DYNAMIC IDENTIFICATION OF MALICIOUS OBFUSCATED JAVASCRIPTS

Vrushali S. Bari
PG Student,
Department of Computer Engineering,
SES's R. C. Patel Institute of Technology,
Shirpur,India.
vrusshalibari@gmail.com

Nitin N. Patil
Department of Computer Engineering,
SES's R. C. Patel Institute of Technology,
Shirpur, India.
Er_nitinpatil@rediffmail.com

***Abstract*— JavaScript is a scripting language. On one hand, it allows developers to create client-side interfaces for web applications. On the other hand, the malicious JavaScript code infects the web user and web browser. In order to detect malicious activities, two methods viz. static and dynamic detection methods have been discussed in the literature. The dynamic analysis method has better capability in detecting malicious activities compared to the static detection method. In this paper, we present a method based on Support Vector Machine (SVM) that would identify the malicious JavaScript code at the beginning itself. In addition, our proposed method supports the analysis of obfuscated code and analyzes the system offline.**

***Keywords:* SVM, Obfuscated JavaScript, Static Analysis, Dynamic Analysis.**

I. INTRODUCTION

The use of Internet has become an integral and necessary part in these days. The internet based activities includes *e-banking, e-mail, e-commerce* etc. The people used to carry out these activities in their day-to-day life. The user needs to take care while using internet for the *e-banking*, since there may a probability of hacking the information by the people and/or by the malicious software. Thus, with the growth of the internet technology, the data security has become a prime area of the research community. We notice in the literature that the prime focus of the attacker is more on the client-web applications. In this, an attacker would simply create a malicious webpage and propagate the malicious script to the clients on web. Most web based attacks take place on legitimate websites. Various types of threats have been discussed in the literature. Among them, a SQL injection attack is the most common type of attack. Through HTML and URIs, the Web was vulnerable to attacks like cross-site scripting (XSS) that came with the introduction of JavaScript. On the other hand, malicious JavaScript code is particularly hard to detect in the content of web pages. For example, JavaScript attacks regularly analysis for the browser environment, check for particular vulnerabilities and use dynamic exploiting techniques, such as heap spraying, for compromising a victim's system. In addition, the direct execution of the code also enables effectively obfuscating the attack, such that indicative patterns are only visible at run-time and not accessible by static detection methods viz. conventional anti-virus scanners. As a result, the detection of malicious JavaScript code at run-time is a prime area in the domain of web security. Various methods have been developed for dynamically detecting the malicious activities. In the work of [Konrad Rieck et al. 2010], [C. Curtsinger et al., 2011], and [M. Heiderich et al., 2011] discussed the method of learning for the detection of the malicious behavior. These learning based detectors provide an accurate identification of malicious code at run-time. However, none of the detectors has been optimized for the early detection of attacks. The longer a malicious code runs, longer it causes the harm to the system. However, spotting attacks is a difficult task and it has two main challenges: First, malicious behavior should be

detected as fast as possible, but never at the prize of accuracy. Second, the detection needs to be resistant against evasion that simply delays malicious activity to a later point in the execution of the code. In this paper, we address the problem of detecting malicious behavior in JavaScript code as early as possible. We introduce an optimized learning method for faster identification of malicious behavior, which extends the learning algorithm of support vector machines, such that the accuracy and time of detection are jointly optimized during learning. Our proposed approach uses Bayesian classification of hierarchical features of the JavaScript abstract syntax tree to identify syntax elements that are highly predictive of malware. Our experimental evaluation shows that the system is able to detect JavaScript malware through mostly dynamic code analysis effectively. We present fast multi-feature matching algorithms that scale to hundreds or even thousands of features.

The rest of the paper is organized as follows. Section 2 presents the related literature work, Section 3 present the methodology of the system. Section 4 presents the experimental results and finally, the conclusion has been presented in Section 5.

II. RELATED WORK

In this section, we present the brief overview of the malicious software, which have been used to infect the victim system. The issues related to the automatic exploitation in the network services have been discussed in [Microsoft Corporation 2008]. In addition, the dictionary attacks have been presented for the system access to remote users. In this, the attacker makes an entry in to the system via trying various passwords that have been stored in a dictionary.

Secondly, a drive-by downloads have been discussed in [Provos et al. 2008] [M. Cova et al., 2010] to attack on the victim web browser. It works in two steps: firstly, it fetches the malicious code from the web, and secondly, it executes it on the victim machine. In addition, Provos et al. [2008] point out that 1.3% of Google search queries tries to attack on the victim machine.

