

Obfuscated Malicious Script Response Technique Deployed at Host Level

Sang-Hwan Oh, Jong-Hun Jung, Hwan-Kuk Kim
Korea Internet & Security Agency, Seoul, Korea.
seif@sharif.edu

Abstract—JavaScript functions have been remarkably enhanced thanks to the emergence of the next generation web standard HTML5 presented by W3C. HTML5 provides powerful functions that could replace non-standard technologies such as Active X by providing functions such as media play, 3-D graphic processing and Web socket communications using JavaScript only without the installation of separate plugins. Along with these trends in the ICT environment, many studies have been done related to threats exploiting JavaScript, which comprises a core of HTML5 functions. There are, however, many limitations in detecting obfuscated malicious scripts since most detection techniques use signature-based pattern matching. This paper will propose a method capable of detecting obfuscated malicious scripts at the host level and preventing the scripts' execution.

Index Terms—Script-based CyberAttack; Web Security; Obfuscated Malicious Script.

I. INTRODUCTION

The type of services provided through the web has been gradually diversified due to the development of web applications. Provision of such services in the past had required use of non-standard plugins such as Active X, entailing a security threat. As such issues have emerged, the HTML 5 standard that can replace Active X by using a JavaScript only has been presented in the recent W3C. HTML5 has powerful measures that can substitute for Active X's functions such as media play, graphic processing and web socket communications utilizing the new JavaScript API using a new tag and JavaScript while maintaining compatibility with the existing HTML. The role of JavaScript has grown, so the security threat has also [1]. Moreover, the concept of obfuscation has emerged to protect the developers' ideas or algorithms since JavaScript is typically used as a client-side language [2]. Apart from its primary purpose, this is often used to bypass security products by hiding malicious codes by hackers.

Most of the ways for web attack by hackers causes a malicious behavior by making a user install and execute a malicious code in the form of drive by download by exploiting the vulnerabilities of web applications. However, the attacks incurring a malicious behavior only by accessing the web site using a JavaScript have recently been found. It is difficult to detect and respond against to the attacks using a JavaScript because the attacks are executed and terminated through a web browser without a separate execution file, and such attacks do not leave any traceable footprints after the browser is closed. Also, most of the JavaScripts are obfuscated for the performance and security, it is possible to bypass detection techniques using signature-based static pattern matching. Therefore, in this paper we describe the

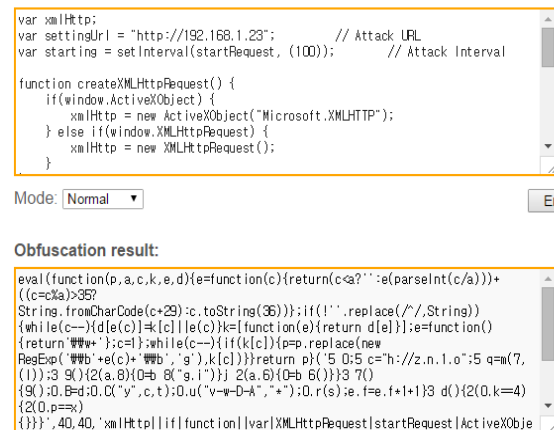
danger of the obfuscated JavaScript attacks and the countermeasure for detecting such attacks and responding against to those.

In Section 2, we explain the method for obfuscating JavaScripts. Section 3 describes the technology for responding to obfuscated malicious scripts. In section 4, we explain the conclusions and further works.

