Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 93 (2016) 768 – 773

Procedia Computer Science

6th International Conference On Advances In Computing & Communications, ICACC 2016, 6-8 September 2016, Cochin, India

AMA: Static Code Analysis of Web Page For The Detection of Malicious Scripts

*Prabhu Seshagiri[a], Anu Vazhayil[b], Padmamala Sriram[b]

[a]*Amrita Center For Cybersecurity Systems and Networks, Amrita Vishwa Vidyapeetham, Amritapuri Campus, India*
[b]*Computer Science and Engineering, Amrita Vishwa Vidyapeetham, Amritapuri Campus, India*

Abstract

JavaScript language, through its dynamic feature, provides user interactivity with websites. It also pose serious security threats to both user and website. On top of this, obfuscation is widely used to hide its malicious purpose and to evade the detection of antivirus software. Malware embedded in web pages is regularly used as part of targeted attacks. To hinder detection by antivirus scanners, the malicious code is usually obfuscated, often with encodings like hexadecimal, unicode, base64, escaped characters and rarely with substitution ciphers like Vigenere, Caesar and Atbash. The malicious iframes are injected to the websites using JavaScript and are also made hidden from the users perspective in-order to prevent detection. To defend against obfuscated malicious JavaScript code, we propose a mostly static approach called, AMA, Amrita Malware Analyzer, a framework capable of detecting the presence of malicious code through static code analysis of web page. To this end, the framework performs probable plaintext attack using strings likely contained in malicious web pages. But this approach targets only few among many possible obfuscation strategies. The evaluation based on the links provided in the Malware domain list demonstrates high level accuracy

Keywords:
JavaScript; Obfuscation; Static Detection; Probable Plaintext attack

1. Introduction

JavaScript is an interpreted programming language embedded into web pages with the aim of providing user interactivity and dynamic content. Upon loading a webpage, the web browser, parses, compiles, and executes scripts (eg: JavaScript, ActionScript, ActiveX etc) in the page. Users browser, extensions and plugins are the major attack surface for attackers. Malicious JavaScript programs can take advantage of execution in the foreign environment containing possible browser-related vulnerabilities. By doing so, it lures the victim into clicking on a link hosted by a malicious host. For example, drive-by-download attack [2] downloads and executes malware onto the browser.

JavaScript/DOM based attacks have been reported as top Internet security threats in recent years [1] [4] [14] [13]. Most internet users rely on the protection provided by antivirus software as it militates against the malware attacks. Obfuscation technique is considered to be an effective way of hiding malicious intent of JavaScript code, mainly due to the following reasons: First, obfuscation could easily evade the static signature-based detection techniques like antivirus and IDS. Second, change in the signature of the code during runtime could be achieved by using the dynamic code

doi:10.1016/j.procs.2016.07.291

generation and runtime evaluation functionalities of JavaScript. In fact, recent trends suggest that, attackers have been increasingly applying obfuscation techniques to evade the anti-virus detection and to conceal its malicious intent[2] [3].

Several approaches[6] [7] [8] [2] [16] [3] focus on the detection of malicious Javascript code through either static or dynamic analysis or combination of both[14]. These approaches either adopt machine learning techniques or perform runtime analysis. Machine learning techniques proven to give better results, achieve high accuracy, with a large set of features. However, most approaches do not make any attempts to deobfuscate the embedded code, to detect the presence of malicious code, hence easily evaded. On the contrary, dynamic analysis can expose the malicious behavior of JavaScript during runtime, but the performance overhead is the major factor preventing its day-to- day and large-scale use.

In this work, we propose AMA, a static code analyzer of malicious scripts in the web page. AMA analyzes certain HTML elements and JavaScript code, detects the presence of obfuscated code, and identifies the ones that can be potentially malicious AMA by launching a probable plaintext attack on the deobfuscated code. This work uses an existing classification method[11] to statically analyze the web page based on the initial HTTP response from the given URL.

In the evaluation of AMA, we use links provided in Malware Domain List[10] as malicious sample set to test the correctness of the framework. The evaluation report shows that AMA has a small number of true negatives. Application of this method shows significant performance gain over other approaches.

2. Contribution

Our work makes the following contributions:

1. **Only static detection of obfuscated malicious code**: AMA is a lightweight, only static approach to detect (obfuscated) malicious scripts, most of which evade the detection of state-of-art anti-virus software.
2. **Probable plaintext attack**: AMA detects limited types (escape, hex, Caesar, Atbash and Vigenere) of obfuscation techniques and launches a probable plaintext attack on deobfuscated strings to test whether the given sample is a malware or not.

