International Journal of Modern Engineering Research (IJMER)
www.ijmer.com Vol.3, Issue.2, March-April. 2013 pp-1019-1021 ISSN: 2249-6645

Runtime Solution for Minimizing Drive-By-Download Attacks

Pratik Upadhya, [1] Farhan Meer, [2] Acquin Dmello, [3] Nikita Dmello[4]
[123]*Department of Computer Engineering, St. Francis Institute of Technology, Mumbai, India*
[4]*Department of Computer Engineering, Fr. Conceicao Rodrigues College of Engineering, Mumbai, India*

Abstract: *The ever growing use of internet has given rise to a lot of security vulnerabilities to web application users. One of the most dangerous among these threats is drive-by-download attacks which target those victims with some vulnerability in their browser or browser plug-ins. Drive-by-download attack allows the attacker to download a code on the victim's host and execute it on victim's host without any prior knowledge to the victim. Due to this, the attacker may get full control of the victim's host. We present a runtime; Behavior based solution, to protect a Browser against Drive-by-download attacks. This paper extends the concept of Behavior based solution to the domain of Internet Browsers, which protects a browser against drive-by-download attacks. Our approach will enhance the security of a browser, thereby avoiding attackers to penetrate a victim's host.*

Keywords: *behavior-based solution, browser, intrusion detection, security.*

I. INTRODUCTION

Web developers recognize the importance of Cyber security for safe browsing on internet. Through this paper, we propose a Runtime, behavior-based solution which is called Browser Guard, against drive-by-download attacks which is one of the most dangerous threat for browsers these days. Malicious web content has become the primary instrument used by miscreants to perform their attacks on the Internet. In particular, attacks that target web clients, as opposed to infrastructure components, have become pervasive [2]. It is observed that, around 45% of the internet users use an outdated browser [3]. Recently, several approaches to protect browsers from drive-by-download attacks have been proposed [1, 4, 5, 6] which addresses protection for browsers against drive-by-download attacks. However many of them take the trouble by false positive, false negative or performance overhead and some are limited to detection of drive-by-attacks of JavaScript [4]. This seriously impacts the usability of web applications.

This paper proposes a runtime, Browser-based solution called as Browser Gaurd, which prevents the drive-by-download attack. Browser Gaurd is the tool being used to monitor the download scenario of all the files that are loaded into the host through a browser. After the inspection of the download scenario, Browser Gaurd restricts the execution of any file that is automatically downloaded into the host without any prior knowledge of the user. As this approach depends on the Behavior-based detection, Browser Gaurd does not need to analyze the source code file of any web application or the run-time states of any script code. Browser Gaurd also does not need to maintain any exploit code samples to compare with malicious web pages and does not need to query the reputation value of any website.

In summary, the contribution of this paper is a runtime, browser-based solution for detecting, analyzing and mitigating drive-by-download attacks.

II. RELATED WORK

A number of approaches and tools have been proposed in recent years to identify and analyze malicious code on the web. We will now briefly present the most relevant ones and compare them with our approach.

Patch Exe: Many drive-by-download attacks are triggered by vulnerable ActiveX controls. Microsoft uses Kill-Bit [7] to mitigate this problem. However, Kill-Bit does not patch any executable. Instead, it just blocks the use of certain known vulnerable ActiveX Controls. If a particular ActiveX Control is marked as unsafe through Kill-Bit, the ActiveX Control will never be invoked by any application. However, attackers can utilize non- ActiveX Control vulnerabilities to launch a drive-by-download attack and not all vulnerabilities of all ActiveX controls are unveiled.

NOP sleds and Shell code: Nozzle [8] detects heap spray attacks based on the observation that shell code used in a heap spray attack is usually prepended with a long NOP sled. Experimental results showed that Nozzle has small number of false positives and false negatives. However, if an attacker writes NOP sleds and shell code into a heap string after Nozzle finishes its examination, Nozzle is not able to detect the attack.

Performance issues to non-browser processes: Blade adopt similar philosophy of blocking the execution of suspicious executable files as Browser Guard does. By sandboxing all downloaded objects in a secure zone, their work, Blade, prohibits supervised process from operating unauthorized files in the secure zone. Blade captures user behaviors, such as clicking a mouse button on a download related popup window, to mark user-consent downloaded files as authorized. Blade has zero error rate. However, it may pose performance issues to non-browser processes due to the special secure zone design that once a user tries to manipulate a file, OS should check whether this file is in the secure zone.

Non-trivial False positives of Browser Reputation System: In this system, before displaying a web page on a browser, the browser automatically connects to a remote database to check the reputation of the web page first. Only web pages with good reputation can display on the browser. Various antivirus vendors adopted this approach to deal with drive-by-download attacks. However, the browser reputation system has no guarantee that all websites are under their monitor. Besides, they have non-trivial false positives and it takes a while to update out-of-date or wrong data in the database or to add new data to the database.

