Journal of Universal Computer Science, vol. 21, no. 3 (2015), 369-383
submitted: 30/6/14, accepted: 28/2/15, appeared: 1/3/15

Polymorphic Malicious JavaScript Code Detection for APT Attack Defence

Junho Choi
(Division of Undeclared Majors, Chosun University
Gwangju, South Korea
xdman@chosun.ac.kr)

Chang Choi
(Department of Computer Engineering, Chosun University
Gwangju, South Korea
enduranceaura@gmail.com)

Ilsun You
(School of Information Science, Korean Bible University
Seoul, South Korea
ilsunu@gmail.com)

Pankoo Kim
Corresponding author
(Department of Computer Engineering, Chosun University
Gwangju, South Korea
pkkim@chosun.ac.kr)

Abstract: The majority of existing malware detection techniques detects malicious codes by identifying malicious behavior patterns. However, they have difficulty identifying new or modified malicious behaviors; consequently, new techniques that can effectively and accurately detect new malicious behaviors are crucial. This paper proposes a method that defines the malicious behaviors of malware using conceptual graphs that are able to describe their concepts and the relationships among them and, consequently, infer their malicious behavior patterns. The inferred patterns are then learned by a Support Vector Machine (SVM) classifier that compares and classifies the behaviors as either normal or malicious. The results of experiments conducted verify that the proposed method detects malicious codes more efficiently than conventional methods. In the experimental results, it exhibits a better detection rate than that of malicious code detection methods that rely solely on the signature based approach. This suggests that the proposed method is not only suitable for detection of malicious codes, but is also more efficient than other detection methods as it combines the advantages of more than two malicious code detection methods.

Key Words: Conceptual graph, Malicious code detection, APT attack defence
Category: D.4.6, I.2.6, K.6.5

1 Introduction

Recently, Advanced Persistent Threat (APT) attacks have become an issue in Information Security. An APT attack is a set of stealthy and continuous computer hacking processes, often orchestrated by humans targeting a specific entity. APT attacks usually target organizations and nations for business or political motives. Further, APT processes require a high degree of covertness over a long period of time.

In order to circumvent APT attacks, integrated detection and monitoring from the routing section, rather than an individual detection policy for each client is needed. Conventional approaches to countering APT attacks are of two types: security solutions based on the network, and interception of spreading malware that occur in a repetitive pattern. These approaches rely on methods such as URL blacklists and signatures. However, by definition, these approaches are ineffective against dynamic attacks that exploit zeroday vulnerabilities. Various techniques can result in a decrease in detection rate, resulting in attempts to stop APT attacks proving ineffective [Dell SecureWorks, 12], [Giura and Wei, 12].

A case in point is the JavaScript language, which is used extensively on the Web. Many attack methods hide malware site URLs and exploit JavaScript code. Recently, these attempts have increased gradually, resulting in a variety of file types, including JavaScript, being vulnerable [Elshoush and Osmank, 11], [Laskov and Srndic, 11].

This paper proposes a new method of detecting malicious JavaScript codes. The majority of existing malware detection techniques examines the character string signature or the behavior pattern in order to distinguish between normal programs and malware. However, with these techniques, finding new malicious codes or their variants is difficult. Furthermore, as the use of signature based systems to detect scripts for malicious behaviors increases, diverse techniques are being devised to counter them. We propose a method that infers relationships among codes and analyzes heir meanings using conceptual graphs. The proposed method analyzes malicious code distribution patterns and expresses their related characteristics via conceptual graphs and patterns. The patterns are then learned by a Support Vector Machine (SVM), a popular machine learning method.

The remainder of this paper is organized as follows: Section 2 discusses conceptual graphs and the Conceptual Graph Interchange Format (CGIF). Section 3 outlines how features are extracted and patterns created for static analysis of malicious scripts. Section 4 discusses SVM learning and classification of malicious patterns. Section 5 presents and evaluates the results of experiments conducted. Finally, Section 6 concludes this paper and outlines future work.