3. Background

3.1. Attack Methodologies

Attackers use a variety of techniques to compromise and inject malicious code into web pages. The recent incidents of malicious advertisement attacks in Yahoo, Youtube and Dailymotion shows the use of highly effective `iframe` based web attacks. An invisible `iframe` has redirected the Daliymotion websites' users to the exploit serving bogus anti-virus malware page[22]. `iframes` are used for loading dynamic content in a webpage from other source. `iframe` based attacks suggest that attackers need not always compromise the servers serving advertisements. Attackers could redirect the users using malicious `iframe` embedded in a web page. Hence attackers embed hidden `iframes` into the page to trick a legitimate user redirect to a malicious page.

Listing 1: Examples of hidden iFrame malwares

```
<iframe width="700" height="500" frameborder="0" src="http://facebook.cn"
    visibility="false">
<iframe width="0" height="0" frameborder="0" src="http://facebook.cn"
    visibility="true">
```

Listing 1 illustrate examples of hidden `iframes`. The first `iframe` is invisible because its HTML attribute visibility is set as false. The second `iframe` has zero width and height.

The Mozilla developer community considers the JavaScript function `eval()` as dangerous[14], which executes the code it's passed with the privileges of the caller. If there's an `eval()` function in a page with a string that could be affected by malicious party, it may end up running malicious code on the user's device with browser/extension's permission. Presence of `eval()` can be classified as suspicious. Listing 2 shows an example of sample obfuscated JavaScript code which injects a hidden `iframe` into the web page.

Listing 2: Obfuscated code

```
<script>eval(unescape('%64%6F%63%75%6D%65%6E%74%2E %77%72%69%......(some
    bytes skipped) .......3E%3C%2F%69%66%72%61%6D%65%3E%27%29')); </script>
```

Listing 3: Deobfuscated code

```
document.write('<iframe src="http://sedpoo.com/?338375" width=0 height=0>
    </iframe>');
```

4. Overview

This section gives an overview on the design of AMA.

4.1. Function Invocation Based Analysis

AMA is designed in such way that the deobfuscated or plaintext code has to contain any of the following HTML attributes or JavaScript functions mentioned in Table 1 to further inspect for the presence of malware.

HTML attributes and JavaScript Functions	Suspicious behavior
`iframe`	Malicious `src` domain, `visibility` attribute set as `false` and dimensions (`width`/`height`) less than one
`img`	Malicious `src` domain
`script`	Malicious `src` domain
`eval()`	Presence of obfuscated code, injecting a plaintext code with invisible or hidden `iframe` and presence of shellcode
`unescape()`	Analyzing the string argument, presence of code injecting invisible or hidden `iframe` and presence of shellcode

Table 1: Suspicious behaviour of HTML attributes and JavaScript

The challenge here is to distinguish function invocations in obfuscated malicious code from that in benign code. AMA is selective about functions in which obfuscated strings are passed as arguments. Figure 1 demonstrates the decision tree of the AMA. It checks for the presence of `iframes`, its visibility and the domain of URL in the `iframe`. Analyzer also uses Google Safe Browsing API[24] to check whether the `src` domain of the URLs of either `iframe`, `img`, web page or `script` sources are blacklisted.

Most of the malicious samples which was collected from the Malware Domain List, contains obfuscated scripts wrapped by a user defined function, and called inside from an `eval()` function, in-order to execute at the user's browser. The JavaScript coding guidelines[14] does not promote the use of `eval` functions in major open source projects. Thus, our proposed system assumes that the presence of `eval()` function is considered as suspicious. But AMA does further role in detecting the presence of malicious code inside `eval()` either as plaintext or obfuscated. Our system uses bruteforce technique in detecting the type of obfuscation and deobfuscates using probable plaintext attack.

