Malicious JavaScript Repository

Andrew Kruzan, Shaun Sitterley, and Steven Ristau

***Abstract -* This report summarizes three major aspects of Team 2’s Malicious JavaScript Repository project: background/motivation for the project, project objectives and project outcomes.**

I.    INTRODUCTION

Before going into detail on the specifics of the project and the team’s current work, some background knowledge on JavaScript, as well as the motivation behind doing this project is discussed.

1. *Background*

Malicious JavaScript code is simply JavaScript code that has been constructed for the purpose of performing malevolent actions on an unsuspecting user. More generally, JavaScript can essentially be thought of as the programming language of the Web.  Whenever a web page is loaded on a website, there is JavaScript code running in the background that specifies the behavior of the elements on the page. Depending on the different actions a user performs on that page, there can be varying results.  These results can range from putting a text box on the screen whenever a page is loaded, to submitting user information to a server when a button is clicked.

From an attacker’s point of view, the ability to control the behavior of web pages is very appealing.  If attackers can find a way to inject their own JavaScript code into the page, then they can change the behavior of the page however they wish. Instead of simply putting a text box on a screen upon page load, the attacker can change this to redirecting a user to one of their malicious pages loaded with malware.

Injecting malicious code onto a page, however, is not that simple. If a site owner wanted to determine if their web pages contained any malicious code, they could do so by simply examining the source code and verifying the validity of the JavaScript it contained.  As a result, malware writers have become smarter and have started to develop methods to hide the functionality of their code. The primary method used in malicious JavaScript is code obfuscation. This involves translating the malicious code into gibberish, not comprehensible by anyone looking at it. Then, at a later time when the code needs to be executed, it is translated back into the original code so it can be ran properly.  With this in mind, it will be necessary for the team to send any obfuscated code to team 3 so that they can de-obfuscate (reveal) the true purpose of these samples.

1. *Motivation*

As discussed above, the purpose of malicious JavaScript samples can be difficult to determine, as they are often obfuscated.  Additionally, there are many different types of samples.  Each one of these types can perform its own actions.  Therefore, the primary motivation behind this project is to obtain a fuller knowledge of the behavior of malicious JavaScript samples.

II.    OBJECTIVES

This project’s main goal is to create a repository of malicious JavaScript samples that are publicly accessible to the research community.  To accomplish this goal, a few steps must be taken.  Firstly, we must accumulate as many samples as possible.  We must then place them in one of several categories of malicious JavaScript. And lastly, we must perform both static and dynamic analysis on notable samples for each category.  The primary task now is to use the tools at our disposal to best accomplish these steps to create a robust repository complete with analyzed examples and a library of samples.

*A. Finding Samples*

The online community for malicious JavaScript analysis is present on many smaller websites. One such site shows several code samples that the owner has come across. These explain how obfuscated code is de-obfuscated and analyzed, showing how the code injects iframes attempting to download malware from a malicious site [1]. These are helpful examples, but there are none left to be analyzed.

Our plan is to compile a large amount of samples that we can analyze. We will utilize VirusShare to meet our demand. VirusShare is a website with over 19 million samples currently on file for users to download and analyze [2]. With an account, we will have access to all of the malicious JavaScript files on the site. We will utilize a scraper to find the JavaScript among the other files so we only download the files we need.

1. *Categorize Samples*

Our second task is to categorize the malicious JavaScript into one of several types.  There are three popular types on which we will focus. The three major categories are drive-by downloads, malicious redirection, and malicious advertising. If evidence of other significant types arises, we will create another category.

1. *Perform Analysis*

Our final task upon gathering as many samples as we are able to is to perform an analysis on some unique or notable examples of each type. We will utilize several sites to help perform static and dynamic analysis on the samples. In addition to the use of these helpful sites, we will also take a look at the code itself to help them determine what exactly is going on. Since Javascript is written in a text editor, any text editor will be able to open the file.

III.    OUTCOMES

We have researched the various malicious JavaScript categories, identified a source of real world viruses that we can analyze, wrote a python script to aid us in collecting samples, analyzed, categorized and wrote descriptions of a few samples and we also made a repository that is available to anyone who is interested in it. As mentioned previously, the team has identified the three main categories of malicious JavaScript to be drive-by downloads malicious redirection, and malicious advertising. As our source of real world viruses, the team has chosen to use VirusShare and has acquired an invitation from the site. Some of the tools the team used to analyze the collected samples were VirusTotal, Wepawet[3] and JSDetox[4].

1. *Category Research*

Drive-by downloads are accomplished usually by the malware injecting an invisible or disguised iframe into the webpage.  This iframe has the ability to download malicious code through the browser from actions performed on the page.  The malicious part of the page may attempt to fool the user into performing that action and unintentionally downloading it [5].

The second popular type of JavaScript malware is malicious redirection. This can be similarly implemented to drive-by downloads. It can be housed in an iframe or other disguised element, waiting for interaction. This, however, redirects your browser to a malicious site that may look legitimate. In some cases, the site resembles an exact copy of a login service to another site in order to steal credentials.  These can be dangerous if the user is not careful with meticulously checking the web address they are providing their information for.