2 Related works

2.1 Advanced persistent threat (APT) attacks

Generally, an advanced persistent threat (APT) attack means a new attack form for damage to specific target such as information hijacking, system failure and so on. An object of APT attack can be divided into three things. The first is to build organization based on the information technology for information hijacking. The second is to weaken and hinder a target's mission. The final is to prepare the foundation for APT attack in the future.

The APT attack is defined as the four steps of preparation, internal network penetration, internal activities and achievement of goal. The preparation step is to prepare an attack such as information gathering, information analysis, web page modulation, preparing command and control (C&C) server and so on. Actually, this step cannot detect the APT attack because of the APT attack is prepared in the external network[Legg et al., 13],[Shen et al., 14].

The next step is internal network penetration and various methods are used such as sending malicious e-mail, modulated web bulletin board connection, modulated software update server connection and so on [Wei and Erik, 09]. The internal activities step is the gathering information of targets internal IT infrastructure such as, connection between zombie PC and C&C server, malicious code download, sending malicious code to zombie PC, additional vulnerability gathering, authorization acquisition for database connection and so on. Finally, the goal achievement step is performed information hijacking and destroying IT infrastructure using malicious code [Ajay, 12], [Liu et al., 13].

2.2 Existing malware detection techniques

Malicious malware exist in many different forms. Viruses, worms, trojans, and adware are the most common categories. However, each category also comprises various types of threats, with equally as many different methods of combating malware. The majority of conventional anti malware scanners combine several of these techniques[Christodorescu and Jha, 03], [Shahzad and Lavesson, 13].

Malicious code detection solutions traditionally strongly relied on signature based scanning, also referred to as scan string based technologies. In this method, the signature based scan engine searches within given files for the presence of certain strings and, if the predefined strings are found, certain actions such as alarms are triggered. However, signature based scanning only detects known malware and is therefore ineffective against new attack mechanisms [Kruegel et al., 09].

Heuristic scanning is similar to signature scanning, with the exception that instead of looking for specific signatures, heuristic scanning searches for certain instructions or commands within a program that are not found in typical application programs. As a result, a heuristic engine is able to detect potentially malicious functionality in new, previously unexamined, code such as the replication mechanism of a virus, the distribution routine of a worm or the payload of a Trojan. However, heuristic methods are time consuming [Ho et al., 13].

Conventional malware protection methods are becoming increasingly incapable of providing protection against some of the latest threats. Network based malware protection systems are therefore becoming important advanced threat detection mechanisms, and conceptual access to malicious code is also crucial [Likarish et al., 09].

2.3 Conceptual graph and conceptual graph interchange format (CGIF)

A conceptual graph is a knowledge representation language that integrates several semantic networks, and has the expressive power of logically concise and natural languages using schematics [Hensman, 04], [Karalopoulos et al., 04]. It can describe meanings in a form that can be easily understood by humans and available natural language processing systems on the computer. For example, a sentence such as “Smith is going to New York by bus” can be expressed as the conceptual graph shown in Figure 1.

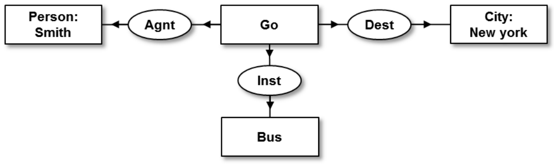

Figure 1: The example of Conceptual Graph

The rectangle in Figure 1 represents concepts, the oval indicates the relationships between the concepts, and the nodes are connected by indicators. Thus, “Agnt”, “Dest”, and “Inst”signify the relationship between the concepts and “Smith”, “New York”and “Bus”refer to the concept of each node. The expression “Person: Smith”means that “Smith”is an instance of the concept “Person”.

In addition, a conceptual graph can be converted to CGIF, an extended graphic notation called Backus Normal Form or Backus Naur Form (BNF). Table 1 lists the conceptual graph notation converted into a linear notation, and an example of CGIF conversion from a conceptual graph based on the linear notation [Baget, 03], [Zhong et al., 02].