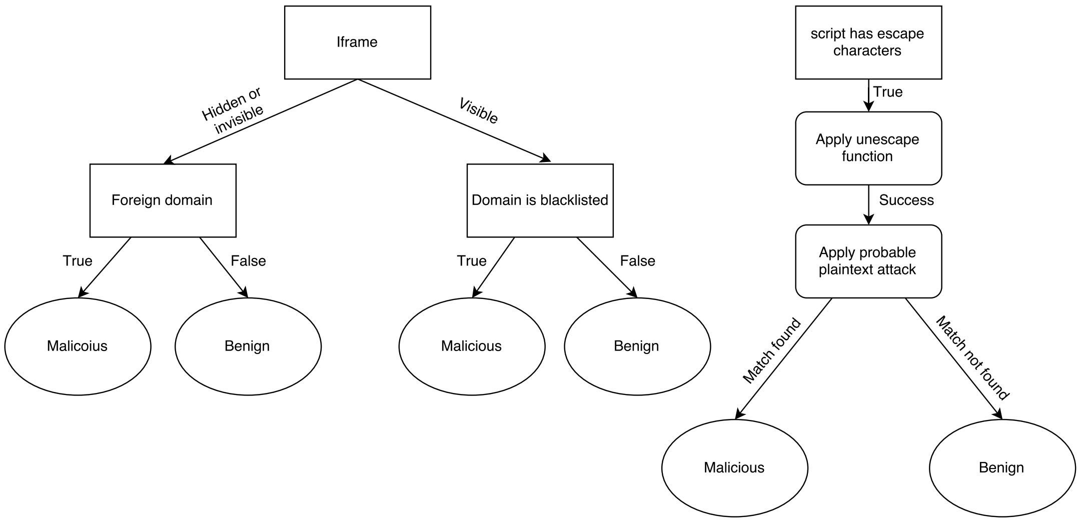

Fig. 1: Decision Tree for malicious `iframe` and `escape` string detection

4.2. *Plaintext attack on deobfuscated strings*

Our system deobfuscates obfuscated strings, it tries to classify whether the plaintext string is malicious or not. It maintains a list of sample malicious scripts' word frequency distribution table in the database. Whenever a test web page is provided, the deobfuscated strings are tokenized and its word frequency distribution table is calculated. Not all words from the sample malware list are used by the malware analyzer, instead, it selects top 20% of the total words from a sample malware word frequency list.

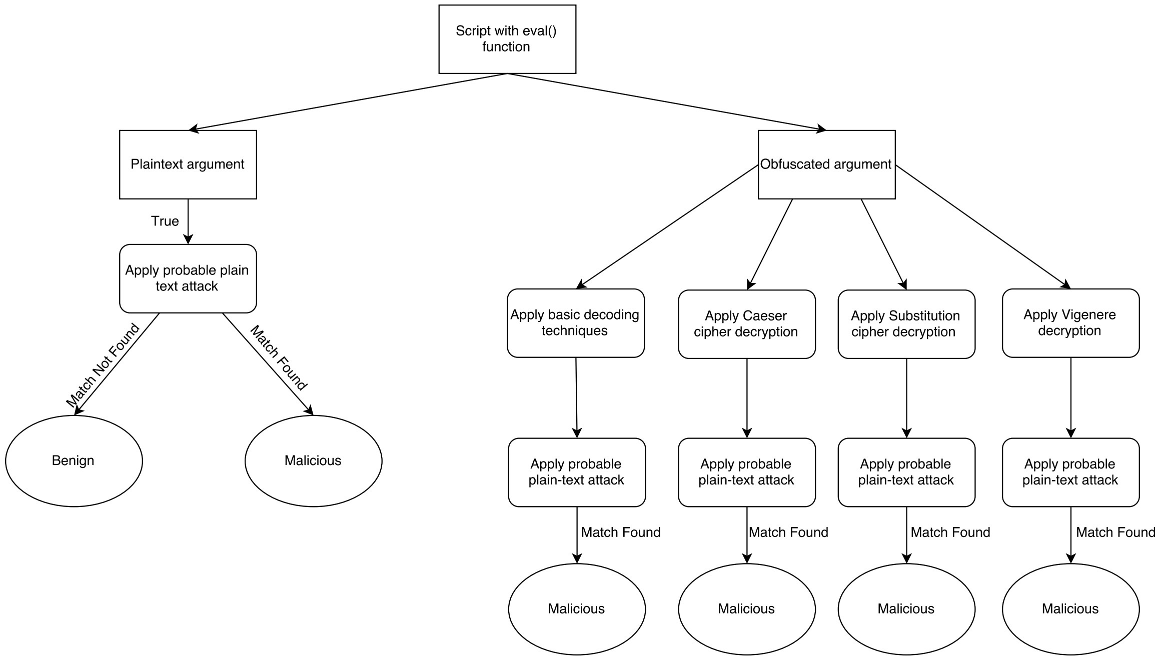
Script with eval() function
Plaintext argument
Obfuscated argument
True
Apply probable plain text attack
Match Not Found
Match Found
Benign
Malicious
Apply basic decoding techniques
Apply Caeser cipher decryption
Apply Substitution cipher decryption
Apply Vigenere decryption
Apply probable plain-text attack
Apply probable plain-text attack
Apply probable plain-text attack
Apply probable plain-text attack
Match Found
Match Found
Match Found
Match Found
Malicious
Malicious
Malicious
Malicious

