Suspicious Malicious Web Site Detection with Strength Analysis of a JavaScript Obfuscation

Byung-Ik Kim, Chae-Tae Im, Hyun-Chul Jung
Korea Internet & Security Agency
kbi1983@kisa.or.kr, chtim@kisa.or.kr, hcjung@kisa.or.kr

Abstract

The cyber attacks using web sites for a personal information sale or break down the infrastructures are increasing. To prevent cyber attacks, virtual computer systems are hired and dynamic malicious web site analysis systems are used. However the dynamic analysis systems have to setting up a targeted environment and have a threat of real attack. Unlikely the dynamic analysis system, a static malicious web site analysis system can download a web page source and analysis the web page. The static analysis algorism also has problem, the analysis time is longer than dynamic systems and need a human checking. For this reason, this journal suggest new analysis system reducing the dynamic and static system's problems using suspicious malicious web site detection with strength analysis of a JavaScript obfuscation for new hybrid analysis system.

Keywords: *Malicious Web Sites, Obfuscation JavaScript, Entropy, Frequency, Density*

1. Preface

Recently, personal information leaks on the Internet and the attacks against social infrastructure are on the rise. Generally, the attacker inserts an attack code into the web site using the Internet, or causes a download of the attacking program to the user's computer. These attacks can be made conveniently using JavaScript on a web site. In addition, the JavaScript obfuscation technique is used to hide JavaScript containing the attack code. Various methods are used to prevent attack made through Internet websites.

There is a dynamic analysis system that detects malicious behavior of the web site by directly visiting the site in question and analyzing the result. As the dynamic analysis system visits the web site and analyzes the result, it is vulnerable to an attack. It takes longer than an hour to visit and analyze the site, and the system environment should be created that can make an attack. There is another system type – static analysis system that downloads and analyzes the source of the web site. This system has less vulnerability against the attack, as the attack is not directly made. However, the shortcomings are that the person should analyze the source code manually, and analysis takes a longer amount of time. Therefore, this paper proposes the JavaScript obfuscation strength examination system that will be included in the hybrid type analysis system, which will be developed to solve these shortcomings.

This paper is composed as follows. Chapter 2 describes JavaScript obfuscation. Chapter 3 introduces existing malicious web site analysis methods. Chapter 4 proposes the JavaScript obfuscation strength examination system, and presents the result in comparison with existing systems. Chapter 5 describes a conclusion and further studies.

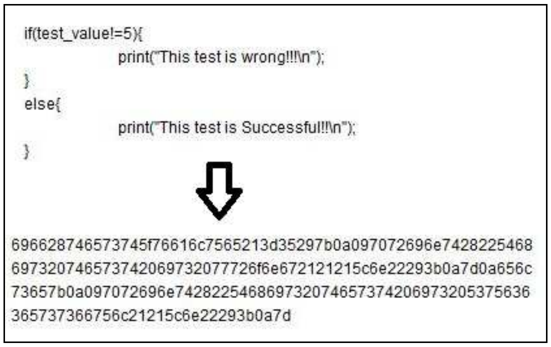
2. JavaScript Obfuscation

Java is a computer language that was first developed by Sun Microsystems in 1995 and became popular and used widely since then. Java can run on almost every platform in the Internet distributed environment. Therefore, Java can be useful for web programming. Also, developers can use a program written in Java anywhere on the network. In addition, Java has the strength of creating and using a module or applet conveniently as a part of a web page. Therefore, special work is performed to protect the output of Java, which is used frequently and widely. It is called "obfuscation", which makes the people not to understand the program code when they look at it. Developers can protect the program or ensure security when transmitting the program via communication, using JavaScript obfuscation methods. [1, 2] JavaScript obfuscation methods include variable name obfuscation and code content obfuscation.

2.1. JavaScript obfuscation methods

JavaScript obfuscation methods can be broadly grouped into 4 types: [3] the method of using ASCII values and Unicode values; using XOR operation; splitting a string; compressing a string and replacing the existing string with the meaningless string. Obfuscated JavaScript codes have no meaning in strings. Instead, codes are expressed as a disorderly set of numbers, alphabets, and special characters. These characters are used throughout JavaScript disorderly, and it takes a significantly long time for the understanding of these characters.

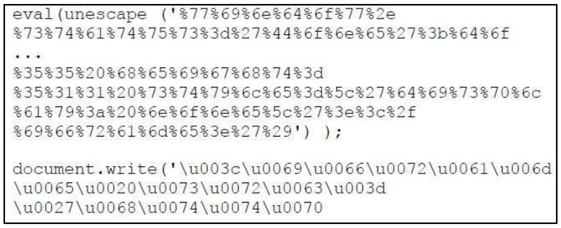
These characteristics of the obfuscation method are used to check the obfuscated JavaScript in Chapter 4. One of the major characteristics in obfuscated JavaScript is that string length increases significantly, compared with the normal string, as other characters are inserted or meaningless characters are listed to hide the meaning of the string. Therefore, the length of the string can be the criteria of checking obfuscated JavaScript. In addition, special functions are used to display meaningless strings on the web browser. Therefore, the use frequency of these functions can be compared with the one in normal JavaScript. Also, if meaningless strings are created, or strings are expressed in another format such as ASCII values or Unicode values, the particular character may be used frequently. Therefore, obfuscation can be confirmed by checking the use frequency of these characters and entropy [8]. As a result, these characteristics are used as the criteria of checking obfuscated JavaScript in Chapter 4.



```
if(test_value!=5){
                print("This test is wrong!!!\n");
}
else{
                print("This test is Successful!!\n");
}
```

```
696628746573745f76616c7565213d35297b0a097072696e7428225468
697320746573742069732077726f6e672121215c6e22293b0a7d0a656c
73657b0a097072696e74282254686973207465737420697320537563636
365737366756c21215c6e22293b0a7d
```

