International Journal of Engineering & Technology, *8 (1.9) (2019) 364-368*

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Drive by Download Interceptor using Abstract Syntax Tree

Patrick Adolf Telnoni[1], Muhammad Barja Sanjaya[2]

[1,2]Department of Informatics Management, School of Applied Science, Universitas Telkom, Bandung 40257, Indonesia

Abstract

Rapid increase of malware attacks cause many malware propagation techniques to arise. One most common techniques is using web browser. Drive by download attacks is the highest attack in web browser used by malware to propagate itself. Drive by download attack download malicious program by executing script -commonly javascript- without user's consent. Drive by script varies from simple one to the hard one. The code become harder to detect when the malware developer craft the code using obfuscation technique. Obfuscated code often slip away from detection and requires high computing process to recognize its original form. This paper will present design of tools to intercept drive by download attack without using excessive computation resource when it comes to obfuscated code by using Abstract Syntax Tree to detect code duplication and de-obfustication technique using DeFusinator plugin. As far this research went, we able to extract AST from a benign javascript, but we also find obstacle to find active malicious page to be extracted. This research will provide tools to prevent user from visiting malicious web pages infected with drive by script.

Keywords: *Product design, malware, drive by download, code obfuscation*

1. Introduction

There are many method used by malware to propagate itself. One of the most common method is drive by download. Drive by download started when a user visits website contain malicious script. When the page is loaded, web browser executes, redirects browser to unintended URL and downloads malicious program or execute shellcode. This shellcode or malware will exploit vulnerabilities in user's computer and could give devastating impact.

Priya state that drive by download is the most occurred in many websites [1]. In 2013, security-based industry even suffer damage from drive by download attack. 61% of common website have been infected and can be classified as malicious. Cisco published that from listed malicious website, 33% of them are hosted in America [2].

In this paper, we propose design of tools for detecting malicious page using Abstract Syntax Tree and combine it with deobfuscator called DeFusinator so that our tools will not foiled by obfuscated javascript.

We organize the rest of this paper in the following way, section 3 discuss literature review, section 4 discuss rudimentary theory, section 5 give explanation about analysis and proposed solution, section 6 about progress.

2. Literature Review

This section is meant to give complete understanding about terms used in this paper. This section will explain definition and example about Drive By Download Attack, Abstract Syntax Tree, and Code Obfuscation.

2.1. Drive by download

Drive by download exploits vulnerability inside web browser or plugin by using injected code inside benign web browser. In Figure 1, drive by download attack is described.

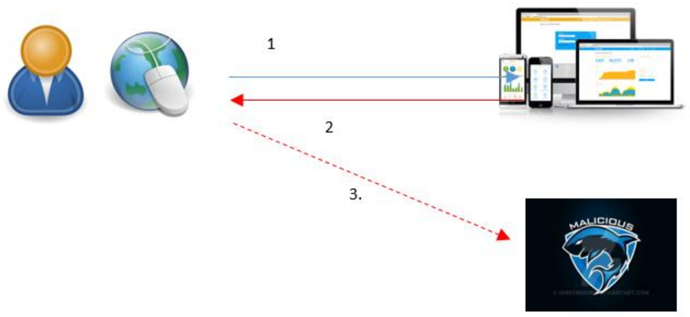

Fig 1: drive by download steps

At step (1), user access benign website which has been compromised with drive by download script. The server return the content of the website and rendered it user's browser. Web browser will run every single line of HTML and javascript fetched into its page (2). In this part, browser will execute drive by script, thus will download malicious file, commonly in .exe format (3). This contain malicious code that will run exploit in user's computer. Factors which cause drive by download are mentioned as vulnerabilities of web browser, well documented attack exploiting web browser and plenty of tools to exploit web browser vulnerabilities.

2.2. Abstract syntax tree

Abstract Syntax Tree (AST) is the form of source code of a file or program represented in tree. Each node in AST represents constructor in a programming language. Example to represents AST can be explained using simple mathematic

operation as follows:

$$5 * 3 + (4 + 2=2 * 8) \tag{1}$$

Equation (1) can be converted into AST as shown by figure 2.

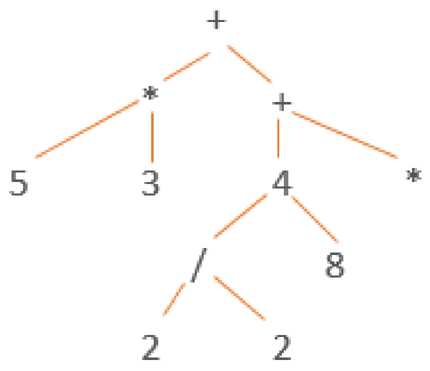

Fig 2: AST from equation (1)

AST itself is part of Syntax Tree. Syntax Tree is a part of compiler where it serves as semantic and syntax analyzer. Syntax analyzer guarantees "grammar structure" of written code. This process is similar to in sentence writing, where a sentence consists of subject, predicate, object and adverb. On the other hand, semantic analysis acts as vocabulary where it checks the writing of a syntax. Figure 3 describes process inside a compiler. The red dashed part is the part where syntax tree works.