Inter-Module Communications: IMC monitoring [5] detects drive-by download attacks by matching the inter-module communication events with predefined vulnerability signatures. However, its signature-based property makes it difficult to detect zero-day attacks.

III. LIMITATIONS

A behavior based solution is not acceptable in case of shadow attacks [10]. Specifically, shadow attacks create shadow process communication (SPC) channels between the rewritten malware and its shadow processes to achieve the original malicious functionalities. As of writing of this Paper, most behavior-based malware detectors are designed based on malicious specifications in terms of system call sequences/graphs of individual single-process program (or these with simple inheritance/fork relationships). It is worth noting that the practically used system call sequence/graph behaviors are rarely just a single system call because that will have high false positive rates as likely many normal programs could use the same single system call with similar parameters as a malware does. Therefore, these behavior-based malware detectors could hardly detect shadow processes because they only contain small segments (e.g., just one system call) of the malicious behavior of malware.

Most existing scanners use file or network flow as scanning granularity [5]. However, to help web application developers, most browsers have the ability to let scripts include other scripts. By leveraging this feature, attackers can split one exploit script into several files without affecting the correct functionality. As a result, scanners without a mature reassemble mechanism will be bypassed. By using behavior-based detection mechanism, the unpacking and reassembling problem stated above could be avoided. But the situation for existing behavior-based protection systems is no better. Firewalls can block illegal traffic, yet from the perspective of firewall, since drive-by download attack is almost the same as legitimate visiting web pages, it has no reason to block this kind of traffic. Even though black lists (e.g. Google Safe Browsing API [11]) can be used to prevent visiting malicious servers, these lists trend to be incomplete and sometimes outdated.

IV. IMPLEMENTATION

This section of the paper specifies the implementation details, design goal and design principle of BrowserGaurd. There are three components involved in the file download operation and the file execution operation in IE [1,7]. There are file download component, file execution component, and event component. According to the file download steps of a browser, Browser Guard sets several *check points* on a browser and the Windows kernel to detect secret download and blocks the execution of downloaded malware at runtime. The structure of Browser Gaurd is not sophisticated. It consists of a Browser Guard Browser Helper Objet in every IE process, a Browser Guard Kernel in the kernel space, and a *list server process*. Each host has only one list server process. But the host may have several browsers executing simultaneously; hence, there may exist multiple Browser Guard-Browser Helper Objects in a host at the same time. A Browser Guard Browser Helper Objet communicates with the list server process through a named pipe. Multiple Browser Guard-Browser Helper Objects can communicate with the list server process simultaneously. The list server process contains two lists, a white-list and a blacklist. The white-list records the URLs of trusted files and the hash vales of trusted executable files. The blacklist records the hash values of detected malicious files.

Browser Guard-Kernel is a kernel component of Browser-Guard. Browser Guard-Kernel enforces the following two tasks to prevent the execution of malware and illegal modifications of a white-list and blacklist. First, Browser Guard-Kernel ensures that the execution of a program is issued by an internal system command which has been hooked by Browser Guard. Second, Browser Guard-Kernel denies a request to modify a white-list, if the request is not issued through the code in functions of Browser Guard.

4.1 Browser Guard Workflow

Browser Guard provides its protection to a host through a two-phase mechanism and a kernel component. In the first phase, namely the filtration phase, Browser Guard distinguishes malicious files from trusted ones based on the situations under which the files are downloaded to a local host. In the second phase, namely the prohibition phase, Browser Guard denies the request to execute malicious files. The kernel component blocks attempts to bypass Browser Guard.

4.1.1Filtration Phase

To be able to distinguish malicious files from trusted ones, Browser Guard needs to know the situation under which a file is downloaded to a local host. With the information, Browser Guard can deduce whether a downloaded file is a trusted one or malicious one. In order to collect the required information, Browser Guard installs several check points to monitor the behavior of a browser.

While a user is surfing the WWW, a browser needs to download various files. All these files are placed in a directory called Temporary Internet Files and they cannot be directly executed without the permission of the browser user. On a Browser

Guard protected browser, normal file download triggers the execution of download functions. These functions connect to the list server process of a host to record URLs in the white-list of the process. The URLs are the URLs of the files that are going to be downloaded to the host. Then other functions checks whether the URL of the file that this function is asked to download is within the white-list. If the URL is within the white-list, the file is downloaded as usual; but if the URL is not within the white-list, Browser Guard calculates the hash value of the file and adds the hash value to the black list.