Figure 1. Obfuscation using ASCII values



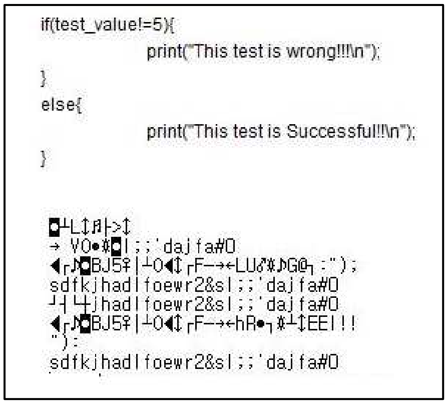
```
eval(unescape ('%77%69%6e%64%6f%77%2e
%73%74%61%74%75%73%3d%27%44%6f%6e%65%27%3b%64%6f
...
%35%35%20%68%65%69%67%68%74%3d
%35%31%31%20%73%74%79%6c%65%3d%5c%27%64%69%73%70%6c
%61%79%3a%20%6e%6f%6e%65%5c%27%3e%3c%2f
%69%66%72%61%6d%65%3e%27%29') );

document.write('\u003c\u0069\u0066\u0072\u0061\u006d
\u0065\u0020\u0073\u0072\u0063\u003d
\u0027\u0068\u0074\u0074\u0070
```

Figure 2. Obfuscation using ASCII values and Unicode values

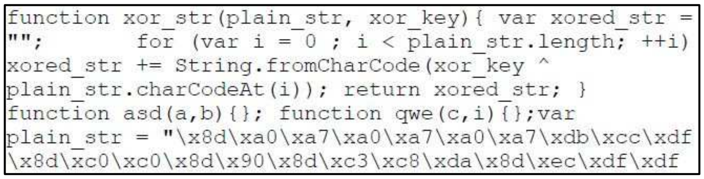
2.1.1. Methods of using ASCII values and Unicode values : This method obfuscates JavaScript by replacing the part in JavaScript codes to protect with ASCII values or Unicode values. As shown in Figure 1, changed characters are encoded into corresponding ASCII values or Unicode values, and it takes long to understand the meaning without decoding. These obfuscated strings can be converted to the original string when displayed on the web browser, using several functions such as eval, unescap, and document.write. Therefore, these functions appear more frequently in obfuscated JavaScript than the normal JavaScript code.

2.1.2 Method of using an XOR operation : The obfuscation method using an XOR operation runs in such way that the key value for an XOR operation is set first, and XOR operation is performed on strings in question with the key value. This method requires declaration of the key value – the core of the operation, and modified strings are displayed in a set of meaningless alphabets and special symbols. As shown in Figure 3, obfuscated strings cannot be understood with our naked eyes. In addition, if the key value that was used for the XOR operation is lost, it takes too much time to normalize it. Therefore, run the XOR operation again with the value set before opening the browser, before normalizing obfuscated strings.



```
if(test_value!=5){
            print("This test is wrong!!!\n");
}
else{
            print("This test is Successful!!\n");
}


◘┴L↕♬├>↕
→ VO•₮◘|;;'dajfa#O
◀┌♪◘BJ5₹|┴O◀↕┌F─→←LU♂₮♪G@┐:");
sdfkjhad|foewr2&s|;;'dajfa#O
┘┤└┼jhad|foewr2&s|;;'dajfa#O
◀┌♪◘BJ5₹|┴O◀↕┌F─→←hR•┐₮┴↕EE|!!
");
sdfkjhad|foewr2&s|;;'dajfa#O
```

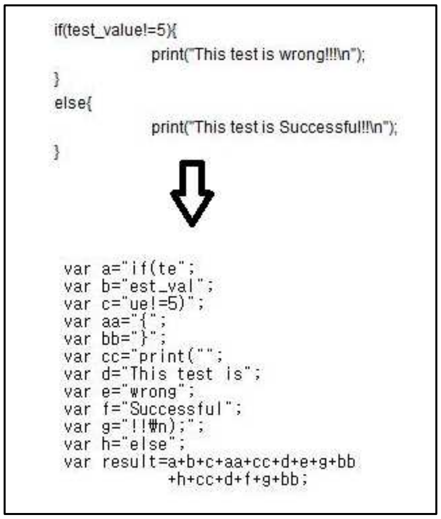


```
function xor_str(plain_str, xor_key){ var xored_str =
"";        for (var i = 0 ; i < plain_str.length; ++i)
xored_str += String.fromCharCode(xor_key ^
plain_str.charCodeAt(i)); return xored_str; }
function asd(a,b){}; function qwe(c,i){};var
plain_str = "\x8d\xa0\xa7\xa0\xa7\xa0\xa7\xdb\xcc\xdf
\x8d\xc0\xc0\x8d\x90\x8d\xc3\xc8\xda\x8d\xec\xdf\xdf
```