Fig. 2: Decision tree for launching probable plain-text attacks on encodings and simple substitution ciphers

$$N = 0.2 \times WordCount(MalWordList_i) \tag{1}$$

Our malware analyzer pre-calculates the threshold value by adding up the scores of the top 20% words chosen from the sample malware word frequency list.

$$Threshold_i = \sum_{i=1}^{N} Score_i \tag{2}$$

Once the threshold of sample is calculated, analyzer checks if the words mentioned in the sample word list are present in the testing list. If present, then it sums up the frequency of the words in the testing list. After going through the 20% of the words from the sample word lists, AMA checks if the threshold of the testing list if greater than the precalculated threshold. If it is greater, it flags the testing list as malicious, otherwise moves to the next sample malicious word list for repeating the same step until all the malicious word lists are covered.

5. Evaluation

We deployed our system locally, tested the detection rate, calculated false positives and false negatives. We have collected more than 5000 malware page links from Malware Domain List. We collected 789 malwares consisting of malicious `iframe`, `eval()`, `fromCharCode()`, `unescape()` and images, our system has detected 143 as benign. From top 1000 Alexa benign websites, our system has wrongly detected 46 malicious URLs. Out of 1789 URLs, our system wrongly classified 189 URLs, with around 89% accuracy.

	Tested Benign URLs	**Tested Malicious URLs**	**Total tested URLs**
	1000	789	1789
Detected as Malicious URLs	46	143	189
Detected as Benign URLs	954	646	1600
	False Positive: 4.6%	**False Negative**: 18.1%	**Accuracy**: 89.44%

Table 2: Testing results

6. Related Work

Several approaches have been proposed to detect Malicious JavaScript and HTML elements in the web page. JStill[14] detects malicious JavaScript code by function invocation based analysis, using a combination of both static and lightweight dynamic analysis. WANG et al[15] analyzes and extracts malicious script features, using the machine learning technology, SVM. NOZZLE[16] does a runtime analysis of the script code for the detection of heap- spray exploits in the web page. ZOZZLE[17] uses Bayesian classification of hierarchical features to identify JavaScript syntax elements that suspicious. Prophiler[3] uses static analysis technique to quickly analyze the content of a web page and filter out benign elements using machine learning techniques. There are tools like Google Safe Browsing API[24], McAfee Siteadvisor[25], DeFusinator[26] and Trafficlight addon[27] to detect malicious elements in the web page. Most of these tools use the cloud database for checking the malicious URLs and require the frequent updation of their database. Our proposed system aims at protecting the user from zero day attacks as it is a content based malware filtering solution.

7. Conclusion

In this paper, Amrita Malware Analyzer is presented for analyzing and detecting malicious HTML elements and JavaScript code in the web pages. To surpass the anti-virus detection, the web malwares are usually obfuscated. This work uses a simple classification method for detecting malicious web pages that requires assessing attributes of the HTTP response. AMA, a framework that efficiently deobfuscates the web malwares using probable plaintext attack.

Our framework is currently deployed as a standalone web application. It could be also used as a browser extension to inspect the incoming web pages. AMA provides detailed analysis report of the page including the geographical location of the website and javascripts, images and iframes of the page.

The classification method used in our framework is based on common attributes of malicious pages, attackers could evade the detection mechanism by changing the structure of the page. For instance, a malicious page will not get detected by our framework if the exploit is not imported via an `iframe` or Javascript eg: the VML exploit[12]. The deobfuscation of various cryptographic ciphers will be explored as part of future work. This approach targets only few among many possible obfuscation strategies, it helps to strengthen current defenses against web malwares.

Based on the evaluation, AMA gives significant accuracy and performance in detection of malicious web pages listed in Malware Domain List.

