A SURVEY ON DYNAMIC IDENTIFICATION OF MALICIOUS OBfuscated JAVA SCRIPTS

VRUSHALI S. BARI, NITIN N. PATIL

1PG Student, Department of Computer Engineering, SES’s R. C. Patel Institute of Technology, Shirpur, Maharashtra, India.
2Department of Computer Engineering, SES’s R. C. Patel Institute of Technology, Shirpur, Maharashtra, India.
E-mail: vrushalibari@gmail.com, nitinpatil@rediffmail.com

Abstract: In recent scenario, the JavaScript has become one of the leading choices for attackers to implement their attacks and infecting users with malicious software. Static analysis fails to protect from this threat. In contrast, the dynamic analysis at run-time has proven to be effective. During the execution of the code, damage may already occur for this early detection is difficult for protection. In this paper, we survey early detection system which identifies malicious activities in JavaScripts. In which uses machine learning techniques for accuracy and the time of detection.

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

I. INTRODUCTION

During the last few years, we observed a shift in attacks against end-users: Instead of attacking network services, many of today’s attacks focus in client applications. For attackers the web browser is a popular target. There are many different kinds of threats and attack vectors against Current browsers. With the rapid development of network information technology, information security issues gains more and more attentions. The malicious script is one of the primary security threats of computer networks. By constructing a special web page, in which Trojans, viruses, worms, or aggressive programs, malicious script propagate to the user’s computer when the user access to these pages. Most web based attacks take place on legitimate websites. The most common of all malware threats is SQL injection attacks against websites. Through HTML and URLs, the Web was vulnerable to attacks like cross-site scripting (XSS) that came with the introduction of JavaScript. JavaScript is an interpreted programming or script language from Netscape. It is most often used for client side web development. It is a dynamic, weakly typed, prototype based language. JavaScript is generally used in website development to change a formatted date on a web page automatically, cause a linked page to appear in a pop up window or to add some rollover effects.

In contrast to other types of network attacks, malicious JavaScript code is particularly hard to detect in the content of web pages. As the code is directly interpreted during the visit of a web page, the attacker is able to literally program his attack. For example, JavaScript attacks regular analysis of the browser environment, check for particular vulnerabilities and use dynamic exploiting techniques, such as heap spraying, for compromising a victim’s system. Even worse, the direct execution of the code also enables effectively obfuscating the attack, such that indicative patterns are only visible at run-time and not accessed by static detection methods, such as conventional anti-virus scanners. As a result of this development, the detection of malicious JavaScript code at run-time has gained a focus on security research. Several methods have been developed for dynamically spotting malicious activity, which range from specific detection systems. Several detection systems have been proposed that apply learning methods to automatically distinguish benign from malicious behavior, for example, the detectors Cujo [7], Zozzle [4] and IceShield [5]. While these learning-based detectors provide an accurate identification of malicious code at run-time, none of the detectors are optimized for an early detection of attacks. The longer a malicious code runs before it is detected, the more harm it can cause to the underlying system. Consequently, the aspired protection can be significantly weakened, if malicious behavior is identified too late. However, spotting attacks early is not a trivial task, as it imposes two challenges on a detector’s design: First, malicious behavior should be detected as fast as possible, but never at the price 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.

Figure 1.1 Architecture of the System
II. BACKGROUND

2.1 Attack Types

[1] Drive-by download attacks in which a vulnerability in the web browser or one of its components/extensions (e.g., Acrobat Reader or Flash plugins) is exploited to execute code of the attacker’s choice.


2.2 Client Side Attacks through Browser

Client side attack exploits the vulnerability in client application running on client agent browser such as Mozilla firebox, chrome, internet explorer etc.. The following are the attacks that may be exploited by hackers through the browser.

2.2.1 Attacks against Plug-ins within the Web browser

Plug-in is a set of software components that adds specific abilities, customize the functionality of the browser, like play video, display new file types. It also changes the browser settings, block the client request and download malware to the user’s system.

2.2.2 Clickjacking

Clickjacking is a malicious technique that can force the user to adjust the user’s system settings without the user awareness. By using JavaScript, an attacker could place a fake button or link under or over a genuine button or link.

2.2.3 Phishing

The Phishing is attempting to acquire information such as usernames, passwords, and confidential details by impersonating as a reliable web page.

2.2.4 Cookie Stealing

Cookies are raw data which are sent by the server and stored on the client system for later retrieval to allow user-side customization of web information. The information in the cookies is easily accessible by attacking, with the aid of these cookies the attackers can steal sessions, and compromise accounts.