Figure 3, 4. Obfuscation using an XOR operation

2.1.3. Method of splitting a string : Instead of listing up a string, this method split a string into several small strings to reduce readability and mixes the order of those small strings. As shown in Figure 5, obfuscation by splitting a string doesn't change the meaning of an entire string but makes it difficult to understand. Before displayed on the browser, split strings are normalized using a + operation with the proper order, using the eval function. Or, as shown in Figure 5, a + operation is performed on strings split for obfuscation according to the order, and the results are put into the eval function to display the normal string.

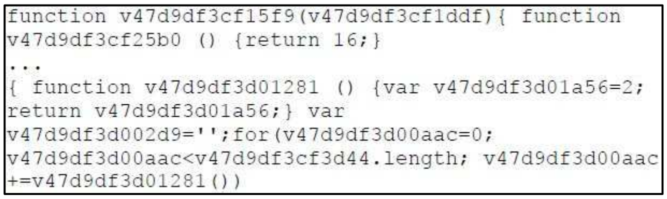
2.1.4. Method of compressing a string and replacing with a meaningless string : This method changes words or characters included in a string, in order to make it looks like a meaningless string. Strings are changed by special characters and meaningless strings. This method is similar to the method of using an XOR operation as described in 2.1.2. Each character or string is mapped to another character or string, so that the string cannot be understood. Obfuscated strings are mainly created using a tool, and basic grammatical sentence like variable declaration and function declaration can be changed to a meaningless string. The method of compressing a string and replacing with a meaningless string makes a string much longer than the non-obfuscated string.



```
if(test_value!=5){
            print("This test is wrong!!!\n");
}
else{
            print("This test is Successful!!\n");
}
```

```
 var a="if(te";
 var b="est_val";
 var c="ue!=5)";
 var aa="{";
 var bb="}";
 var cc="print("";
 var d="This test is";
 var e="wrong";
 var f="Successful";
 var g="!!₩n);";
 var h="else";
 var result=a+b+c+aa+cc+d+e+g+bb
          +h+cc+d+f+g+bb;
```

Figure 5. Obfuscation by splitting a string



```
function v47d9df3cf15f9(v47d9df3cf1ddf){ function
v47d9df3cf25b0 () {return 16;}
...
{ function v47d9df3d01281 () {var v47d9df3d01a56=2;
return v47d9df3d01a56;} var
v47d9df3d002d9='';for(v47d9df3d00aac=0;
v47d9df3d00aac<v47d9df3cf3d44.length; v47d9df3d00aac
+=v47d9df3d01281())
```

Figure 6. Obfuscating variable names, function names, and content splitting

3. Related Studies

Currently, the "Drive-by Download" attack is the main attack pattern, where the file conducting malicious behavior is downloaded without the user's awareness when the user visits the malicious web site. As a result, many studies are carried out that constrain access to the malicious web site by checking the malicious nature of the web page concerned. These studies can be broadly grouped into the static analysis method, which analyzes web site codes only, and the dynamic analysis method that checks the system change by visiting the web page concerned. Description of dynamic analysis will follow that of static analysis.

3.1. Obfuscated web site static analysis system

Obfuscated web site analysis using Malzilla [4] is most representative. Malzilla is the method of analyzing source codes sequentially after downloading them first by entering the URL of the web site to analyze. Obfuscated strings and JavaScript codes are normalized by adding the particular function (alert, eval, unescape, etc.) while decoding source codes. If decoding of obfuscated codes still results in obfuscated codes, normal JavaScript can be checked by decoding the obfuscated code again. There is a shortcoming that the analyzer should determine whether the detected JavaScript code is an attack JavaScript that induces malicious behavior.

There is another method that analyzes the pattern of web page strings. [5] This method separates obfuscated JavaScript codes from normal ones, using N-grame, entropy, and string size. If JavaScript is found to exceed the certain threshold value, the web site in question will be deemed as a malicious web site. As this method detects a malicious web site mainly by obfuscation, instead of the characteristics of the certain malicious code, it could be difficult to detect a malicious web site that contains normal JavaScript.

However, the static analysis method can find the obfuscation characteristics after analyzing source codes of the web site in question, or can analyze the web site by interpreting normal JavaScript after decoding obfuscated JavaScript. As only source codes are read without visiting the web site, this method doesn't cause direct damage by an attack. Therefore, the analysis system doesn't have to be initialized. However, it takes too long to analyze one web site, even considering the advantage of this method. That is, web site source codes should be decoded sequentially or additional functions should be inserted after downloading them, so that the analyzer can recognize the format. In addition, even though the static analysis time can be shortened by generating a signature, based on the characteristics of the malicious web site, more time can be taken to detect a variant and generate a signature, if a variant is created.