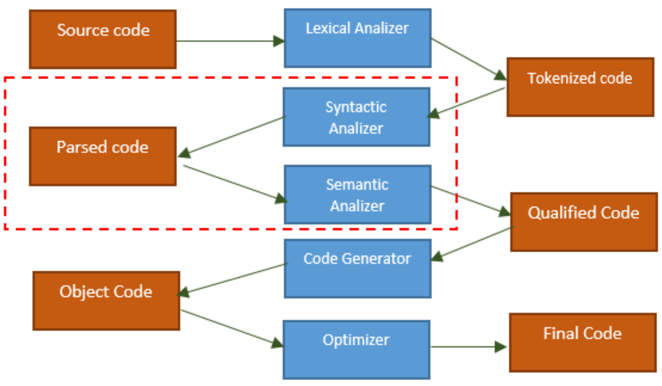

Fig 3: Workflow inside compiler

A few research was using AST to detect plagiarism, duplicate code, [6], code versioning [7] and malware analysis [3].

2.3. Code obfuscation

Code obfuscation is an evasive technique to hide original source code by dividing the original source into few chunks and assign the chunks into variable. Code obfuscation change very quickly as the developer try to hide the obfuscated code into the new one thus resulting into more complex code. Simple code obfuscation illustrated in table 1.

In table 1, a source code consists of 3 lines and 2 variables. Each variable assigned with a value, which is string. The code executed by concatenating value from each variable. The concatenated code print a Hello world using unescape command. This technique is used to obscure the shellcode, from simple form in figure 4, to the more complex form using complicated character such as “$”, “%”, “(”, and many more.

Table 1: Simple Obfuscated scipt

No	Code
1	Var a = “Hello”;
2	Var b = “World”;
3	Unenscape (a+b);

2.4. Related work

There are previous works to detect drive by download attack. Revolver uses honeypot to malicious web page [3]. The result from honeypot extracted into AST from and analyzed. The code which is converted into AST is not the whole web page code, but only javascript code. The extracted javascript code is not directly processed into recognition process, but will go through normalization process to reduce duplicated code. Revolver detects drive by download by running static method. The methods are to extract HTML web page and matched with the rules using WEKA tools. Extraction process run by executing URL in csv file. Extraction result will be chunked and stored into few files to ease reading and recognition process. Extraction result will pay attention to few rules:

1. <iframe> tag;

2. Shellcode look alike string or string with more than 40 characters length;

3. Shell executing function i.e. eval(), setTimeout(), InitiateFrameRequest(), setInterval();

4. Etc [3].

Sachin developed SurfGuard plugin to prevent drive by download attack [4]. The developed plugin run de-obfuscation process. Deobfuscation process run by using hooking technique. Hooking is interception technique to prevent code from being executed, in this case javascript. In interception process, code examined for malicious action. The malicious variable/parameter will be substituted using benign variable/parameter. The malicious terms for the code are described as follows:

1. Active X exploit;

2. String suspected as shellcode;

3. Redirect into suspicious URL;

4. Shellcode executing function such as decode() and unescape().

Priya used static analysis to detect drive by download attack on webpages [1]. In this research, webpages were analyzed using Support Vector Machine, Regression and K Nearest Neighbour (KNN). Detection rate, stated in success rate (%), gave result as stated :87.56% for SVM, 86.70% for regression and 91.36% for KNN;

From the related works, we come up with several analyses such as follows:

1. Revolver [3] requires relatively expensive infrastructure to run AST processing. This is normal because Revolver is designed to detect evasive code. Evasive code is the advanced and modified version of obfuscated code. Evasive code will be further developed into the more complicated one in hope for avoiding antivirus detection. Also, evasive code will require high computation load, considering the complex character and variable used to obscure shellcode.

2. SurfGuard [4] works as browser’s extension with light performance. Drawback of SurfGuard is that it only run simple analysis on native javascript, not the obfuscated one.

3. Priya uses static analysis, while it does not state how to overcome the obfuscated script.

3. Proposed Solution

From the analysis above, we will try to combine AST and Defusinator plugin. DeFusinator is a tool created to automatically de-obfuscate obscured code [21]. The purpose of this combination is to avoid heavy computing process on obfuscated script. Thus, this tool will The design of the tools is shown in Figure 4.