II. RELEVANT RESEARCH

A. JavaScript Obfuscation Technique

Recent web attacks have made signature-based detection difficult through obfuscation. Two most simple ways of the web attacks are: a string split technique which splits a malicious script code into a number of script codes and then, recombines and outputs; and a string replacement technique which replaces a set of strings into an ASCII type string using an encoding function such as escape and unescape. In addition to the two techniques, there are a variety of attacks such as a technique using a known obfuscation technique as shown in Figure 1, a bypass technique using Base64 encoding, a technique using XOR encoding, etc. which simply bypass a signature-based IDS/IPS complexly using various obfuscation methods [3].



```
var xmlhttp;
var settingUrl = "http://192.168.1.23"; // Attack URL
var starting = setInterval(startRequest, (100)); // Attack Interval

function createXMLHttpRequest() {
    if(window.ActiveXObject) {
        xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
    } else if(window.XMLHttpRequest) {
        xmlhttp = new XMLHttpRequest();
    }
}

Mode: Normal Encode

Obfuscation result:
eval(function(p,a,c,k,e,d){e=function(c){return(c&?"":e(parseInt(c/a)))+((c&?a)>3&?
String.fromCharCode(c+29):c.toString(36))};if(!''.replace(/\./,String))
{while(c--){d[e(c)]+=k[e(c)]};function(e){return d[e]};e=function()
{return ""};c=[];while(c--){if(k[c]){p=p.replace(new
RegExp(""+e(c)+""+e(b),"g"),k[c])};return p}('5 0:5 c="h://z.n.1.o":5 q=m(7,
(1)):3 9()2(a,8){0= 8("g,i")} 2(a,6){0= 6}})3 7()
{9():0,0=0:0("v",c,t):0.u("v-w-D-A", "+"):0.r(s).e.f=e.f+1+1)3 d(){2(0,k=4)
{2(0,p=)
}}}',40,40,'xmlhttp|if|function||var|XMLHttpRequest|startRequest|ActiveXObj
```

Figure 1: An example of JavaScript obfuscation

B. Malicious Web Attack Detection Technique

Methods for detecting web attacks can be roughly classified into a method which detects attacks at network level and a method which detects attacks at the host level. In this paper, countermeasures at network level are not considered.

Many studies for detecting malicious web attacks at the host level have been performed. One simple, immediate method to address attacks is to examine an accessed URL

address. One representative technology is McAfee’s SiteAdvisor [4]. This is a method for detecting malicious attacks by checking whether the URL address, which a web browser is accessing, has a history of malicious attacks. This technology is capable of immediate response since it only checks the URL being accessed however, it is difficult to address when the attacker attempts attacks by frequently changing URLs. In order to address the problem, methods for analyzing downloaded web contents and responding against the attacks has been devised. Long Lu has proposed one solution, called BLADE [5], which is a web browser-based incident detection system. BLADE is a detection system which detects an execution file being downloaded through web browser by isolating the execution file into a safe area, executing the file and detects malicious attacks based on behaviors incurred due to the execution. However, it has many difficulties in detecting the incurred malicious behaviors using a script without execution file.

To solve such limitation, a WMDS technique has been prepared [6]. WMDS proposes a malicious behavior detection module, called Observer, to monitor the web browser which is a base for web attack.

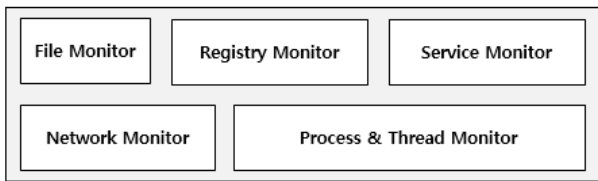


Figure 2: Configuration of Observer

The configuration of Observer consists of File Monitor, Registry Monitor, Service Monitor, Network Monitor and Process & Thread Monitor as shown in Figure 2. It detects malicious behaviours by detecting events occurring at each process through API hooking by attaching Observer to the each of the corresponding processes such as File, Registry, Service, etc. and by analysing the events detected by Observers. This technique is capable of detecting malicious behaviours regardless of the existence of execution file since it monitors a browser which is a base for web attack, however, it results in a high load on the system as Observers are used for the all the processes.

Therefore, we present a technique which can detect malicious behaviours resulting solely from a obfuscated script at the host level through the web with a relatively lower load, compared to the existing ones.