3.2. Obfuscated web site dynamic analysis system

Unlike the static analysis method, the dynamic analysis method visits the web site to see if any of the web page in question causes any change in the user's system in order to analyze malicious behavior of the web site. The web site obfuscated with JavaScript is not directly decoded. Instead, JavaScript codes are executed, and the result is analyzed to check malicious behavior.

Wepawet [6] from isecLAB visits the web site from the virtual environment system, which was developed for analysis, using the entered web site address, and analyzes the visit result. The virtual environment system analyzes whether the visited web site is obfuscated or not, and whether any malicious behavior is committed or not, and shows the analysis result. Frequency of variable and function declaration, use times of the particular function, and length of the dynamically generated code are analyzed to check the obfuscation status. In particular, the result of obfuscated JavaScript can be checked. In addition, and the result and type of malicious behavior can be directly checked by checking the change of the virtual environment system, by comparing characteristics of the particular malicious behavior code with those of analyzed JavaScript. Unlike the existing static analysis method, the content of JavaScript can be checked, even though the analyst doesn't manually look into the content of JavaScript, as the system change is checked and the result is analyzed.

In addition, MonkeyWrench[7] from G Data Softwafe AG. also analyzes the web page, using an analysis method similar to Wepawet.

The dynamic analysis method is fast and shows more accurate examination results than the static analysis method. In addition, it has the advantage of being able to analyze JavaScript codes that are generated dynamically. The dynamic analysis method can check the malicious nature of the web site in question within shorter time than static analysis. However, an environment should be created that allows execution of malicious behavior contained in the malicious obfuscated web site.

Malicious codes make an attack by exploiting vulnerabilities of the particular application or auxiliary program. Therefore, malicious behavior can be analyzed only when that particular application has been developed in a virtual environment in advance. In addition, an environment should be created in several proper ways, in order to analyze various types of malicious codes. Consequently, much time and efforts are required to build these analysis environments by malicious code. Also, dynamic analysis is performed in the virtual environment system, as the attack is made directly in the system. These virtual environment systems should be initialized as many times as web site analysis, which results in a significant amount of time consumption and inefficient use of resources.

4. JavaScript obfuscation strength check system

Most malicious web sites use obfuscated JavaScript. When obfuscated JavaScript is used, JavaScript analysis takes a considerable amount of time, and the program blocking malicious web sites can be bypassed even temporarily. Detecting these malicious websites and controlling access to these web sites becomes the conversation topic. In addition, the number of malicious obfuscated websites is gradually increasing. So, it is inefficient to use the static analysis method, as analysis takes too much time, or the dynamic analysis method that is difficult to create the environment. As a result, it is necessary to build a system that collects suspicious malicious websites in a hybrid format, which can supplement these shortcomings. The long-term purpose of this study also lies in the development of such a system. Collecting as many suspicious websites as possible and sending to the analysis system is the major purpose of the JavaScript obfuscation strength check system, which is proposed by this paper. Therefore, as many obfuscated web sites as possible should be detected among all websites. That is, reducing the number of non-detectable websites is the major objective of the JavaScript obfuscation strength check system. The additional objective is to provide a probability that is similar to a false detection ratio of the existing dynamic analysis system.

Obfuscated JavaScript looks like a list of meaningless strings, if obfuscated codes are not decoded. For that reason, when people look at these obfuscated JavaScript with naked eyes, they may see the strings with unidentifiable meanings. As a result, obfuscation can be checked with ease. However, it would not be easy to check JavaScript that was obfuscated by machine. As a machine cannot understand the meaning of the particular string, it cannot distinguish between meaningful strings and meaningless strings. If we assume that a machine is configured in such way that it can understand the meaning of the string, and check the obfuscation status of the string, understanding the meaning of the string and developing the database would take considerable amounts of time and resources. Therefore, we propose the method of checking obfuscated JavaScript, using the characteristics of obfuscated JavaScript, instead of the analysis focused on the meaning of strings. With focus on the characteristics of obfuscated JavaScript as described in Chapter 2, we propose the JavaScript obfuscation strength check system, which becomes the major module of the hybrid type analysis system that combines strengths of the static and dynamic analysis system as described in Chapter 3.

4.1. Criteria of obfuscation strength check

Studies are conducted that search obfuscated malicious attack websites using web page string patterns. [5] These studies select N-Gram, entropy and word size as the primary items to be scrutinized. Obfuscation of the websites is checked using these criteria. The JavaScript obfuscation strength check system proposed by this paper presents more detailed criteria than the existing detection criteria, and takes a different approach than existing scrutinizing methods. [5] The primary criteria that are scrutinized include string length, density, frequency of the particular function, frequency of special characters, and entropy value. These criteria are used to measure the obfuscation strength of an obfuscated website. In addition, malicious behavior is checked by directly examining malicious behavior characteristics, instead of interpreting JavaScript which is usually obfuscated by the characteristics of malicious codes.