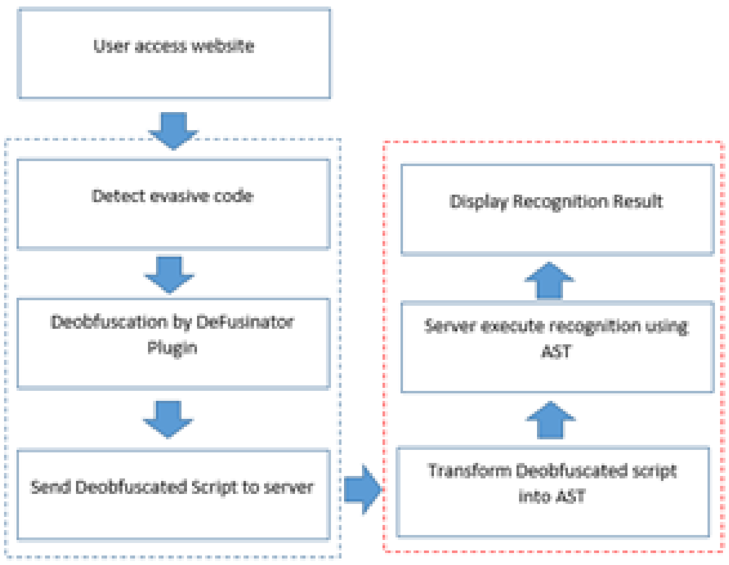
User access website
Detect evasive code
Deobfuscation by DeFusinator Plugin
Send Deobfuscated Script to server
Transform Deobfuscated script into AST
Server execute recognition using AST
Display Recognition Result

Fig 4: Workflow of the proposed solution

The designed tool has two main parts which work dependently. The blue part is the part where DeFusinator Plugin works, while the red part is the recognition process. According to Figure 4, the tools works as follows:

1) User visits web page;

2) Plugin searches from javascript code;

3) If javascript is found plugin will check for obfuscated code;

4) Plugin de-obfuscate the code and delivers original form;

5) Deobfuscated will be relayed to server as a string;

6) Server receives javascript code and converted into AST;

7) AST will be processed with method such as SVM, K Nearest Neighbour, Regression and Eucledian Distance;

8) Result from recognition process is presented into plugin and displayed.

Plugin will run hooking technique to prevent drive by download from executing. Plugins also run de-obfuscation process to avoid processing evasive code/obfuscation code so that it will not give server too much burden on computation process.

Server is used for recognition process to avoid putting unnecessary burden on client's machine. Running recognition process in user's PC will affect success rate considering user's workstation.

Since AST is a represented in tree, an effort to represent AST into a readable format were conducted. The tools itself is built with python, thus interchanging message is inevitable. Therefore, a universal format is required. Using universal format has been attempted from early millennial. A few examples were demonstrated by Holt [8], Mamas [9], and Microsoft [10] which are based on XML.

XML was designed to support may application, easy to create and many more [11]. Despite its rich feature, XML has its own drawback. The main drawback of XML is the performance regarded as heavy, therefore we choose JSON instead, which give lighter performance [12].

3.1. Dataset

As far we run this research, we often found problem with some obfuscated Javascript. Not every benign website is provided with obfuscated javascript. Our dataset itself use malicia dataset which contain 30,000 malicious websites. According Nappa, exploit server tend to have short lifespan varies from less than a day (16 hour) to several months [13]. It is implied that the exploit server could live up to several months, but the lifetime of the web page(s) itself does not guaranteed. The web page(s) may be modified into new web page(s).

Online malicious website also available in malware domain list [14], hphost online [15], suspicious comain from sans [16], zeltser [17], quttera[18], and malc0de[19]. Unfortunately, data in those websites are quite obsolete.

3.2. Progress

One of the main concern of this research is to avoid drive by download from executing. The safest way to access web page without executing javascript code is by command line like CUrl. CUrl is the most common way to access a URL without

using a web browser.

In this project, we use Python which provides AST library of the python syntax grammar [21]. Collecting dataset is done manually by scrapping website in malicia dataset, since python has many library for scrapping purpose. However, as we tried to scrap every 15813 available pages, but since the dataset is 3 years obsolete, hence most of the accessed link are unreachable.

Other obstacle we found during this research is that complex javascript cannot be fully represented into AST. This caused by different format in programming language. While in python, quotation mark is interpreted as comment, javascript interpret it as string or variable, thus causing error in scrapping process.

This research also still limited on scrapping webpages and the process is conducted separately from the de-obfuscation process. This means that this research is still limited with the javascript without commented line.

3.3. Future work

In response to the unreachable domain, we suggest attempting to find simple obfuscated script from ads-full website. Despite such website does not contain malicious software, the pattern to retrieve unwanted content – the ads – has the similar pattern as drive by attacks. For the experiment, we will use virtual environment.

