ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 3, Issue 2, pp: (662-666), Month: April - June 2015, Available at: www.researchpublish.com

JAVA SCRIPT MALWARE DETECTION ON WEB BROWSER

[1]B.P. Anushree, [2]J.Lakshmi, [3]D.Meghana, [4]S.Nandini, [5]S.Suresh Kumar

[1,2,3,4] BE Student, Department of CSE, Vivekananda Institute of Technology, Bangalore, India,
[5]Asst.prof, Department of CSE, Vivekananda Institute of Technology, Bangalore, India,

Abstract: **In the recent times, most of the systems connected to Internet are getting infected with the malware and some of these systems are becoming zombies for the attacker. When user knowingly or unknowingly visits a malware website, his system gets infected. Attackers do this by exploiting the vulnerabilities in the web browser and acquire control over the underlying operating system. Once attacker compromises the users web browser, he can instruct the browser to visit the attackers website by using number of redirections. During the process, user's web browser downloads the malware without the intervention of the user. Once the malware is downloaded, it would be placed in the file system and responds as per the instructions of the attacker. These types of attacks are known as Drive by Download attacks. Now-a-days, Drive by Download is the major channel for delivering the Malware. In this paper, JavaScript malware detection an extension to the browser is presented for detecting and defending against Drive by Download attacks via HTML tags and JavaScript.**

Keywords: **Malware, HTML tags, DOM Change Methods, JavaScript Functions, Web Browser, Web Browser Extensions, Drive by Download Attacks.**

I. INTRODUCTION

With the increasing usage of Internet, the attacking channels are flagging towards the usage of web browsers and web applications widely. Browsers have evolved from static document renderers to today's sophisticated execution platforms for web applications. Browsers are very much susceptible to attacks through exploitable vulnerabilities. Attacker uses browser/ browser plug-in/ webpage as vehicles to infect end system without directly connecting to them.

Attacks are launching through memory, web content, web mark-up or scripting language level exploits. In a typical XSS attack, due to the vulnerability in validating the input, attacker can inject malicious JavaScript code as a comment in the blog or reply to a post. This injection leads to the execution of malicious JavaScript code with the privileges of web application. This injection affects the users who visit these websites. This makes attacker get unauthorized access to data stored at users end system without their knowledge.

In this attack, initially the attacker compromises a legitimate web server and inserts a script in web application. When user accesses the web site that was compromised by the attacker, web server sends the injected script along with the requested page. This script is either an exploit script or it imports exploit from a central server which is controlled by the attacker and this import is either a direct inclusion of the resources from the remote server or through a number of redirections the browser is instructed to follow. A redirection starts from one web server to the other that actually plays part of hop points. After following a number of redirections the users request reaches the central exploit server. The central exploit server sends the exploit script depending on the fingerprint of the user end system. Fingerprinting is done by using the User-Agent field present in the HTTP request coming from the user's web browser. Fingerprinting includes web browser type and underlying operating system along with version details. Imported exploit script is used to exploit the vulnerability present either in the browser/ browser plug-in/ webpage. This exploit instructs the browser to visit the malware distribution site. This is, actually, when the Drive by Download starts. Malware executables are downloaded and user's end system automatically installs and executes the malicious code.

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 3, Issue 2, pp: (662-666), Month: April - June 2015, Available at: www.researchpublish.com

II. BACKGROUND

In the earlier days attackers used vulnerabilities in the operating system configuration or installed applications for exploitation. With the advent of web, attackers have changed their target to web browser and its plug-ins. Some of the current day attacks such as XSS (Cross Site Scripting) and CSRF (Cross Site Request Forgery) does not require exploiting the vulnerabilities in the client's browser or system. In these attacks, malicious code is injected into the webpage and attacker tricks the client to visit the infected webpage for getting access to user's web browser. Through this Drive by Download attack is carried out.