4.1.1. Density : Length of a string used in the web source code is selected by the density of the JavaScript obfuscation strength check system. Generally, the obfuscated string is longer than the existing string. The length of a string tends to increase while encoding a string or replacing it with a meaningless string. Therefore, if the length of a single string is longer than 200 characters, when measured, the website in question is deemed a malicious obfuscated website. A single string refers to the listed string that is separated by a special character (<,>,“,‘,[,],{,}, etc.), unlike other studies on malicious obfuscated website detection, which use string patterns. [5] Existing studies set one string based on a blank character. By this method, normal strings like the URLs of normal websites can be interpreted as malicious behavior. Therefore, the criteria are set to be stronger than blank characters, in order to prevent false detection. Existing studies propose 350 characters as the criteria for determining malicious obfuscated websites, whereas this study set “over 200 characters” as the criteria of detecting a single string to prevent false detection. Even though there is a possibility of non-detection, the number of non-detected websites can be reduced, by setting the detailed examination criteria with focus on malicious behavior codes using frequency and entropy, which will be described below.

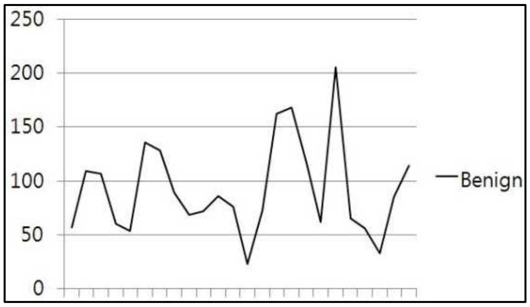


Figure 7. Maximum length of a single string in a normal website

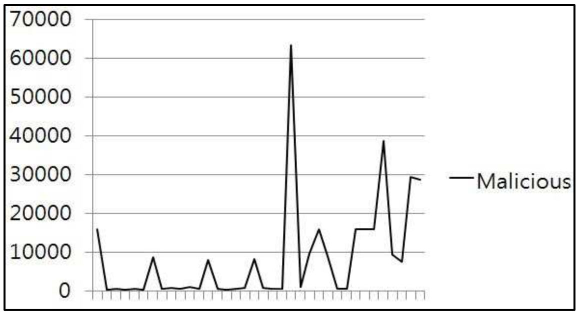
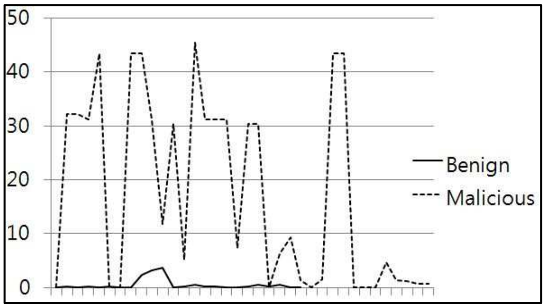


Figure 8. Maximum length of a single string in a malicious website

4.1.2 Frequency : The JavaScript obfuscation strength check system uses the frequency of the particular function, encoding mark, and % symbol occurrence, as the detailed check items to check frequency.

One of the major characteristics in obfuscated JavaScript is that the part is also included, which normalizes obfuscated strings using particular functions such as eval, replace, and fromCharCode. Therefore, the proposed system checks the possibility of obfuscation, by checking the use frequency of these functions. Besides, those which are considered dynamic functions are also checked, such as document.write and document.create.Element, as obfuscation can be released through dynamic generation.



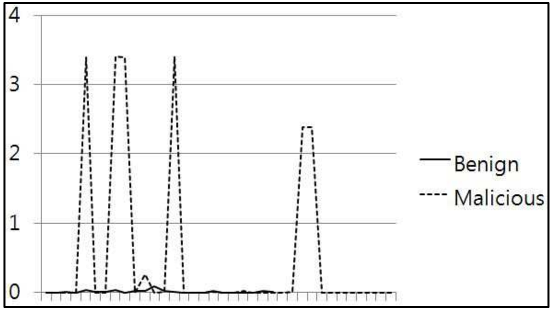


Figure 9. Frequency of the particular function

Figure 10. Frequency of % symbol besides the HTTP link (%)

To calculate the frequency of a particular function, the total use times of the particular function is divided by the number of total strings. The reason that the total use time is divided by the number of total characters in a web source instead of the total of JavaScript strings is that it reflects the total size of the web page. Another reason that the total functions used in JavaScript are divided by the total number of the particular function is as follows. If divided by the total number of used functions, there is no big frequency difference between normal JavaScript and obfuscated JavaScript. In particular, the string can be split, or the length of a single string can be reduced (one of the obfuscated JavaScript characteristics), so that the string can be combined later, using a + operation. [3] The number of the particular function in JavaScript using this obfuscation method increases more than other JavaScript using other obfuscation methods. Therefore, frequency varies significantly, if obfuscated with the same content. However, if divided by the total number of strings, the use frequency of the particular function doesn’t change much, as the number of strings increased by obfuscation is greater than the increase in the use times of the particular function, or the number of strings is already large enough. Therefore, it is more effective to calculate frequency, using the number of used strings, instead of the total number of functions, in order to scrutinize malicious obfuscated JavaScript. Generally, obfuscated JavaScript shows more occurrence frequency of a particular function than normal JavaScript.

An encoding mark is another frequency check item. One of the characteristics of malicious behavior codes is that these codes deteriorate readability using ASCII characters, Unicode, and number system conversion, instead of a general string; therefore, the fact that these encoding marks appear more frequently than in normal JavaScript can be utilized. Frequency can be calculated by dividing the total use times of each encoding mark by the number of characters used in JavaScript. Generally, malicious obfuscated JavaScript shows a higher frequency than normal JavaScript.