Thirdly, the issues related to the social engineering attacks have been discussed in [Stasiukonis 2007]. Fourthly, various detection systems viz. Cujo, Prophiler, BLADE, IceShield, and Zozzle have been presented in the literature. [Konrad Rieck et al. 2010] has presented Cujo for the prevention of drive-by downloads. Cujo inspects web pages and block malicious code. [D Canali et al., 2011] has proposed efficient system that analyzes the web pages and identifies the malicious web pages. [L.Lu et al., 2010] proposed BLADE that protect a host machine from the dirve-by download attacks. [M. Heiderich et al., 2011] proposed IceShield that help to detect and protect the user from the various types of attacks. [C. Curtsinger et al., 2011] proposed Zozzle that is used to detect malware in a web browser.

III. METHODOLOGY

The basic aim of the study is to perform the classification of the malicious pages. In order to perform this kind of classification, we used a supervised machine learning approaches that evaluate the feature of the extracted web pages. The features extracted from a web page are helpful to decide whether the web pages are malicious page or not. We inspect two main sources viz. HTML page and JavaScript code for the extraction of the features. We notice that most of the JavaScript are obfuscated and therefore, becomes difficult or the analysis. In order to detect these characteristics, we implemented the extraction of some statistical measures viz. string entropy, whitespace percentage, and average line length. We also consider the structure of the Java-Script code itself, and a number of features are based on the analysis of the Abstract Syntax Tree (AST) extracted using the parser. For example, we analyze the AST of the code to compute the ratio between keywords and words, to identify common decryption schemes, and to calculate the occurrences of certain classes of function calls (such as fromCharCode(), eval(), and some string functions) that are commonly used for the decryption and execution of drive-by-download exploits.

We extract a total of 25 features from each piece of JavaScript code viz. the number of occurrences of the eval() function; the number of occurrences of the setTimeout() and setInterval() functions; the ratio between keywords and words; the number of built-in functions commonly used for deobfuscation; the number of pieces of code resembling a deobfuscation routine; the entropy of the strings declared in the script; the entropy of the script as a whole; the number of long strings; the maximum entropy of all the script's strings; the probability of the script to contain shellcode; the maximum length of the script's strings; the number of long variable or function names used in the code; the number of string direct assignments; the number of string modification functions; the number of event attachments; the number of fingerprinting functions; the number of suspicious objects used in the script; the number of suspicious strings; the number of DOM modification functions; the script's whitespace percentage; the average length of the strings used in the script; the average script line length; the number of strings containing "iframe" ; the number of strings containing the name of tags that can be used for malicious purposes, and the length of the script in characters.

Identifying malicious activity in web pages requires a detection system to monitor the execution of JavaScript code at run-time. The flow of the execution is tracked using events that indicate changes in the state of the environment. Depending on the granularity of the monitoring, these events may range from calls to certain JavaScript functions to the observation of every state-changing action.

All statements S in javascript code can be added in to event list. If S is assignment operation then that will be added into event as SET variable Name To value in Events list , if S is function calladd event as FUNCTIONCALL name with its parameter to Event list, If S is constructor add event as CONSTRUCTOR name with its parameter to events list. Otherwise, added into event list.

```
1 a=new Array("AML", "Ioo", "exe")
2 try {
3    o=new ActiveXObject("MS2"+a[0]+"."+a[0]+"HTTP")
4    o.open("GET","http://"+a[1]+".com/x." + a[2],true);
5 } catch(e) {};
```

Figure 1 Obfuscated JavaScript code

```
1  SET CUSTOM_OBJECT_22.0 TO "XML"
2  SET CUSTOM_OBJECT_22.1 TO "foo"
3  SET CUSTOM_OBJECT_22.2 TO "exe"
4  SET global.a TO CUSTOM_OBJECT_22
5  CONVERT ActiveXObject TO A FUNCTION
6  CONSTRUCTOR ON CUSTOM_OBJECT_24 CALLED
7  SET global.o TO NEW_OBJECT_FROM_CONSTRUCTOR
8  CONVERT NEW_OBJECT_FROM_CONSTRUCTOR TO A OBJECT
9  CALL open
10 CONVERT NEW_OBJECT_FROM_CONSTRUCTOR.open TO A FUNCTION
11 FUNCTIONCALL open ("GET", "http://foo.com/x.exe",
12                    "BOOLEAN PRIMITIVE true")
```

Figure 2 Monitored events

In figure 3.1, we shows obfuscated javascript code and in figure 3.2, we shows monitored events of that code. The detector supports five basic types of events, where each type is recorded with respective arguments during the execution. For example, line 3 in Figure 2 shows a SET event that assigns the string "exe" to an internal object. The code snippet contains a trivial form of obfuscation that hides the download of an executable file. After a series of different events, this hidden download is revealed in the FUNCTIONCALL event at lines 11–12 of Figure 2. Sequences are a natural representation of behavior, yet they are not directly suitable for the application of learning methods, as these usually operate on vectorial data. So that we can generate events to vector. If each event e in Events list exists in database DB, get id from DB for e, otherwise add e to DB and assign new Id. In addition, id added into vector.