4. Conclusion

From the proposed design and the progress so far, we can conclude that representing the drive by download has fast evolution and fast lifespan. This means that even though we may able to report its server location, the attacker can quickly arrange new exploit server with new address. Though DeFusinator Plugin has successfully decipher obfuscated script, this research still find difficulties with obsolete dataset and scrapping javascript. The challenge in interpreting javascipt into other programming language can render the scrapping process into failure. This research still has the potential to success, providing tools to prevent user from drive by attack with less complicated recognition process from obfuscated script.

References

[1] M. Priya, L. Sandhya, and C. Thomas, “A static approach to detect drive-by-download attacks on webpages,” in Control Communication and Computing (ICCC), 2013 International Conference on, Dec 2013, pp. 298–303.

[2] Cisco. (2013) Cisco 2013 annual security report. Website. Last checked: 24.11.2014. [Online]. Available: http://www.cisco.com/c/en/us/products/security/annual security report.html

[3] A. Kapravelos, Y. Shoshitaishvili, M. Cova, C. Kruegel, and G. Vigna, “Revolver: An automated approach to the detection of evasiveweb-based malware,” in Proceedings of the 22Nd USENIX Conference on Security, ser. SEC’13. Berkeley, CA, USA: USENIX Association, 2013, pp. 37–652. [Online]. Available:http://dl.acm.org/citation.cfm?id=2534766.2534821

[4] V. Sachin and N. Chiplunkar, “Surfguard javascript instrumentation-based defense against drive-by downloads,” in Recent Advances in Computing and Software Systems (RACSS), 2012 International Conference on,April 2012, pp. 267–272.

[5] T. Matsunaka, J. Urakawa, and A. Kubota, “Detecting and preventing drive-by download attack via participative monitoring of the web,” in Information Security (Asia JCIS), 2013 Eighth Asia Joint Conference on, July 2013, pp. 48–55.

[6] I. Baxter, A. Yahin, L. Moura, M. Sant’Anna, and L. Bier, “Clone detection using abstract syntax trees,” in Software Maintenance, 1998. Proceedings., International Conference on, Nov 1998, pp. 368–377.

[7] I. Neamtiu, J. S. Foster, and M. Hicks, “Understanding source code evolution using abstract syntax tree matching,” in Proceedings of the 2005 International Workshop on Mining Software Repositories, ser. MSR ’05. New York, NY, USA: ACM, 2005, pp. 1–5. [Online]. Available: http://doi.acm.org/10.1145/1082983.1083143

[8] R. C. Holt, A. Winter, and A. Schurr, “Gxl: toward a standard exchange format,” in Reverse Engineering, 2000. Proceedings. Seventh Working Conference on, 2000, pp. 162–171.

[9] E. Mamas and K. Kontogiannis, “Towards portable source code representations using xml,” in Reverse Engineering, 2000. Proceedings. Seventh Working Conference on, 2000, pp. 172–182.Microsoft. (2000) Biztalk framework 2.0 draft:

[10] Document and message specification. Website.Last checked: 24.11.2014. [Online]. Available: http://www.xml.com/pub/r/686

[11] W3C. (2000) Extensible markup language (xml) 1.0 (fifth edition). Website. Last checked: 25.10.2016. [Online]. Available: https://www.w3.org/TR/xml/

[12] N. Nurseitov, M. Paulson, R. Reynolds, and C. Izurieta, “Comparison of JSON and XML Data Interchange Formats: A Case Study.” CAINE, 2009, pp. 157–162. [Online]. Available: http://www.mendeley.com/research/comparison-json-xml-data-interchange-formats-case-study-4/

[13] A. Nappa, M. Z. Rafique, and J. Caballero, “The malicia dataset: Identification and analysis of drive-by download operations,” Int. J. Inf. Secur., vol. 14, no. 1, pp. 15–33, Feb. 2015. [Online]. Available:http://dx.doi.org/10.1007/s10207-014-0248-7

[14] “Malware domain list,” http://www.malwaredomainlist.com/, accessed: 2015-10-30.

[15] “hphost online,” http://www.hosts-file.net, accessed: 2016-08-30.

[16] “Suspicious domain - sans internet storm center,” https://isc.sans.edu/suspicious domains.html, accessed: 2016-08-30.

[17] “Free online tools for lookup malicious websites,” https://

[18] zeltser.com/lookup-malicious-websites/, accessed: 2015-10-30. “Malicious website list with securit report — quttera,”https://quttera.com/lists/malicious, accessed: 2016-08-30.

[19] “Malc0de database,” http://malc0de.com/database/, accessed: 2015-10-30.

[20] 32.2. ast — abstract syntax trees – python 2.7.11 documentation. [Online]. Available: https://docs.python.org/2/library/ast.html

[21] Google code arhive. [Online]. Available: https://code. google.com/archive/p/defusinator/