

A Client-Side Browser-Integrated Solution for Detecting and Preventing Cross Site Scripting (XSS) Attacks

Bernice Chan and Gordon Chiu

Introduction

Our proposed work aims to create a client-side in-browser solution for mitigating cross-site scripting (XSS) and other attacks, which addresses concerns and shortcomings present in previous work. The solution will be implemented as an extension for the popular web browser FireFox; this extension will automatically detect possible instances of a cross-site scripting attack and prevent execution of malicious JavaScript code. We also proposed several additional features to help combat phishing, IDN domain name spoofing, and identifying possibly suspicious links.

Progress

The project can be divided into three components, which can be worked on separately and integrated at the end. These components are:

- *In-browser security manager*: Infrastructure for intercepting JavaScript code and programmatically disabling its execution (depending on the return result from the XSS threat analyzer). Infrastructure for intercepting phishing pages or suspicious links and alerting users prior to display (also depending on the return result of the phishing threat analyzer).
- *XSS threat analyzer*: Given a document, analyze for the threat of an XSS attack. Return results to the security manager.
- *Phishing threat analyzer*: Given a document, analyze for the threat of a phishing or suspicious link attack. Return results to the security manager.

In-browser security manager

Investigations have been made into developing the infrastructure for the FireFox extension, where the primary challenge is intercepting the malicious JavaScript *prior* to execution by the user. An original plan was adopt a JavaScript disable mechanism similar to that employed by the popular NoScript [1]. However, upon further investigation, the mechanism employed Mozilla's Configurable Security Policies [2]: all JavaScript is disabled by default, and a "whitelist" of domains is maintained. Essentially the NoScript extension is a fancy mechanism for managing this whitelist. However, this extension does provide an example of how to traverse the HTML Document Object Model (DOM) to find all instances of JavaScript execution.

The challenge of how to intercept JavaScript *prior* to execution remains. The mechanism of using a domain-based whitelist does not work, because XSS attacks commonly occur when executing JavaScript on a *trusted* domain. It is possible to implement the XPCOM interface `nsIContentPolicy`. This interface provides two methods, `shouldLoad` and `shouldProcess`, which are polled prior to the download, and then execution, of the JavaScript on the page. We can add a step to the `shouldProcess`. This step would pass the JavaScript content of the page to the threat analyzers prior to permitting execution. If the threat analyzer determines that the page should not be processed, some additional work would be required to identify all script calls on the page and disable them in the DOM.

XSS Threat Analyzer

The work on an XSS threat analyzer is proceeding independently from the investigations on the FireFox extension infrastructure. Our proposed XSS Level-0 and Level-1 detection schemes use the HTTP request query in addition to the resulting page to detect cross-site scripting attacks. Thus, we are currently working on an XSS threat analyzer, which takes as input two text files, the HTTP request query and the resulting page, and outputs whether or not an XSS attack is present (with a confidence value). The basic algorithm performs a fuzzy matching to see if any of the HTTP input variables become executable code in the resulting HTML page.

Phishing Threat Analyzer

Our proposed detection scheme uses the Google PageRank system to determine the reputation/relevance of a page. The preliminary work for the phishing threat analyzer has been completed. The mechanism for setting up a PageRank query with the calculated page checksum has been determined. A prototype browser toolbar to display the status of results was constructed, and the mechanism was tested by constructing queries and processing the responses for a number of websites. We will correlate the PageRank scores for known “reputable” and “unreputable” websites to determine the effectiveness of using the Google PageRank system to determine phishing threats.

Building a test vector set

The primary challenge is to build a large set of test vectors covering many different types of XSS attacks, to be sure our tool catches as many known attack classes as possible. Our initial work has been to build a test set to test the development of the XSS threat analyzer. There are two degrees of freedom when considering XSS attacks:

1. The type of vulnerability on the server
2. The delivery mechanism the attacker chooses (encoding of parameters, which tags to exploit, etc).

We are constructing a series of CGI scripts (written in perl) that mimic the different vulnerabilities. The simplest vulnerable page mirrors the HTTP request to the output. More complicated vulnerable pages perform various decoding tasks, including de-base64 encoding, common character encodings, and common escaping mistakes. A comprehensive list of these vulnerabilities types can be easily found on certain grey and black-hat security websites.

We are also compiling a list of attack vectors. These vectors are taken from a comprehensive list on a black-hat web security website [3]. These commonly include attempts at server-side filter evasion, using tricks such as encoding parameters, character encoding, upper and lower-casing text to bypass case-insensitive checks, etc. By automatically running the more than 50 attack vectors on each of the vulnerable perl CGI scripts, we can construct a large test set of hundreds of attack attempts – each one test vector.

Each test vector consists of a pair of the input attack vector and the output HTML page. We can use this large set to tune and measure the progress of our XSS threat analyzer as we develop it.

References

- [1] “NoScript - Whitelist JavaScript blocking for a safer Firefox”, <http://noscript.net>
- [2] Mozilla, “Configurable Security Policies”, <http://www.mozilla.org/projects/security/components/ConfigPolicy.html>
- [3] ha.ckers.org, “XSS (Cross Site Scripting) Cheat Sheet”, <http://ha.ckers.org/xss.html>