT). Most internet citizens have enough navigating through the plethora of web sites out there, thus, while the US-CERT advice is sound, it is unrealistic to expect. Additionally, anti-phishing features have potential to provide a false sense of security. Of 10 tools tested, “only one tool was able to consistently identify more than 90% of the phishing URLs,” but had a 42% false positive rate (Zhang et. al). Since phishing is only a limited portion of the attacks that can be carried out with JavaScript, it is crucial that users are protected via other means. The “common user” does not have the expertise necessary to determine what sites are ok to white-list, nor does he have the ability to decipher JavaScript source code and make a determination about how malicious it looks. As good citizens, it is the responsibility of developers to build tools that presents the necessary data to the user in an interpretable format.

1.3. Social Engineering Attacks and XSS

With the current script blocking tools, there are clear avenues for social engineering attacks. For example, if a user consistently uses a web site with an expired certificate,

then the attacker can infer that the user is more likely to proceed to a web site that has an expired certificate in the future (i.e. a malicious site). Additionally, cross-site scripting (XSS) attacks usually utilize social engineering. Even when paying attention to the URL bar of a browser, there are XSS attacks that work through pop-up windows or by executing remotely downloaded source code. Again, approaching this problem with the mindset of stopping all of the malicious scripts on the internet is both naïve and futile. The premise of social engineering attacks is that even with complete blockage of malicious scripts, the user is ultimately in control of the web site content. If the user decides to override the script blocker due to a message or other social attack, then even a perfect script analyzer will fail. Given that social engineering attacks are becoming more and more prevalent (as evidenced by the US-CERT warning), the computer science community needs to recognize that merely blocking all JavaScript and then entrusting the user to only activate benign scripts is not an effective treatment of the problem.

2. Short Comings of the Script Blocking World (A Motif)

While script blocking seems to be one of the newer and more popular fields in malicious code prevention, there is a motif throughout computer security history that points to why the current tools may not be good enough. Additionally, there are extremely dangerous implications that compromising browser usability can cause (mainly in terms of social engineering attacks). Finally, these script blocking tools rely on stopping scripts, but there is room for teaching users how to be active and protective citizens in regards to malicious scripts.

2.1. Virus Detection Issues

Virus software provides both a false sense of security as well as a tool for attackers to determine if they will be caught. "Virus writers are often a step ahead of the software, and new viruses are constantly being released that current anti-virus software cannot recognize" (Stanford). In the same way that viruses are constantly being written to evade anti-virus software, new JavaScript attacks are being concocted to evade easy detection. It is unrealistic to expect users to be able to determine what is malicious when malicious code is evolving at a very rapid rate. It would be futile to attempt to create a system that correctly analyzes all JavaScript and designates it as benign or malicious. Since code blocking (which relies on the user being able to determine the nature of code) and code scanning (which relies on a perfected anti-virus model) are both non-attainable, there must be a compromise that can be achieved.

2.2. Browser Usability Compromise

As previously discussed, besides being an unrealistic method for protecting users from script based attacks, the currently existing tools for script blocking pose a threat to browser usability. Since JavaScript represents a significant portion of many web pages, blocking every instance of a script will render most pages useless. The user is able to activate each script element individually, but the process of sifting through what is "necessary" can be tedious and discouraging. This process may actually lead users to delete tools or worse yet; blindly allow JavaScript elements to run. When protecting users, the browser should fully functional unless a serious threat or risk is detected.

2.3. Live Script Monitoring

The solution and compromise to this problem is live script monitoring that utilizes a spectrum of warnings to keep users informed about the risks of visiting web sites. This live script monitoring system (LSMS) can be implemented to scan the JavaScript that is being run on the page and make a determination about what dangers the script or page pose. This method will avoid rendering pages unusable (since it does not interfere with script running), and it will also not fall into the pitfall of anti-virus software by making binary determinations about the scripts. As with every piece of security software, a skilled attacker could leverage the source code of the LSMS in order to trick users into thinking that pages are safe. The main purpose of this software would not be to create a fool-proof script filter, this tool would be used to inform and educate users about the sites that they visit and notifying them of suspicious behaviors (helping the US-CERT).