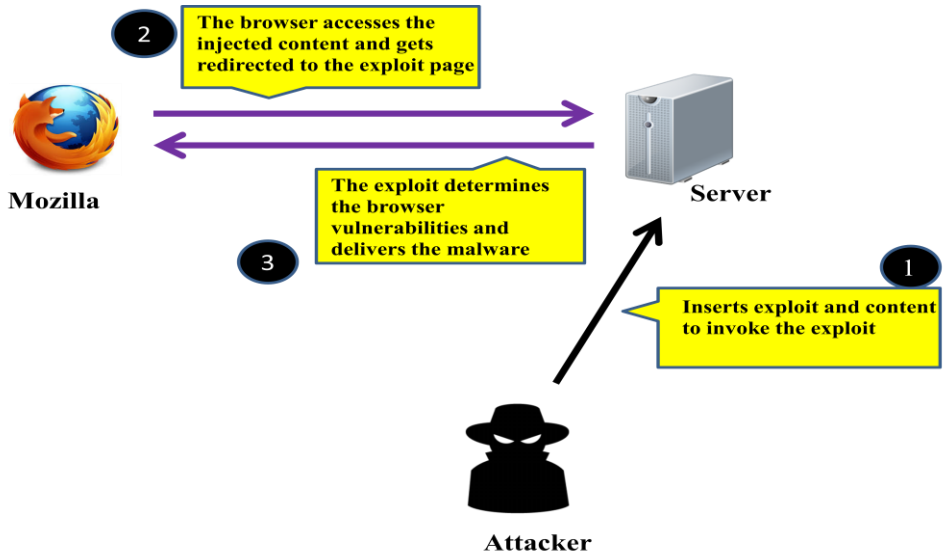

Fig. 1: Drive by Download attack scenario - Exploit code resides on Target server

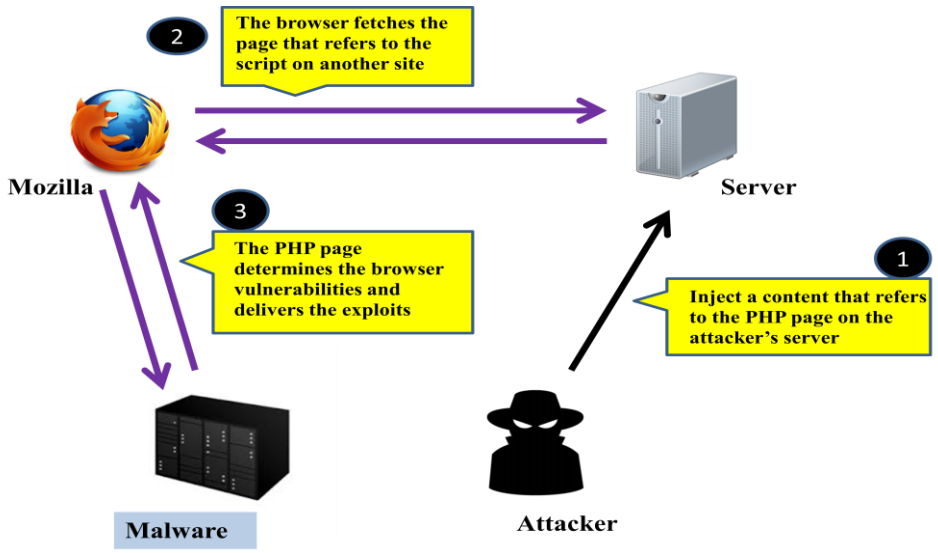

Fig. 2: Drive by Download attack scenario Exploit code resides on Attacker Server

In the first scenario, attacker prepares the attack using a genuine web server. Attacker injects into the target web server, PHP code as well as web content to redirect the user to PHP code through iframe tag. Web browser accesses the injected web page when connected to the target web server. After accessing the injected web page from the server, web browser gets redirected to PHP page. This redirection is possible through iframe tag. Now the web server sends the attack code or payload to the web browser if it is vulnerable. Target browser runs the exploit script received from the target web server as it is from the same origin. This is one scenario for Drive by Download attack, where the exploit code also resides in the target web server as shown in Figure 1.

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 3, Issue 2, pp: (662-666), Month: April - June 2015, Available at: www.researchpublish.com

In second scenario exploit code resides on attacker server as shown in Figure 2. In this scenario, attacker injects the content into target web server. Injected content refers to a script residing in attacker's web server. Target browser fetches the injected web page from the target web server. Whenever browser renders the fetched web page, client browser is being redirected to a script on attacker's web server which is referred by using Script src tag. Attacker's web server delivers the exploit code to the target browser if it is vulnerable.

III. OUR APPROACH

Drive by Download attacks made through malicious Java Scripts is detected by examining "The structure of the static HTML page, Dynamic JavaScript Code and the DOM changes in a web page". These characteristics are intercepted for every incoming webpage and intercepted behaviour is com-pared against the ruleset for analyzing and deciding whether the webpage is malicious or not. Ruleset is designed by analyzing various malicious JavaScript injection attack patterns. This detection approach is implemented as an extension to web browser.