Instead of using file extensions to find executable files, Browser Gaurd uses hash value to find executable files. This can prevent an attacker from naming an executable file with a non-executable filename extension first and then changing its filename extension back to an executable filename extension before executing the file. The hash value is calculated based on the first 512 bytes of a file. Hence, instead of comparing all the bits of a file to check if it's malicious, Browser Gaurd just compares the first 512 bytes of a file.

4.1.2 Prohibition phase

A system function resides in IE browser to execute executable file on the disk. Browser Guard hooks this API to ensure that the API will not execute malware. Browser Guard calculates the hash value of the executable file first. Then Browser Guard checks whether the white-list and blacklist contain the same hash value. If the blacklist does not contain the hash value but the white-list contains the hash value, Browser Gaurd runs the executable file. Otherwise, it blocks the execution of the file.

4.1.3Avoiding Checkpoint-Bypassing

Various checkpoints installed by Browser Guard are the critical instructions used to detect downloaded malware and prevent the execution of the downloaded malware. If an attacker can bypass these checkpoints, she/he can successfully accomplish a drive-by-download attack on a Browser Guard protected browser. Browser Guard utilizes various approaches to ensure that, if the download and execution of a program do not follow the normal path and does not pass the predefined checkpoints, Browser Guard can detect it and block the execution.

V. CONCLUSION AND FUTURE WORK

As the web browser advances to an integral component in every day computing, it becomes an attractive target for attackers. Outdated browsers and plug-ins allow attackers to infect computer systems via drive-by attacks that download and install malicious software upon the mere visit of a web page containing malicious content. As drive-by attacks target the browser and its components directly, we propose to have defensive mechanisms built into the browser itself to mitigate the threats that arise from drive-by download attacks. Therefore, the paper proposes and elaborates a runtime browser-based strategy that bares the potential to protect users from these threats. Implementing countermeasures in the browser, to some extent, also allows for the protection of otherwise vulnerable components. Therefore, users would benefit directly and immediately from the security enhancements that browser built-in protection mechanisms would provide. In this paper, we present Browser Guard, a runtime, behavior-based solution to drive-by-download attacks. Browser Guard analyzes the download scenario of every downloaded object. Based on the download scenario, Browser Guard blocks the execution of any executable file that is downloaded to the host machine without the consent of a user.

In future, we try to work on the limitations and to make it compatible to more number of browsers across multiple platforms and also improve the usability of our technique.

References

[1] Fu-Hau Hsu, Chang-Kuo Tso, Yi-Chun Yeh, Wei-Jen Wang, and Li-Han Chen, Browser Guard: A Behavior-Based Solution to Drive-by-Download Attacks, IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, VOL. 29, NO. 7, AUGUST 2011.

[2] N. Provos, P. Mavrommatis, M. Rajab, and F. Monrose, All Your iFRAMEs Point to Us, In Proceedings of the USENIX Security Symposium, 2008.

[3] S. Frei, T. Dübendorfer, G. Ollman, and M. May, Understanding the Web browser threat: Examination of vulnerable online Web browser populations and the "insecurity iceberg", In Proceedings of DefCon 16, 2008.

[4] Marco Cova, Christopher Kruegel, and Giovanni Vigna, Detection and Analysis of Drive-by-Download Attacks and Malicious JavaScript Code, International World Wide Web Conference Committee (IW3C2), April 26–30, 2010.

[5] Chengyu Song, Jianwei Zhuge, Xinhui Han, Zhiyuan Ye, Preventing Drive-by Download via Inter-Module Communication Monitoring, ASIACCS'10 April 13–16, 2010.

[6] Konrad Rieck, Tammo Krueger, Andreas Dewald , Cujo: Efficient Detection and Prevention of Drive-by-Download Attacks, ACSAC '10 Dec. 6-10, 2010.

[7] Microsoft Security Research & Defense, [Online], Available: http://blogs.technet.com/srd/archive/2008/02/06/The-Kill 2D00 Bit-FAQ 3A00 -Part-1-of-3.aspx

[8] P. Ratanaworabhan, B. Livshits, and B. Zorn, NOZZLE: a defense against heap-spraying code injection attacks, USENIX Association, 2009, pp. 169–186.

[9] L. Lu, V. Yegneswaran, P. Porras, and W. Lee, BLADE: an attack agnostic approach for preventing drive-by malware infections, 17th ACM conference on Computer and communications security, ser. CCS '10. ACM, 2010, pp. 440–450.

[10] Weiqin Ma, Pu Duan, Sanmin Liu, Guofei Gu and Jyh-Charn Liu, Shadow Attacks: Automatically Evading System-Call-Behavior Based Malware Detection, Department of Computer Science and Engineering, Texas A&M University College Station, TX, USA 77843-3112

[11] G. Inc. Google safe browsing api. http://code.google.com/apis/safebrowsing/.