3. Defenses (or Lack Thereof)

Currently, defenses against JavaScript based web attacks lack user empowerment. These tools also render websites unusable in many cases, which creates a situation where users are forced to make uninformed decisions about running foreign scripts. One of the NoScript review pages contains a section titled “To Block or Not to Block” in which the writer describes how to determine what scripts are “good” (Faziri). All of the scripts that Faziri suggests the user allow have just as much potential to cause damage as any other script. Nothing stops attackers from injecting malicious code into the menu bar of their website. Again, NoScript fails to provide the user with any useful information. It is effective at blocking scripts, but when a user needs to

activate scripts, they are blind. Defenses need to be implemented in a different way in order to protect users from arbitrarily activating scripts.

3.1. Static vs. Dynamic Analysis

In malware classification tools, there are two types of code analysis techniques: static and dynamic analysis. Static analysis consists of decompiling the source code and determining the behavior of the code based on flow graphs. Static analysis techniques can also utilize signatures (i.e. MD5 value) to determine if the code has been processed/classified previously. Dynamic analysis, on the other hand, runs the code in a sandbox and then identifies what types of behaviors are exhibited (Cavallaro). The live script monitoring system (LSMS) would likely use a static method to analyze JavaScript. By looking for specific phrases (such as “alert” or “window”), the tool should be able to determine the risk level of the script. By culminating information about all of the JavaScript elements on a web page, the LSMS could provide the user with information about the level of danger associated with the page (i.e. how risky would it be to click on a link?).

3.2. NoScript, NotScripts (Where They Miss)

The proposed LSMS (live script monitoring system) aims to solve two main shortcomings that both NoScript and NotScripts have. Firstly, web page usability will not be impeded by the LSMS in most cases. There are certain JavaScript phrases that should trigger blockage of webpage access in order to protect the user from confirmed malicious

scripts. Overall, however, web pages will be presented in complete form and the user will be cautioned about using certain features (such as links) based on the results of the static analysis. Ideally, the LSMS could be used in conjunction with NoScript or NotScripts, but for the most usable web pages the LSMS should be used alone. For maximum security, the two tools could be used together.

Besides a lack of web page usability, NoScript and NotScripts fail to inform the user about the risks of visiting web pages or activating scripts (Faziri). These tools could add a feature that allows the user to receive the score of a static analysis of a specific JavaScript before they allow it to run. While this may again be the most secure solution to the problem, efficiency of browser use is a large component of this issue. The LSMS alone would provide the user with live information (i.e. no interruption of web page function). For users concerned with efficient function of their browser, the LSMS alone would be the optimal system. Both of the issues with NoScript and NotScripts can be fixed by the LSMS, and for added security, the two tools can be run in conjunction (LSMS and NoScript/NotScripts).

4. Modifications to Make

ScriptRaider, a live script monitoring system, is a prototype tool developed by Paul Pemberton (based off of source code from the Chromium Team). This tool presents a first look at what JavaScript monitoring tools could look like if instead of focusing on blocking scripts the tools were built to inform users about risks associated with visiting sites. As of now, the tool is

functional, but the static analysis algorithm for determining the nature of the pages being analyzed needs to be corrected.

4.1. ScriptRaider (A Solution?)

Currently, ScriptRaider informs the user about how dangerous the page that they are visiting is by utilizing the defcon ranking system. The following ranks are designated to inform the user about varying levels of risk:

DEFCON 5: lowest risk level – ScriptRaider posts text to the Chrome info bar that the page is low risk.

DEFCON 4: low risk level – ScriptRaider posts text to the Chrome info bar indicating slight risk.

DEFCON 3: moderate risk level – ScriptRaider posts text to the Chrome info bar indicating that the user should be mindful of clicking links.

DEFCON 2: high risk level – ScriptRaider produces a pop-up window that warns the user of heightened risk. Chrome info bar contains text that user should consider navigating away from page.

DEFCON 1: highest risk level – ScriptRaider produces a pop-up window warning the user that the page is very dangerous. Chrome info bar contains text that user should leave site immediately.

As stated previously, this extension is merely meant to serve as a proof of concept. Currently, it uses a "page scoring system" that checks for the following expressions in the JavaScript of pages visited:

"alert" – may cause a pop-up window which can be used to execute an XSS attack.

"window" – May redirect user to an alternate page (XSS attack).

"http" – May call code from an outside source (unknown intent without source code)

"src" – May call code from within the page or an outside source (unknown intent of code)

"eval" – May be evaluating user input text, can lead to browser/plugin crash.