Table 1: Examples of Linear Notation and CGIF Expression of Conceptual Graph

Linear Notation of Conceptual Graph	CGIF Expression of Conceptual Graph	
$[Go]-$	$[City : NewYork]$	$(Dest?x1NewYork)$
$(Agnt) \to [Person : Smith]$	$[Bus : *x2]$	$(Inst?x1?x2)$
$(Dest) \to [City : NewYork]$	$[Person : Smith]$	$(Inst?x1?x2)$
$(Inst) \to [Bus]$	$[Going : *x1]$	

3 Conceptual graph and SVM using malicious script code analysis

After collecting normal script codes and malicious script codes, it is necessary to create a conceptual graph by conceptual analysis of the ability of the attack code to attack the source codes vulnerability. Then, tokenization of the code is carried out through static analysis of the malicious script codes collected [Mishne and Rijke, 04], [Choi et al., 11].

The relationship between the tokenized codes and the concepts can be created using the conceptual graph, which is then converted into a predefined pattern through CGIF [Zhang and Yu, 01]. Patterns of malicious script codes are generated as the frequency of CGIF codes for malicious behaviors is checked, and then stored in a database. The stored malignant patterns are used to create SVM training data by matching the CGIF converted from the normal scripts with those converted from the malicious scripts, and also for SVM learning. After the CGIF script codes that are to be classified are created, they are matched with the malicious patterns to form test datasets for SVM classification and, based on the dataset, scripts codes are SVM classified for malicious script detection. Figure 2 shows the overall flow of the malicious code analysis procedure described above.

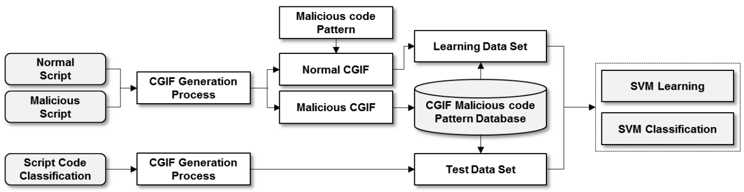

Figure 2: Processing of Malicious Script Analysis using Conceptual Graph and SVM

3.1 Definition of JavaScript Code Concept and Relation

Malware authors create and distribute malicious codes in JavaScript because most web services are based on JavaScript. As a result, malicious JavaScript codes have been increasing every year [Cova et al., 10], [Fredrikson et al., 10]. Figure 3 shows a sample malicious script code created with JavaScript. The code is a malicious script that induces users to access another malicious web page and to download and run malicious code (real.swf) using vulnerability existing in Flash Player.

```
<script type = "text/javascript">
function whQd8() {
        var pcss = navigator.userAgent.toLowerCase();
        var kxkx = deconcept[ "SWFOb" + "jectU" + "til"][ "getPlay" + "erVer" + "sion"]();

        if (((kxkx['major' ] == 10 && kxkx[ 'minor'] <= 3) && kxkx[ 'rev' ] <= 183 || (kxkx['major' ] == 11 && kxkx['minor' ] <= 1
                && kxkx[ 'rev' ] <= 102 && ((pcss.indexOf( 'msie 6.0')>0) || (pcss.indexOf( 'msie 7.0' )>0) || (pcss.indexOf('msie' ) == -1)))))
        {
                document.writeln( "<iframe src=ff.html><\/iframe>" );
        }
        else
        {
                var UaYcKzD2 = window.navigator.userAgent.toLowerCase();
                                ⋮
                kaixin.code = "GondadExp.Ohno.class";

                kaixin.setAttribute( "xiaomaolv" , "http://slet.xvozz.com/pic/avi36.jpg " );
                kaixin.setAttribute( "bn" , "woyouyizhixiaomaolv" );
                kaixin.setAttribute( "si" , "conglaiyebuqi" );
                kaixin.setAttribute( "bs" , "748" );
                document.body.appendChild(kaixin); } } } }

                whQd8(); }
< / script>
```

Figure 3: An Example of Malicious JavaScript Code

In Figure 3, the JavaScript code with the malicious behavior, "navigator.user Agent.toLowerCase()," is a function that obtains the user agent information used by the browser. On ascertaining the version of the web browser the user is using, the JavaScript script obtains the vulnerable part of the browser by accessing the web associated with the corresponding vulnerability part using the iframe method of the document.write() class. The concepts in the source code are hierarchically classified using the components used in programming languages in order to express the malicious JavaScript code as a conceptual graph, as shown in Figure 4. The component of each hierarchy is defined as the concept of the source code.