Frequency of the “%” symbol – the last frequency examination criteria, can be divided into the frequency inside the HTTP link and the frequency outside the HTTP link. The occurrence frequency of the “%” symbol outside the HTTP link can be checked, using the fact that an encoded URL generally uses more % symbols in the HTTP link.

Studies on malicious obfuscated website detection using string patterns [5] check the use frequency of each character, using the N-Gram method. For this method, the use of various characters is utilized, which are the characteristics of obfuscated JavaScript. However, it has the shortcoming of a high probability of false detection, as examination is performed with focus on the individual characters e.g. x, u, and @, instead of characters that conduct specific behavior.

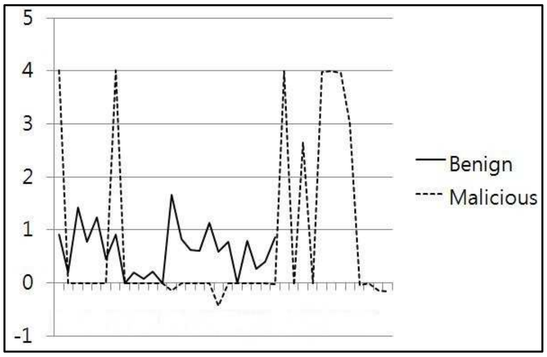


Figure 11. Difference between total Java entropy and average Java block entropy

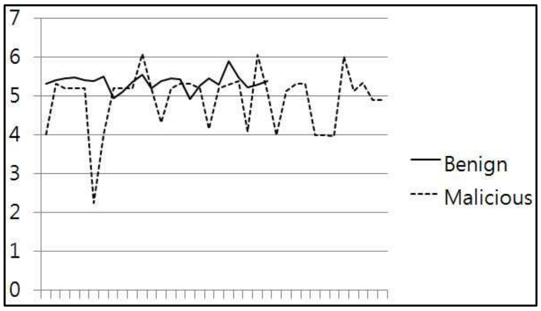


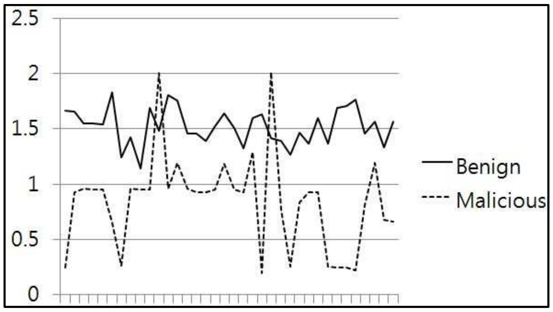
Figure 12. Total JavaScript entropy

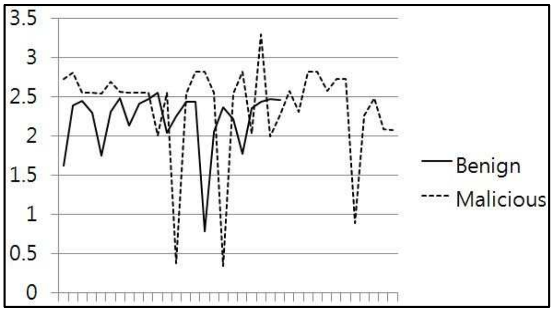
4.1.3 Entropy[8] **:** The JavaScript obfuscation strength check system provides 8 detailed check criteria for obfuscation strength check, such as entire JavaScript entropy, special character entropy, and variable and function name entropy.

The existing website entropy measurement method detects the malicious nature of the website, by measuring entropy based on the entirety of a web site's sources. The proposed system puts more emphasis on JavaScript entropy of the website in question, rather than entropy of the entire website source; and in addition, the entire entropy of JavaScript, and the code beginning with <script **> and ending with </script> as a Java block. Therefore, the average Java block entropy can be obtained by calculating entropy of each Java block, and dividing the total sum by the number of Java blocks, and the entire JavaScript entropy is the one that is calculated by setting a set of Java blocks as a big set.

The proposed system has two detailed examination criteria, using entire JavaScript entropy. Obfuscation can be checked, using the difference between the normal JavaScript entropy value and the obfuscated JavaScript entropy value. Obfuscation is also checked, using the difference between the entire JavaScript and average Java block entropy. Generally, the entire JavaScript entropy value of obfuscated JavaScript shows a smaller value than normal JavaScript. In addition, the difference between the average Java block entropy shows a value smaller than 0, or a significantly larger value than normal JavaScript.

Five detailed examination criteria can be obtained, using special character entropy. Entropy of the most frequently occurring character among special characters will be set as the criteria. Obfuscated JavaScript uses particular special characters frequently. Therefore, there can be some values for which the entropy value of the maximum use special character is larger than the normal one. Three criteria can be set, using the entropy of the special character based on the used special character, special character





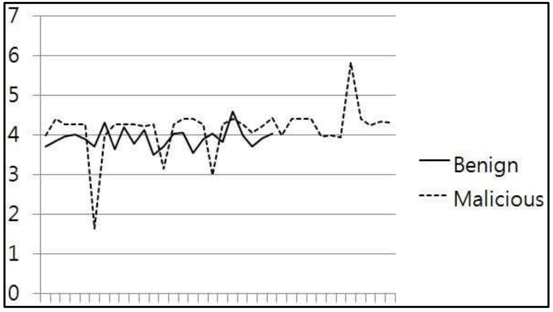
entropy in the total character set used by JavaScript, and the difference between the calculated two values.