A. SVM Training and Classification

a. Support Vector Machine

For automatically generating detection models from the Reports of attacks and benign JavaScript code, apply the Technique of Support Vector Machines Given vectors of two classes as training data, an SVM

determines a hyper plane that separates both classes with maximum margin. In our setting, one of these classes is associated with analysis reports of drive-by downloads, where as the other class corresponds to reports of benign WebPages. An unknown report φ(x) is now classified by mapping it to (x) drive-by downloads. Figure 3.3 Schematic vector representation of analysis reports with maximum-margin hyper plane.

b. JavaScript Extraction

As first analysis step, they aim at efficiently getting a comprehensive view on JavaScript code. To this end, inspect all HTML and XML documents passing the system for occurrences of JavaScript. For each requested document, extract all code blocks embedded using the HTML tag script and contained in HTML event handlers, such as on load and on mouse over. Moreover, recursively preload all external code referenced in the document, including scripts, frames and iframes, to obtain the complete code base of the web page. All code blocks of a requested document are merged for further static and dynamic analysis.

c. Obfuscated JavaScript

Obfuscation is different from minification, which removes the comments and unnecessary whites-pace from a program" to reduce the code size. Both benign and malicious JavaScript code has been observed adopting obfuscation techniques; hence, obfuscation does not imply maliciousness. However, their purposes of Obfuscation are different. Benign JavaScript code mainly Leverages obfuscation to protect code. This purpose requires obfuscated code to be Human unreadable and without down grading the execution performance. Normally, execution performance is not a concern for attackers. In fact, attackers often apply multiple obfuscation to hide the malicious intent.

d. Analysis Approach

Static analysis of JavaScript detection is used to detect the standard JS abnormality detection. It will detect the DOM changes to the web page layout. It is usually performed using IFrames in the page. The IFrames manipulated through JS. Before the source code of a program can be interpreted or compiled, it needs to be decomposed into lexical tokens. The static analysis component in Cujo takes efficiently extracts lexical tokens from the JavaScript code of a web page using a Yacc grammar. The lexical analysis closely follows the language specification of JavaScript. As the actual names of identifier do not contribute to the structure of code, replace them by the generic token ID. Similarly, they encode numerical iterals by NUM and string literals by STR. The dynamic JavaScript analysis is the core of system to detect malicious websites. The main advantage of dynamic analysis is that they are able to analyse obfuscated JavaScript, too. This is very important, since most JavaScript based exploits currently observed in the wild try to hide their presence using several obfuscation techniques. Usually obfuscation in JavaScript is reached through escaping or encoding the actual script. This code is then unescaped or decoded and executed by the JavaScript eval function. This procedure is often one several times recursively and thus it is quite some work to understand what the JavaScript actually does. Nevertheless, it is usually even impossible to automatically analyze a JavaScript. Additionally, it must to be easier to detect malicious JavaScript based on its behavior than on its source code.

IV. EMPIRICAL EVALUATION

After discussing the rather technical details of our method, we proceed to present an empirical evaluation using real JavaScript code of malicious and benign web site. Besides studying the overall detection performance of our system, we focus on experiments concerning the performance over time. Furthermore, we examine the robustness against simple evasion attacks and provide exemplary explanation for the earlier detection compared to the regular SVM.

A. Evaluation Data

As a data set of (mostly) benign JavaScript code, we consider the 100 most visited web sites according to the Alexa ranking1. Each of these web sites is visited

automatically and its JavaScript code is executed using the dynamic analysis implemented in the Cujo detector. While we can not rule out the presence of some malicious behavior in this set, our experiments do not indicate any influence from such behavior on the final results. Table 1 lists the data sets of malicious JavaScript code used in our experiments together with their origin and size. These attacks have already been used to evaluate Cujo. Malware Forum, SQL Injection and Alexa are taken from Cova et al. [4], whereas the Obfuscated set consists of 84 additionally generated obfuscated attacks from the other sets [see 1].