References

1. Fossi, M., Johnson, E., and Mack, T. Symantec global internet security threat report. *Tech Report* 2009; Symantec.
2. Cova, M., Kruegel, C., and Vigna, G. Detection and analysis of drive-by- download attacks and malicious javascript code. In Proceedings of the 19th international conference on World wide web (New York, NY, USA, 2010), WWW '10, ACM, pp. 281290.
3. Canali, D., Cova, M., Vigna, G., and Kruegel, C. Prophiler: a fast filter for the large-scale detection of malicious web pages. In Proceedings of the 20th international conference on World wide web (New York, NY, USA, 2011), WWW 11, ACM, pp. 197206.
4. Percoco, N. J. Global security report 2010 analysis of investigations and penetration tests. *Tech. report* SpiderLabs; 2010.
5. Online JavaScript Obfuscator: http://www.daftlogic.com/ projects-online- javascript-obfuscator.htm.
6. Choi, Y., Kim, T., Choi, S., and Lee, C. Automatic detection for javascript obfuscation attacks in web pages through string pattern analysis. In Proceedings of the 1st International Conference on Future Generation Information Technology (Berlin, Heidelberg, 2009), FGIT 09, Springer-Verlag, pp. 160172.
7. Likarish, P., Jung, E., and Jo, I. Obfuscated malicious javascript detection using classification techniques. In Proceedings of the 4th International Conference on Malicious and Unwanted Software (Oct 2009), MALWARE '09, pp. 4754.
8. Kaplan, S., Livshits, B., Zorn, B., Siefert, C., and Curtsinger, C. nofus: Automatically detecting + string.fromcharcode(32) +obfuscated .tolowercase() + javascript
9. Ratanaworabhan, P., Livshits, B., and Zorn, B. Nozzle: a defense against heap-spraying code injection attacks. In Proceedings of the 18th conference on USENIX security symposium (Berkeley, CA, USA, 2009), USENIX Association, pp. 169186.
10. Malware domain list, http://www.malwaredomainlist.com/
11. Christian Seifert, Ian Welch, Peter Komisarczuk, Identification of Malicious Web Pages with Static Heuristics, Telecommunication Networks and Applications Conference, 2008. ATNAC 2008. Australasian.
12. Microsoft Cooperation, Microsoft security bulletin ms06-055: Vulnerability in vector markup language could allow remote code execution, 2006, pp. available from http://www.microsoft.com/technet/ security/Bulletin/MS06–055.mspx; accessed on 14 February 2007.
13. W. Xu, F. Zhang and S. Zhu, "The power of obfuscation techniques in malicious JavaScript code: A measurement study", Proceedings of the 2012 7th International Conference on Malicious and Unwanted Software
14. Xu, W., Zhang, F., Zhu, S.: JStill: mostly static detection of obfuscated malicious JavaScript code. In: Proceedings of the Third ACM Conference on Data and Application Security and Privacy, CODASPY13 (2013).
15. WANG Wei-Hong, LV Yin-Jun, CHEN Hui-Bing and FANG Zhao-Lin. A Static Malicious Javascript Detection Using SVM. In proceedings of the 2nd International Conference on Computer Science and Electronics Engineering (ICCSEE 2013)
16. Ratanaworabhan, P., Livshits, B., and Zorn, B. Nozzle: a defense against heap-spraying code injection attacks. In Proceedings of the 18th conference on USENIX security symposium (Berkeley, CA, USA, 2009), USENIX Association, pp. 169186.
17. Curtsinger, C., Livshits, B., Zorn, B., and Seifert, C. Zozzle: Fast and precise in-browser javascript malware detection. In Proceedings of the 20th conference on USENIX security symposium (2011), USENIX Association
18. Ariana (2010). Q1'10 web-based malware data and trends. [ONLINE] Available at: http://blog.dasient.com/2010/05/q110-web-based-malware- data-and-trends.html. [Last Accessed 20 March 2016].
19. Yan, Guanhua, et al. "Malware propagation in online social networks: nature, dynamics, and defense implications." Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security. ACM, 2011.
20. Larry Seltzer (2014). Yahoo serves malicious ads. [Online] Available at: http://www.zdnet.com/yahoo-serves-malicious-ads-7000024775/ [Last Accessed 20 March 2016]
21. Delete Malware blogspot (2013): Remove the "Ads not by this site" Browser Hijacker. [Online] Available at: http://deletemalware.blogspot.sg/ 2013/01/remove-ads-not-by-this-site.html [Last Accessed 20 March 2016].
22. Michael Mimoso (2014). Malicious ad on dailymotion redirect to fake AV attack. [ONLINE] Available at: http://threatpost.com/malicious-ads-on- dailymotion-redirect-to-fake-av-attack/103494. [Last Accessed 20 March 2016].
23. Mozilla Developer Network, eval() documentation, [Online] Available at: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/eval [Last Accessed 20 March 2016].
24. Google Inc. Safe Browsing for Firefox. Google Inc. Safe Browsing for Firefox.
25. McAfee SiteAdvisor. http://www.siteadvisor.com
26. DeFusinator: https://code.google.com/p/defusinator
27. Trafficlight: https://addons.mozilla.org/En-us/firefox/addon/trafficlight