Figure 13. Special character entropy in the entire set

Figure 14. Difference of special character entropy(special character set – entire character set)

Special character entropy in the total used character set shows lower value in obfuscated JavaScript and normal JavaScript. Due to this characteristic, the difference between entropy in the used special character set and entropy in the entire character set increases further, and the value of obfuscated JavaScript is larger than normal JavaScript. The difference can be calculated for general characters, not special characters, and can be set as another detailed check criteria. Generally, obfuscated JavaScript shows the lower value in the difference between general character entropy – the last examination criteria, as obfuscated JavaScript uses various characters.

Entropy using variable and function names is set as the last examination criteria. There are some cases in which variable names and function names in obfuscated JavaScript are also obfuscated. Therefore, entropy of each variable name and function name with focus on these characteristics can be measured, and the average of the total value can be used to calculate the entropy value. Many characters are not used for function and variable names in normal JavaScript, and many characters are used in duplicate. Therefore, the entropy value is generally lower in normal JavaScript.



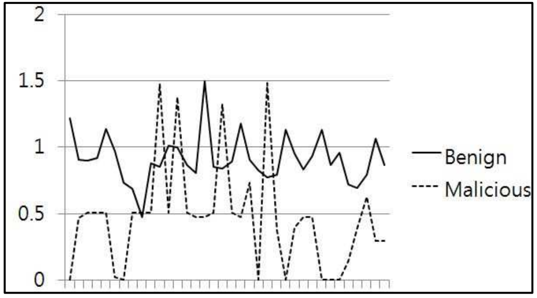


Figure 15. General character entropy in the entire set

Figure 16. Difference of general character entropy(general character set – entire character set)

4.2. Characteristics of the obfuscation strength check system

Considering that the string length detection value is between 256 and 300 according to the existing study results [5, 6], the proposed system determines malicious obfuscated websites with lower values. To reduce the non-detection ratio of malicious obfuscated websites, which is the major objective of obfuscation strength check systems, density is selected as a core obfuscation strength factor. Frequency and entropy results are selected as the factors that show obfuscation strength in more detail. And, each detailed check item is assigned a different weight, depending upon the characteristics of the malicious behavior code and the importance of the obfuscation characteristic. The proposed system can indicate how much danger the website poses in later analysis, by marking malicious obfuscation strength, based on the measured scores after website check. In addition, the proposed JavaScript obfuscation strength check system can reduce the web site check time, because it reads the website source only to check for malicious behavior, thereby compensating for the shortcomings of the existing dynamic website analysis system. In addition, the probability of non-detection and false detection of malicious obfuscated websites is reduced, enabling to check obfuscation and malicious behavior by checking the web source code characteristics only, unlike the existing static website analysis which requires longer analysis time.

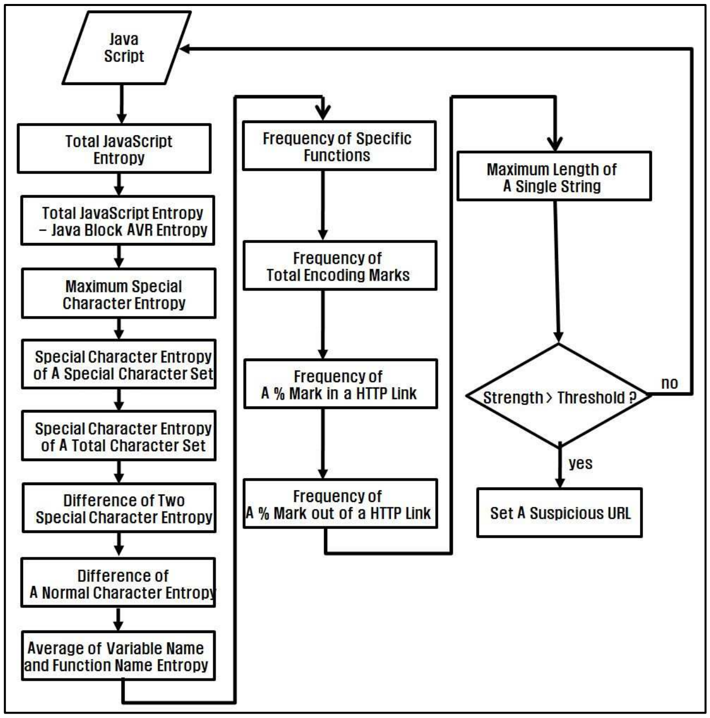

Java Script
Total JavaScript Entropy
Total JavaScript Entropy – Java Block AVR Entropy
Maximum Special Character Entropy
Special Character Entropy of A Special Character Set
Special Character Entropy of A Total Character Set
Difference of Two Special Character Entropy
Difference of A Normal Character Entropy
Average of Variable Name and Function Name Entropy
Frequency of Specific Functions
Frequency of Total Encoding Marks
Frequency of A % Mark in a HTTP Link
Frequency of A % Mark out of a HTTP Link
Maximum Length of A Single String
Strength > Threshold ?
no
yes
Set A Suspicious URL

Figure 17. JavaScript obfuscation strength check system

4.3. Comparison with existing systems

Wepawet [6] and MonkeyWrench[7] – dynamic malicious websites, and the detection system provided by Google are compared. Examination target websites are selected by referring to the malicious domain reporting site, and normal websites are also selected for examination as comparison targets.