2.2.5 Stealing Information from Storage

The browser cache and browser history are also precious portions of information that attackers can gain access.

2.3 Different Detection Systems

[1] Cujo

Konrad Rieck et al. has presented Cujo, a system for effective and efficient prevention of drive-by downloads. As an extension to a web proxy, Cujo transparently inspects web pages using static and dynamic detection models and allows for blocking malicious code prior to delivery to the client. In an experimental evaluation with 200,000 web pages and 600 drive-by-download attacks, a prototype of this system significantly outperforms current anti-virus products and enables detecting 94% of the drive-by downloads with few false alarms and a median run-time of 500ms per web page a delay hardly perceived at the webclient[7].

[2] Prophiler

Marco Cova et al. Has proposed malware on the Internet spreads and becomes more sophisticated, anti-malware techniques need to be improved in order to be able to identify new threats in an efficient, and, most importantly, automatic way. Prophiler, a system whose aim is to provide a filter that can reduce the number of web pages that need to be analyzed dynamically to identify malicious web pages. We have deployed our system as a front end for Wepawet, a well-known, publicly-available dynamic analysis tool for web malware. The results show that Prophiler is able to dramatically reduce the load of the Wepawet system with a very small false negative rate[1].

[3] Blade

L. Lu et al. proposed BLADE, a system that effectively immunizes a host against all forms of drive-by download malware installs. A distinguishing aspect of the BLADE design is that it is both attack and browser agnostic, i.e., it neither requires exploit signatures nor changes to the browsers. Rather, BLADE relies on limited semantic knowledge about a handful of user interface (UI) elements common across web browser applications[8].

[4] IceShield

M. Heiderich et al. proposed IceShield, a novel approach to perform light-weight instrumentation of JavaScript, detecting a diverse set of attacks against the DOM tree, and protecting users against such attacks. The instrumentation is light-weight in the sense that IceShield runs directly within the context of the browser, as it is implemented solely in JavaScript[5].

[5] Zozzle

B. Zorn et al. proposed ZOZZLE is a mostly static JavaScript malware detector that is fast enough to be used in a browser. While its analysis is entirely static, ZOZZLE has a runtime component: to address the issue of JavaScript obfuscation, ZOZZLE is integrated with the browser’s JavaScript engine to collect and process JavaScript code that is created at runtime. Note that fully static analysis is difficult because JavaScript code obfuscation and runtime code generation are so common in both benign and malicious code [4].

III. BASIC TERMS

3.1 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 hyperplane that separates both classes with maximum margin. In our setting, one of these classes is associated with analysis reports of drive-by downloads, whereas the other class corresponds to reports of benign webpages. An unknown report \( \phi(x) \) is now classified by mapping it to \( (x) \) drive-by downloads.

\[
\phi(x) \rightarrow \text{drive-by downloads} \\
\text{maximum margin} \\
\text{benign code}
\]

Figure 3.1 Schematic vector representation of the analysis reports with a maximum -margin hyperplane.

3.2 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 then merged for further static and dynamic analysis.

3.3 Obfuscated JavaScript

Obfuscation is different from minification, which “removes the comments and unnecessary whitespace from a program” to reduce the code size. Both benign and malicious JavaScript code have 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 the code. This purpose requires obfuscated code to be Human unreadable and without downgrading the execution performance. Normally, execution performance is not a concern for attackers. In fact, attackers often apply multiple obfuscation to better hide their malicious intent.

3.4 Analytical Approach

3.4.1 Static Analysis

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 identifiers do not contribute to the structure of the code, replace them by the generic token ID. Similarly, they encode numerical literals by the NUM and string literals by STR.

3.4.2 Dynamic Analysis

The dynamic JavaScript analysis is the core of a system to detect malicious websites. The main advantage of dynamic analysis is that they are able to analyze 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 decoded and executed by the JavaScript eval function. This procedure is offend one several times recursively and thus it is quite some work to understand what the JavaScript actually does. But it is usually even impossible to automatically analyze such a JavaScript. Additionally, it must be easier to detect malicious JavaScript based on its behavior than on its source code.

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. So there is a flexible detection method for early identification of malicious JavaScript behavior. For this, the method uses machine learning techniques for optimizing the accuracy as well as the time of detection.

ACKNOWLEDGMENT

I express my sincere thanks to my guide Prof. Nitin N. Patil, the faith & confidence shown by him in me, boosted me and motivated me to perform better.

REFERENCES

A Survey on Dynamic Identification of Malicious Obfuscated Java Scripts


★★★