Concepts are defined as shown in Table 2 based on the hierarchy classified above. The relationship between the concepts within the source code is defined as shown in Table 3. For example, the grammatical concept "Procedure" indicates that it is related with the relationship Condition, Argument [Gregoire, 09].

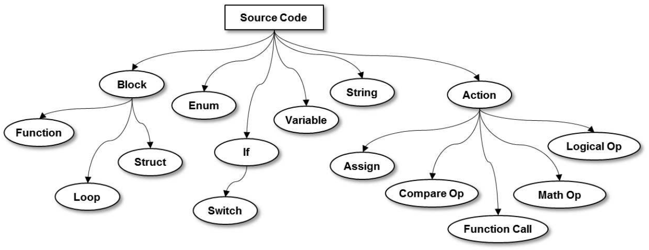

Figure 4: Hierarchy of Components in Programming Language

Table 2: Definition of Source Code Concept

Concept		Description
Procedure		A series of task sequences and processes executed to solve a problem
Statement	Conditional	Sentence used to control program execution in different ways depending on the given condition
	Loop	Program sentence to execute a series of given commands repeatedly
	Error	Operation carried out in a way different from the way predetermined for that operation
Operator	Comparison	Task of comparing the sizes of two input data
	Logical	Operation to create the result of being true or false, by applying logical operators to logical variables
	Arithmetic	Arithmetic operations for numeric data such as actual numbers and integers
	. . .	

3.2 Definition of Malicious Code Pattern and Conceptual Graph Expression

Codes for malicious behaviors in the JavaScript code are expressed according to the previously defined concepts and relationships, as shown below:

```
iframe code(width,height value (0 or 1)
document.writeln or document.write
eval(jsString)
                    :
navigator.userAgent.toLowerCase()
```

Table 3: Definition of Source Code Relationship

Relationship	Definition	Relationship Condition	
		High-level Concept	Low-level Concept
Condition	Condition for jump syntax	Conditional, Loop	Statements, Operator, Assign, Procedure, (-Call), String, Variable
Contains	Concept that contains another concept	*	*
Comment	Comment	*	String
Return	Concept that returns a value	Function, Method	Function, String, Variable
(* : Set of concepts, which contains all concepts and relationship conditions)			

In order to analyze for methods of attack to carry out certain malicious behaviors or damage security products, it is necessary to create concepts of malicious codes and the relationship among the codes, express them in a conceptual graph, and then convert the graph to a canonical format applicable to the system.

Figure 5 shows a conceptual graph of a malicious code that uses "JS/Redirect". It is the conceptual graph of the part of the code that verifies the vulnerable section of flash player and downloads a piece of malicious code that then executes another malicious code. JS/Redirect is a typical piece of malicious code in JavaScript. The conceptual graph in Figure 5 contains "Statements"(part A) for carrying out a malicious behavior, and "Methods"(part B) for downloading and executing the malicious script using the statements. According to the conceptual graph, the malicious code uses "Statements"to check the vulnerability to be used for a malicious behavior, and induces downloading and execution of the malicious code using the "Method: iframe"concept. When various forms of malicious codes are expressed in conceptual graphs using the method described above, it is possible to detect conceptual malicious behaviors even if their sources are modified or new malicious codes are generated.

4 Learning malicious script code patterns using SVM

4.1 CGIF conversion of conceptual graph

In order to create malicious script code patterns, it is necessary to translate the codes related to the malicious behaviors into a conceptual graph, and then