Selected target websites are compared with the existing dynamic analysis system. The major objective of the JavaScript obfuscation strength check system – reducing the ratio of non-detected websites, was achieved. As shown in Table 1, the proposed system shows a lower non-detection ratio than the dynamic analysis system. Also, the analysis hours are reduced, and a similar level of false detection ratio is achieved. These results indicate that the proposed system provides a method of detecting as many malicious obfuscated websites as possible.

Table 1. Comparison of the proposed system with existing systems

	Proposed system	Weapwet	MonkeyWrench	Google System
Non-detection ratio	3.84%	12.82%	11.78%	10.15%
False detection ratio	12.13%	4.12%	14.68%	12.57%
Analysis failure (took longer than 2minutes)	0%	7.85%	19.89%	21.68%

Table 2. Major detection values of JavaScript obfuscation strength check

Assessment item	Average		Max. value		Min. value	
	Normal	Obfuscation	Normal	Obfuscation	Normal	Obfuscation
String length	91.9167	10191.7	205	63363	23	367
Frequency of particular function	0.53299	18.1484	3.66	45.454	0	0
Java entropy difference	0.6144	0.82288	1.66098	4.01651	0	-0.4246
Entire Java entropy	5.34948	4.89889	5.88991	6.06263	4.90447	2.24381
Special character	1.526275	0.84121	1.83064	2.00827	1.14211	0.1976

entropy (entire set)						
Non-special character entropy difference	0.91492	0.46986	1.49404	1.48538	0.47757	0.000267

5. Conclusion and Further Studies

This paper proposed a malicious behavior suspected website detection system based on JavaScript obfuscation that analyzes JavaScript density, frequency, entire JavaScript entropy, and entropy of each characteristic. If the analysis time is short when checking malicious behavior websites, which are gradually on the rise, the non-detected malicious website ratio is smaller, and more websites can be detected. However, a hybrid type analysis system should be developed that compensates for the shortcomings of the existing dynamic analysis system, in order to use the JavaScript obfuscation strength check system efficiently. Therefore, more studies are required to develop a hybrid type analysis system. Also, a study on the website analysis method needs to be carried out, using a method other than JavaScript.

Acknowledgement

This work was supported by the IT R&D program of MKE/KEIT. [10035427, The Development of Automatic Analysis and Malicious Site Detection Technology Against Malware]

References

[1] Jien-Tsai Chan, Wuu Yang, "Advanced obfuscation techiques for Java bytecode", The Jouarnal of Systems and Software 71, August 2002

[2] Zhanyong TANG, Xiaojiang CHEN, Dingyi FANG, Feng CHEN, "Research on Java Software Protection with the Obfuscation in Identifier Renaming", 2009 Fourth International Conference on Innovative Computing, Information and Control, 2009

[3] Kolisar, "WhiteSpace:A Diffrent Approach to JavaScript Obfuscation", DEFCON 16, August 2008

[4] Malzilla.org Rhino:JavaScript for Java. http://www.mozilla.org/rhino

[5] YoungHan Choi, TaeGhyoon Kim, SeokJin Choi, "Automatic Detection for Javascript Obfuscation Attacks in Web Pages through String Pattern Analysis", International Journal of Security and Its Applications, 4(2), pp.13-26, April 2010

[6] Marco Cova, Christopher Kruegel, Giovanni Vigna, "Detection and Analsis of Drive-by-Download Attacks and Malicious JavaScript Code", Management of Computing and Information Systems, April 2010

[7] Armin Bűscher, Michael Meier, Ralf Benzműller, "Throwing a MonkeyWrench into Web Attacks Plans", International Fedration for Information Processing 2010, pp.28-39, 2010

[8] Robert M.Gray, "Entropy and Information Theory", July 16 2009

Authors

Byung-Ik Kim

Byung-Ik Kim received the B.S. degree in Information and Computer Science from Ajou University, in February 2010.

He is currently working as Research Associate of Korea Internet & Security Agency
His research interests include computer security.

Chae-Tae Im

Chae-Tae Im received the B.S. degree in Computer Science from Chungnam Nationl University, in February 2000, and the M.S. degree in Computer Science fromPohang University of Science and Techinology, in 2003,
He is currently working as Senior Research Associate of Korea Internet & Security Agency

Hyun-Chul Jung

Hyun-Chul Jung received the B.S. degree in Statistical Computing from University of Seoul, in February 1996,
and the M.S. degree in Computer Science from Kwangwoon University, in August 1998,
He is currently working as Director of Korea Internet & Security Agency
His research interests include network security.