A. Monitoring Structure of the static HTML page:

The incoming web page can be monitored and the vulnerable HTML tag properties can be extracted from it [13], [14], [17], [18].

Table I: Monitorable Behaviours in HTML Tags of a webpage

HTML Tag	Monitorable behaviours
frame/ iframe	- Cross Domain Source File - Invisible/Hidden Iframes
script	- Cross Domain Source File - Wrong File name Extensions - Monitoring inline Script
img	- Cross Domain Source File - Wrong File name Extensions
anchor/link/area	- Cross Domain Source File
meta	- If contentType = refresh then monitor src attribute
object	- Cross Domain Source File - Class ID of the plug-in to be invoked

The extracted properties specify the behaviour of the web page and are compared against the Ruleset and if any match is found then the web page is treated as malicious page. The HTML tags and their vulnerable properties to be monitored are presented in Table I.

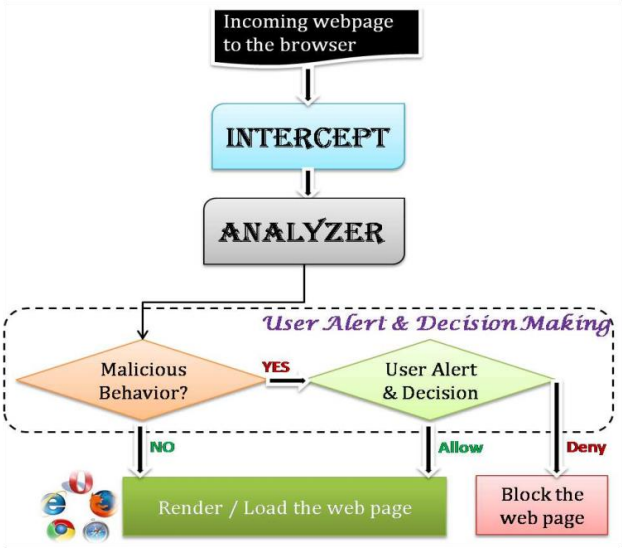

Fig. 3: Block Diagram of JavaScript malware detection

B. System Design:

Whenever user is browsing a webpage, client initiates a HTTP request for the required resource and the server response is called as HTTP Response.

HTTP Response includes the requested resource. The HTTP Response is the incoming webpage and this webpage is monitored for determining whether it has any malicious con-tents or not. In this incoming webpage, vulnerable tags, DOM change functions and dynamic code execution functions are monitored for determining the maliciousness of the webpage.

Block diagram of the Java script Guard is shown in Figure 3. Incoming web page is intercepted for detecting vulnerable HTML tags, DOM change functions and runtime JS code execution functions in it. If any of them are detected in the webpage then the respective properties are sent to Analyzer module. Analyzer module gets the extracted properties and it checks for "Hidden iframe redirections, Unauthorized Redi-rections and Obfuscated code". Analysis report is sent to the User Alert & Decision Making component in which user will be alerted if the webpage contains any malicious behaviors present. Webpage may be either rendered in the web browser or blocked depending on the decision taken by the user.

C. Implementation:

The detection mechanism is implemented in JavaScript language. Implemented security mechanism consists of various JavaScript components namely tagMonitor.js, dynamicJSMon-itor.js, reportGenerator.js and jQueryAlert.

"tagMonitor.js" monitors the vulnerable HTML tags and specified attributes of tags in every webpage by using DOM language. The list of HTML tags is presented in Tables 1 and 2. The HTML tags are stored in an array.

monitorTags = new Array("iframe", "script", "img", " area", "link", "a", "frame", "form", "embed", " applet", "meta", "object", "html", "head", " title", "body");

dynamicJSMonitor.js tracks the function calls that are used to dynamically interpret JavaScript code (e.g., eval, setTime-out, unescape), and DOM changes that may lead to code execution (e.g., document.write, document.createElement, doc-ument.location). It also retrieves the parameters of these functions and verifies whether any vulnerable HTML tags are dynamically created. For tracking the specified functions, hooks are being created for each function.