The third malicious JavaScript type we have deemed important to analyze is malicious advertising (or malvertising) [6]. Banner ads on pages are a target for many exploits because of their ubiquitous nature. They can be used to download a malicious file or for malicious redirection or possibly more discreetly implemented malware. Security vulnerabilities on ads are possible to allow spyware or virus infection to be caused by just loading the ad itself. The ad is usually for a legitimate product or service on a high-traffic site or program using ads [7]. These can be damaging to brands and sites that use these ads and dangerous to many users’ private information.

1. *Sample Collection*

As mentioned above, the team used VirusShare to collect samples for our analysis. VirusShare has an extremely large number of samples but, unfortunately, these samples are not organized in any particular manner. Basically, there are lists upon lists of hashes that a user of the site must use to search each sample individually. Since we were required to collect a large number of samples, this would be an extremely tedious process. In order to speed up the process, the team wrote a python script to automate it.

The script mimicked what a regular user would do to find samples of a specific type. The basic logic of the script was to navigate to the VirusShare website, enter a hash code into the search bar, search it and then scan the returned page for a file type of HTML document. In order to accomplish this, the team used two python libraries, Requests and Beautiful Soup. Requests is a python library that allows users to easily integrate with web services seamlessly. In our project, we used it to navigate to the various parts of VirusShare, log into the site and then send post requests to the sites server to return the samples information and download page. Beautiful Soup is another python library that allows users to easily extract data from a pages source. While VirusShare doesn’t specifically organize the samples they provide, they do at least tell you what type of file it is on the download page. Anytime our scraper came across a web page with a file type of HTML document, it would save the download link to a text document for the team to make use of later.

1. *Analysis*

After the team had collected a large variety of samples it was time to analyze them. In order to analyze these samples, the team used VirusTotal, Wepawet and JSDetox. JSDetox was used to ease the analysis process and VirusTotal and Wepawet were used to verify malicious URLs.

The team analyzed their collected samples with JSDetox. JSDetox is a tool that aids in the manual analysis of malicious JavaScript code. The setup process only took a couple of minutes and the team decided it would be best to install it on a virtual machine to reduce the risk of accidentally infecting the local machine. The tool had many useful features that helped speed up the analysis process. One feature offered by JSDetox was that it would extract all of the JavaScript code from the HTML which helped us focus on what was important. Another feature offered is the simplification of complex code. Lately, programmers who write malicious JavaScript tend to craft it together in such a way that it makes it next to impossible for the average person to determine what is actually going on. JSDetox does its best to simplify the code and make it legible to the person analyzing it.

Often times, the malicious JavaScript will link to an outside website that contains the malicious content. In order to test whether these outside websites contained malicious content, the team used sites such as VirusTotal and Wepawet to see if the site was a reported malicious site. These tools were very easy to use and are constantly being kept up to date. If the URL that the JavaScript linked to was reported as malicious on these sites, then the JavaScript is considered malicious.

1. *Categorization*

Once the sample was analyzed, the team categorized the sample. There are three main categories that the team focused on which included drive-by downloads, malicious advertising and malicious redirection. Samples that cause a user to unintentionally download malicious software were put in the drive-by downloads category. Samples that infect a viewer’s computer through the display of advertisements fall into the malicious advertising (malvertising) category. Samples that redirect the user to an outside malicious website fall into the malicious redirection category.

1. *Descriptions*

After the sample has been analyzed and categorized, the next step is to write a description of it. When writing the description, the team aimed to keep it short but detailed. We wanted to inform the user without going into too much detail and overwhelming them. The descriptions also include which malicious JavaScript category that particular sample falls into.

1. *Repository*

After we completed analysis on some samples we created a repository on GitHub. This repository holds all of our analyzed samples as well as the tools we created and used to aid us in our analysis. The idea behind this is that the community can begin performing their own analysis on samples and everyone can learn more about malicious JavaScript. The tool that we created and uploaded to the repository was the VirusShare scraper. This tool will allow users to collect their own malicious JavaScript samples from VirusShare. This tool can easily be modified to collect other types of malicious software if the user wishes to do so. In addition to the scraper, we also uploaded our list of 7000 download links to malicious JavaScript samples. There are links on the repository to websites that we used for our analysis like VirusTotal and Wepawet as well links to software such as JSDetox.

.

REFERENCES

[1] *Examples of malicious JavaScript* [Online] Available: http://aw-snap.info/articles/js-examples.php

[2] *VirusShare.com* [Online] Available: <http://virusshare.com>

[3] *Wepawet* [Online] Available: <http://wepawet.iseclab.org>

[4] *JSDetox* [Online] Available: <http://www.relentless-coding.com/projects/jsdetox/>

[5] R. Siciliano (2013, Apr 2) *What is a “Drive-By” Download?* [Online] Available: <https://blogs.mcafee.com/consumer/drive-by-download>

[6] W. Salusky (2007, Dec 6) *Malvertising* [Online] Available: https://isc.sans.edu/diary/Malvertising/3727

[7] L. Zeltser (2011, Jun 6) *Malvertising: Some Examples of Malicious Ad Campaigns* [Online] Available: <https://zeltser.com/malvertising-malicious-ad-campaigns/>