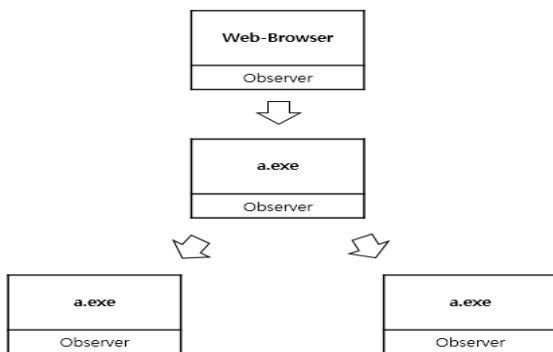


Figure 3: Observers attached to the each of the processes

III. OBFUSCATED MALICIOUS SCRIPT RESPONSE TECHNOLOGY

A basic configuration of the technology is shown in Figure 4. It collects target analysis contents from a web browser through Browser Helper Object (BHO) and extension-type script security program for Internet Explorer and Chrome, respectively, and transmits the collected contents to an agent which is in charge of analysis. The analysis agent determines whether the contents is malicious or not by analyzing the corresponding contents and transmit the result to the script security program. The script security program that receives the result redirects to a safe page or stops the JavaScript on the corresponding page if a malicious behavior is detected and terminates the analysis of the corresponding web page if the result is found normal without further processing.

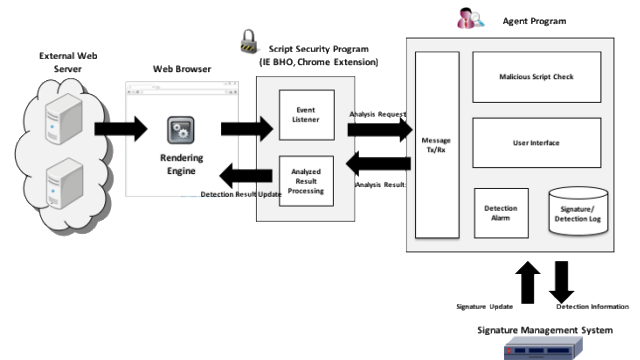


Figure 4: Software configuration for malicious script detection

A. Script Security Program

A script security program has been realized in the form of browser plugin (BHO) in order to detect a specific event of the web browser, and the method detects, collects and transmits the specific event of the web browser through the script security program to the analysis agent.

This technology uses a predetermined interface, called IObjectWithSite provided by IE-BHO to detect events occurring on a web browser. It detects an event [7] which occurs at the time of Object Download shown in Figure 5 using the interface and extracts the downloaded object by stopping rendering. The script information among the information of the object is transmitted to the analysis agent.

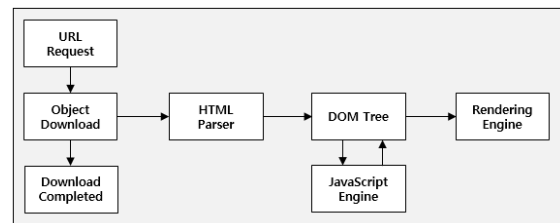


Figure 5: The time of web browser event occurrence

Furthermore, it receives the analysis result from the analysis agent and performs post-process according to the result. If the analyzed result is normal, the rendering of the web browser is resumed. But, if a malicious code is found, two different post-processes will be performed.

The first one is to protect a user from the malicious script by redirecting the user to a safe page and the second one is to prevent execution of the malicious script by stopping the JavaScript on the corresponding web page.

B. Analysis Agent Program

In the analysis agent, the received script information is analyzed and whether the information has a malicious code is determined. First, the analysis agent determines which target analysis scripts are obfuscated. Functions frequently used for obfuscation are the criteria for the determination. Functions, such as Eval(), Document.write(), etc. capable of execution JavaScript, are often used for obfuscation [3] and the analysis agent consider there is obfuscation when those functions are found. The corresponding script performs a routine for deobfuscation. To deobfuscate the script, the agent extracts the original script by inputting the corresponding script into the customized JavaScript engine called V8 which is being used on a Chrome browser and by executing the script. As shown in Figure 6, the extracted original script is determined whether it has a malicious code through pattern matching with a YARA RULE-type signature [8] recording pattern information of malicious scripts.

```
rule yara_4115_4_4115{strings:$str1 = "xmlHttp"$str2 = "count"$str3 = "settingUrl"$str4 = "Attack"$str5 = "value"$str6 = "ActiveXObject" $str7 = "startRequest"$str8 = "createXMLHttpRequest"$str9 = "temp"$str10 = "handleStateChange"$str11 = "Origin"$str12 = "starting"$str13 = "setInterval"$str14 = "status"$str15 = "Control"$str16 = "alert"$str17 = "Response"$str18 = "open"$str19 = "true"$str20 = "Microsoft" $str21 = "responseText"$str22 = "onreadystatechange"$str23 = "setRequestHeader"$str24 = "send"$str25 = "Access"$str26 = "Allow"conditional of them)
```

Figure 6: YARA RULE signature of malicious script

If the extracted original script is determined to be malicious, detection information such as detection time and type of malicious script as shown in Figure 7 is provided to the user in real time using a pop-up notification.