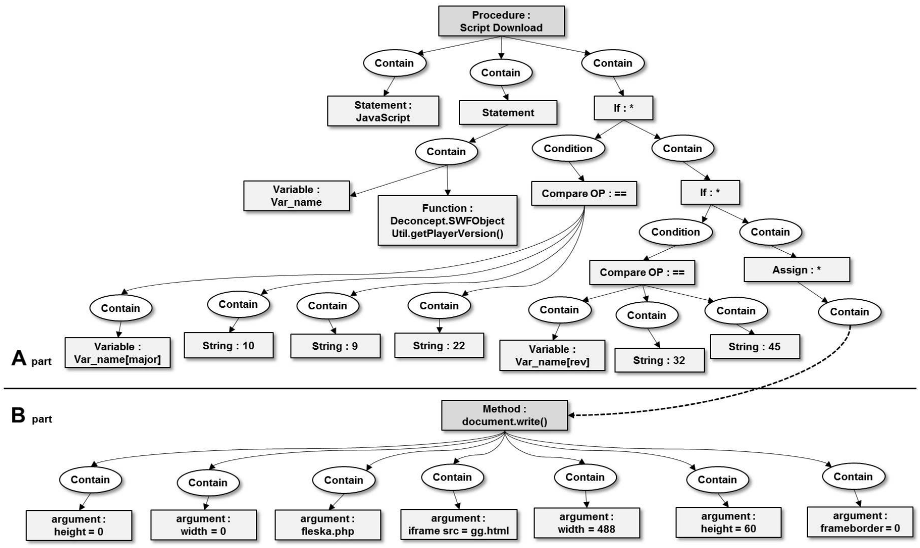

Figure 5: A Conceptual Graph for Malicious Script Code

convert the conceptual graph into CGIF. Table 4 shows an example of CGIF, in which "[ ]"refers to the concept and "*"refers to the concept type. The symbol ":"is used to distinguish between concepts and relationships, "()"shows the relationship, and "?"expresses the relationship between the concepts involved.

Table 4: An Example of CGIF converted from Conceptual Graph

01 :	[Else_if: *x1]	14 :	[argument: width=468]
02 :	[Variable: Var_name[major]]	15 :	[argument: height=60]
03 :	[String: 10]	16 :	[argument: framebord=0]
04 :	[argument: frameborder=0]	17 :	[argument: fleska.php]
05 :	[Compare_OP: ==]	18 :	[String: 12]
06 :	[Assign: *x2]	19 :	[Procedure: Script Download]
07 :	[__: *x3]	20 :	[Statement: JavaScript]
08 :	[Variable: Var_name[rev]]	21 :	[Statement: *x4]
09 :	[String: 22]	22 :	···
10 :	[Method: document.write]	23 :	···
11 :	[argument: iframe src=gg.html]	24 :	(contain ?x4 Var_name)
12 :	[argument: width=0]	25 :	(contain ?x2 document.write)
13 :	[argument: height=0]	26 :	(contain ?x6 ?x2)

4.2 Malicious script code pattern generation

Malicious scripts are analyzed and the codes related to malicious behaviors are extracted from the converted CGIF codes. The extracted CGIF codes are defined as malicious code patterns, and stored in a malicious pattern database. Then, by matching the CGIFs of normal script codes and those of the malicious codes to be classified, SVM learning datasets and test datasets are created.

Malicious behavior patterns are determined based on the malicious attack techniques known to date and the codes that attempt malicious behaviors. Then, the frequency with which each malicious pattern appears in normal scripts and malicious scripts is calculated. The malicious code pattern with a higher frequency in malicious scripts and a lower frequency in normal scripts is recorded on a higher level index. Using the finalized ranking list of each malicious code pattern, the malicious CGIF patterns are generated after the patterns of less frequent malicious codes (lower rank); that is, the malicious patterns and normal patterns with less frequencies, are excluded. The malicious CGIF patterns are generated based on the frequencies of normal patterns and malicious patterns.

4.3 Learning script code pattern using SVM

The codes related on malicious activity are extracted to use the converted CGIF codes through malicious script analysis. The extracted CGIF codes are defined as malicious code pattern. And then, the data set (SVM learning data set and test data set) is building through matching between CGIF of general script codes and CGIF for classification.

The malicious activity pattern is defined based on the well-known malware codes and attack methods. The frequency of malicious activity is calculated between general scripts and malware scripts. If malicious pattern indicates low frequency of general scripts and high frequency of malicious scripts, this pattern can be defined the malicious activity pattern and it is recorded the malicious pattern list. Table 5 is malicious CGIF pattern based on frequency between general and malicious pattern. Actually, malicious patterns are consist of concept and relation for malicious activity.