Further development would be necessary to provide a practical service to users.

4.2. Where We Need to Go

Script Raider must have two crucial areas of improvement that would allow for the tool to be used successfully by everyday computer users. First, the static analysis algorithm needs to be developed so that complex JavaScripts are correctly classified/scored. Merely looking for "hot" words is not a good algorithm for scoring JavaScript. The ideal way to develop an algorithm would be to compile a list of malicious and benign JavaScript and then determine what attributes are indicative of risky code. Additionally, the implementation of a local database or white-list would help users have control over their settings. Again, trusting non-computer scientist users can cause susceptibility to

social engineering attacks. Users should be extremely careful when adding scripts/web pages to the white-list.

Secondly, the notification system for alerting the user to risks should be more comprehensive. For example, the DEFCON 1 rating should prevent the user from accessing the page until the user consciously overrides the DEFCON 1 designation. Additionally, the info bar should change color with each rating so that the user is more alerted to the risks associated with their current web site. The info bar could also provide an option for the user to see which scripts are being flagged as dangerous. ScriptRaider clearly has a large amount of development needed, but as a proof of concept it demonstrates why website usability and educational risk alerts are valuable.

5. The Meaning of Script Security

The online world is becoming more dangerous every day. The best way to maintain security is to be informed about the online environments that you visit. A sound analogy is the comparison between a child who is kept in a bubble and a child that is taught about hygiene. The child in the bubble will be ultra-sensitive to any virus or bacteria that they come in contact with. If, by chance, a malicious organism slips through the plastic, the damage to the child can be catastrophic. The educated child, however, knows the risks and dangers of bacteria and viruses. He washes his hands and when he is confronted by bacteria and viral particles, he fairs better. Having all of the scripts on a page blocked is analogous to being in a plastic bubble. Security comes from learning and practicing safe JavaScript running.

5.1. Can We Do Better?

The answer is undoubtedly yes. Developers that are building tools to help shepherd uninformed internet users away from dangerous sites need to keep in mind that blindly blocking scripts from running is not effective. By informing users about the dangers of certain scripts/web pages, the internet community will greatly benefit. Malicious sites will be visited far less, and users will learn that even benign looking sites can house malicious code.

5.2. Why Aren't We Doing Better?

This question is slightly harder. The primary reason that script blockers are not providing the correct type of security is that these tools are written/developed by computer scientists. When computer literacy is taken for granted, it is easy to customize and tailor script blocking tools so that they work well. As a general public tool, however, these script blockers fall short. Most of the computer users who need these tools the most are completely illiterate in the world of programming. Without an expertise in computers, NoScript and NotScripts become obsolete. Users are forced to arbitrarily activate JavaScript without an inkling about what the script will do. By using a tool like ScriptRaider, users can be informed about what risks they are taking and make educated decisions about what pages to visit. For computer security to progress, education and information distribution must be utilized before user-driven tools are implemented (NoScript).

6. References

1. Cavallaro, Lorenzo. *Malicious Software and Its Underground Economy: Two Sides to Every Story*. Rep. Royal Halloway University of London, 30 Aug. 2013. Web. 26 Nov. 2013.

2. Faziri. "NoScript Protects Your Firefox." *NoScript Protects Your Firefox*. Gizmo's Freware, 12 Sept. 2013. Web. 17 Oct. 2013.

3. Mookhey, K.K., and Nilesh Burghate. Detection of SQL Injection and Cross-site Scripting Attacks. Publication. Black Hat, 2004. Web. 21 Nov. 2013.

4. "NotScripts Brings Real Script Blocking to Chrome." Lifehacker. Lifehacker. Web. 22 Nov. 2013.

5. "Security Tip (ST04-014)." Avoiding Social Engineering and Phishing Attacks. US-CERT. Web. 01 Dec. 2013.

6. "Virus: A Retrospective - Anti-Virus Software." Virus: A Retrospective. Stanford University. Web. 01 Dec. 2013.

7. Zhang, Yue; Egelman, Serge; Cranor, Lorrie; and Hong, Jason, “Phinding Phish: Evaluating Anti-Phishing Tools” (2006). Human-Computer Interaction Institute. Carnegie Mellon. Web. 01 Dec. 2013.

7. Supplemental Materials

Link to Source: https://github.com/pauljpemberton/ScriptRaider