Figure 7: Real time detection notification

Also, the user’s terminal can be protected by transmitting the analyzed result to the script security program and making the script security program perform post-process. Finally, the corresponding module periodically or upon request updates the signature and stores a detection log which includes information such as the detection time, type of malicious script, etc. Figure 8 shows the entire flow of malicious script detection described above.

IV. CONCLUSIONS AND FUTURE WORKS

Compared to functions using the existing HTML, JavaScript functions have been remarkably enhanced due to the emergence of HTML5 as presented by W3C. As a result, the functions that can replace non-standard technologies such as Active X have drawn closer to daily life, but the threats

that can result in damage have, too.

This paper proposed a technique for responding to malicious attack threats occurring by using JavaScript at the host level. Since JavaScript operates on a browser, the analysis targets were collected by stopping the loading of the browser before the corresponding script was executed through browser extension. We determined whether the analysis targets were obfuscated by checking the functions often used for obfuscation among the collected analysis targets, and proposed a measure for detecting and blocking malicious scripts based on signature. This technique has several limitations, however, since it responds to such attacks at the host level using browser extensions. Another problem is dependence on a web browser on the type and version due to the features of a browser plugin known as a development type of script security program. Furthermore, the technique could detect known obfuscation methods such as base62 and base64, but had difficulty detecting new obfuscation methods. So, we can improve performance by minimizing dependence on an operating system and browser on the type and version, and research an algorithm for detecting obfuscation.

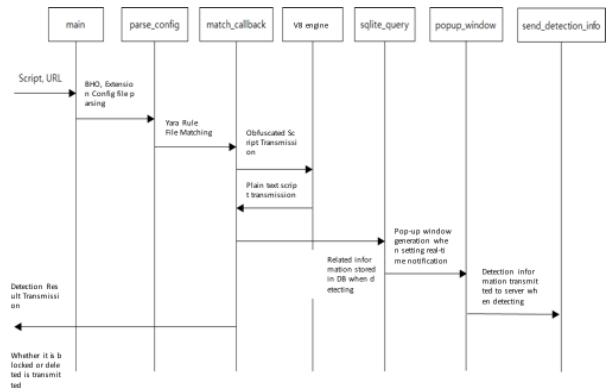


Figure 8: Malicious script detection flow

ACKNOWLEDGEMENT

This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) [B0101-15-0230, Development of Script-based Cyber Attack Protection Technology]

REFERENCES

- [1] Seokchul Kang, 2013. Security issues in a HTML5 service environment Internet & Security Focus
- [2] JScrambler. <https://blog.jscribler.com/protecting-javascript-source-code-using-obfuscation-facts-and-fiction/>
- [3] ASEC Jihun Kim, Understanding JavaScript Obfuscation
- [4] SiteAdvisor. McAfee. Available: <http://www.siteadvisor.com>
- [5] Long Lu, Vinod Yegneswaran, Phillip a. Porras. 2010. BLADE: An attack-agnostic approach for preventing drive by malware infections.
- [6] Young-Wook Lee, Dong-Jae Jung, Sang-Hun Jeon and Chae-Ho Lim, 2012. Design and Implementation of Web-browser based Malicious behavior Detection System (WMDS) Journal of the Korea Institute of Information Security and Cryptology, 22(3).
- [7] DWebBrowserEvents2 interface, MSDN, Microsoft. Available: [http://msdn.microsoft.com/en-us/library/aa768283\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/aa768283(v=vs.85).aspx)
- [8] YARA Documentation, <http://yara.readthedocs.org/en/latest/index.html>.