B. Experimental Discussions

This section describes experiments performed to evaluate Our System's performance and detection effectiveness. Furthermore, we describe the insights on prevalent web-attacks that we gained during our analysis of web pages and we present an in-depth analysis of one of the malicious web pages. The experiments were performed on a 2.4 GHz Intel Core-2-Quad test system with 8 GB of RAM running Debian Linux "Lenny" connected to the internet with 2 MBit/s DSL. For comparative evaluation, the high-interaction honey client Capture HPC [7] version 2.51 was installed on the test system running on a Windows XP SP2 client with Internet Explorer 6 using the default configuration. To emphasize the need for an early detection of malicious activity, Figure 4 presents graph of the number of monitored events for some no. of links. We observe that there are short and long sequences for both malicious and benign web sites with up to 106 events. Clearly, there is potential to reduce the ratio of executed malicious code and limit possible damage with our approach to early detection. During the examination, 100 web pages have been checked. The candidate list was created by querying Google's search engine with promising search terms and URLs that were reported by users. Our System found 56 malicious web pages equaling at a rate of about 5.6% and 19.317 inline frames invisible to the user pointing to malware distribution pages. During the study, we found that the system saved approximately 3 GB of HTML, JavaScript (obfuscated and deobfuscated) and binaries including 2.114 unique (disffering MD5 values) malicious executable samples. All files generated by Our System were scanned utilizing the G Data Linux antivirus engine. The scanner marked 43.175 files as malicious. The bulk of the antivirus detections were triggered by files that were deobfuscated by the Our System. Therefore, a HTTP scanner as utilized by many common antivirus solutions would not have detected these attacks, since the attacks are dynamically decrypted in the browser. A noticeable amount of detections were triggered by signatures not targeting web-based exploit code but inline frames pointing to known (blacklisted) malware distribution domains. Several large-scale attacks were identified using the result database of Our System. Thereby several thousand infected pages were linked to the malware distribution servers used.

Table 1 Improved results of Proposed System compared with Existing System

Web Link	Proposed Time	Existing Time
andtsgame.com	11182	18534
ceskarepublika.net.html	12424	14876
cracks.vg.html	26976	39575
crackspider.us.html	8029	11239
Internet Security And Computer Maintenance!.html	7862	9237
m2132.ehgaugysd.net.html	55078	74008
nerez-schodiste-zabradli.com	14889	18109
Amazon.co.uk_ Low Prices in Electronics, Books, Sports Equipment & more.html	72242	77886
Ask.com - What's Your Question_.html	51695	97523
baidu.html	71695	97523
China Daily Website - Connecting China Connecting the World.html	20082	24310
China.com - Your guide on traveling and living in China.html	8454	10632
CNN - Breaking News, U.S., World, Weather, Entertainment & Video News.html	40809	47327
sogou.html	13282	16339

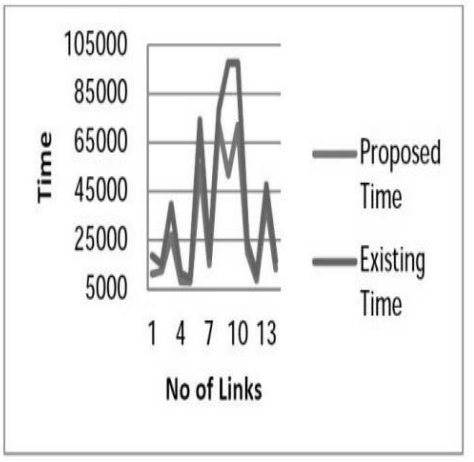

Figure 3 An Experimental Analysis

VII. CONCLUSION

In this paper, we have discussed different malicious detection strategies. We have carried out comparison and analysis between different detection techniques. Detection techniques have been improved dramatically over time, especially in the past few years. Developing new malicious detection schemes is necessary because attackers develop their strategies continuously too. Therefore, there is a flexible detection method for early identification of malicious JavaScript behavior. For this, method uses machine learning techniques for optimizing the accuracy as well as the time of detection.

REFERENCES

[1] K. Rieck, T. Krueger, and A. Dewald. Cujo: Efficient detection and prevention of drive-by download attacks. In 26th Annual Computer Security Applications Conference (ACSAC), pages 31-39, Dec. 2010.

[2] L. Lu, V. Yegneswaran, P. A. Porras, and W. Lee. BLADE: An attack-agnostic approach for preventing drive-by malware infections. In Proc. of Conference on Computer and Communications Security (CCS), pages 440-450,Oct. 2010.

[3] D. Canali, M. Cova, G. Vigna,and C. Kruegel. Prophiler: a fast Filter for the large-scale detection of malicious web pages. In Proc. of the International World Wide Web Conference (WWW), pages 197- 206, Apr.2011.

[4] M. Cova, C. Kruegel, and G. Vigna. Detection and analysis of drive-by-download attacks and malicious JavaScript code. In Proc. of the International World Wide Web Conference (WWW), 2010.

[5] M. Heiderich, T. Frosch, and T. Holz. IceShield: Detection and mitigiation of malicious web sites with a frozen dom. In Recent Adances in Intrusion Detection (RAID), Sept. 2011.

[6] C. Curtsinger, B. Livshits, B. Zorn, and C. Seifert. Zozzle: Fast and precise in-browser javascript malware detection. In Proc. of USENIX Security Symposium, 2011.

[7] C. Seifert and R. Steenson. Capture – honeypot client (Capture-HPC). Victoria University of Wellington,NZ, https://projects.honeynet.org/capturehpc,2006