Internal function hooking (IFH) mechanism implemented for monitoring eval() function is given below

```
var evalp=[];
var oldEval = eval;
eval = function eval(param) {
evalp.push(param); return oldEval(param);
}
```

reportGenerator.js generates the report of the webpage by consolidating the result from tagMonitor.js and dynamicJS-Monitor.js components and it includes the mechanism for alerting the user if the webpage is malicious.

```
Report(Result1, Result2) {
//Writing the result to panel finalReport = Result1 + Result2; }
```

Result1 is the webpage analysis report received from component 1, i.e., tagMonitor.js and Result2 is the webpage analysis report received from component 2, i.e., dynamicJS-Monitor.js.

jQueryAlert component uses jQueryAlert.js and jQueryAlert.css files which internally handles jConfirm() method for

alerting the user.

```
jConfirm (MaliciousResult, 'Warning by Javascript malware detection');
```

User can confirm whether to continue browsing the web-page or quit the website

from the popup given by the Add-on.

IV. CONCLUSION

In this paper, JavaScript malware detection is presented for analyzing and detecting malicious JavaScript injections in the incoming web pages. JS Guard resides as part of the client web browser and detects the malicious web pages. In addition to the detection, JS Guard extension provides the flexibility to user for viewing the detailed analysis report of the webpage. Furthermore this extension gives an option for user to decide whether to continue browsing the webpage or not. Testing has been performed on JavaScript Guard using known Genuine and Malicious URLs. JavaScript Guard has detected the malicious web pages with lesser rates of false positives (0.72 %) and false negatives (9.04 %). JS Guard installed web browser have taken 180 ms more time for loading a webpage when compared with the page load time of the web browser when JavaScript Guard is not installed.

In the future, we plan to evaluate more number of URLs and try to improve the detection mechanism to further reduce the false positive and false negative rates by analyzing new attack trends and building intelligence. Further, we want to improve the performance by applying various code optimization techniques.

ACKNOWLEDGEMENT

Our sincere thanks to Department of Electronics & Information Technology (Deity), Ministry of Communications and Information Technology, Government of India for supporting this research work.

REFERENCES

[1] 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)

[2] D.Nandhini, Kalpana.G and Abhilash.R. Browser Authentication and Inline Code Analyzer with Common Centralized Browser Security Violation Report Management System for Mitigating Web Attacks. In proceedings of International Journal of Engineering Research and Development

[3] Pratik Upadhya, Farhan Meer, Acquin Dmello and Nikita Dmello. Runtime Solution for Minimizing Drive-By-Download Attacks. In proceedings of International Journal of Modern Engineering Research (IJMER).

[4] Bin Liang, Jianjun Huang, Fang Liu, Dawei Wang, Daxiang Dong and Zhaohui Liang. Malicious Web Pages Detection Based on Abnormal Visibility Recognition. In proceedings of IEEE 2009.

[5] Wei Xu, Fangfang Zhang and Sencun Zhu. JStill: Mostly Static De-tection of Obfuscated Malicious JavaScript Code. In proceedings of ACM, February 1820, 2013.

[6] Krishnaveni Raju, C.Chellappan. Integrated Approach of Malicious Website Detection. In proceedings of International Journal Communications & Network Security (IJCNS), Volume-I, Issue-II, 2011

[7] D. Canali, M. Cova, G. Vigna, and C.Kruegel. Prophiler: A fast filter for the large-scale detection of malicious web pages. In Proceedings of the 20th international conference on World Wide Web, pages 197-206. ACM, 2011.

[8] Andreas Dewald, Thorsten Holz, Felix C. Freiling. PADSandbox: Sand-boxing JavaScript to fight Malicious Websites.In proceedings of SAC10 March 22-26, 2010.

[9] Google Inc. Safe Browsing for Firefox. Google Inc. Safe Browsing for firefox.

[10] McAfee SiteAdvisor. http://www.siteadvisor.com

[11] DeFusinator: https://code.google.com/p/defusinator

[12] Trafficlight: https://addons.mozilla.org/En-us/firefox/addon/trafficlight

[13] D.J.Guan, Chia-Mei Chen, Jing-Siang Luo, and Yung-Tsung Hou. Malicious Web Page Detection Based on Anomaly Semantics.

[14] Birhanu Eshete, Adolfo Villa orita, and Komminist Weldemariam. BINSPECT: Holistic Analysis and Detection of Malicious Web Pages.

[15] S.S. Sarma. Propagation of Malware Through Compromised Web-sites: Attack Trends and Countermeasures.In proceedings of 11th Association of Anti-Virus Asia Researchers International Conference, December 2008

[16] Aikaterinaki Niki. Drive by Download attacks: Effects and Detection methods.In proceedings of IT Security Conference for the Next Generation.