Many of the existing classification results of the SVM classification algorithm have proved highly accurate. It is used predominantly in the field of pattern recognition, and is most successful in the area of document classification [Wang and Chiang, 09], [Chen et al., 09]. In this paper, a malicious code detection approach is proposed in which conceptual graphs are created using the concepts and relationships of the source codes and malicious code SVM learning conducted. Figure 6 shows the process of generating an SVM dataset through matching of the malicious code patterns in CGIF and normal script codes or script codes subject to classification.

Table 5: Malicious code patterns

Pattern Index	Malicious Pattern
1	[String: 1]
2	[Variable: Var_name[major]]
3	[String: 10]
...	...
60	Contain ?x2 gaobumingbai())

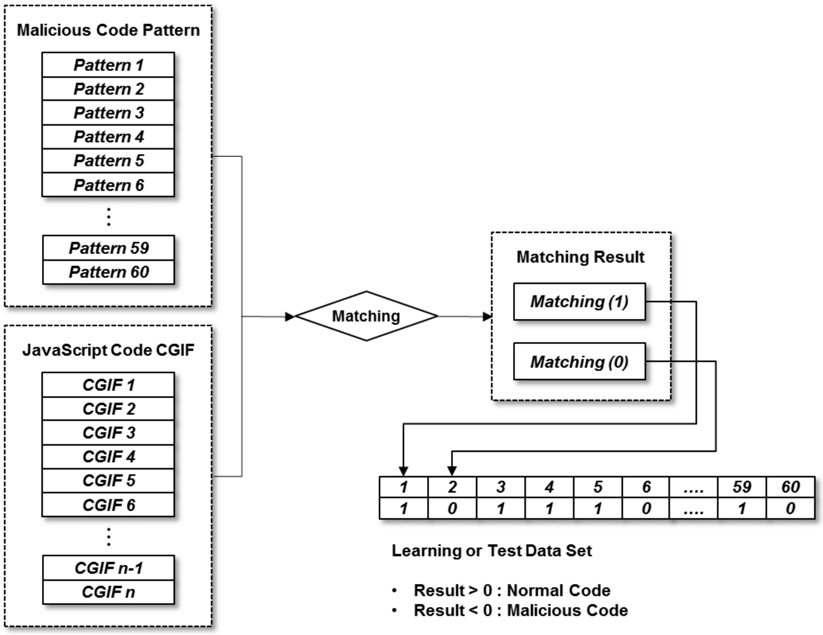

Figure 6: SVM Data Set Creation

Learning data and test data are created using the results of matching based on 60 selected malicious patterns. CGIFs of the selected malicious codes are matched with those of the normal script codes, with resulting values used as index values for SVM learning data and test data. When the CGIF pattern of the normal script code or of the script code to be detected matches that of the malicious code, a 1 is returned; otherwise, a 0 is returned. Table 6 shows examples of datasets created using the resulting values from the pattern matching process in Figure 6. When the patterns are matched, the data has a truth value of 1; otherwise, it has a false value of 0.

The new script is performed the pattern learning. If the new script is ma-

Table 6: Created Data Sets

Code Pattern	1	2	3	4	$\cdots$	60
Code Set(1)	1	0	1	1	$\cdots$	0
Code Set(2)	1	0	0	1	$\cdots$	1

licious code, it needs classification of malicious codes. Therefore, we explain the weight value measurement method. In this paper, we selected five malicious pattern based on the highest frequency. The extracted malicious patterns are integrated after malicious pattern learning. Equation 1 shows the weight value measurement method based on keyword.

$$iPf(t) = \log(\frac{|w|}{fr_w(t)}) \tag{1}$$

where w is the total number of scripts, fr_w is the frequency of t (malicious pattern) in learning scripts. In equation 1, iPf can be classified the noise and general pattern (small malicious concept and relation). Equation 2 shows weight value of malicious pattern using Term Frequency (TF) and Inverse Term Frequency (iTF).

$$KW = TF(t) \times iTF(t) \tag{2}$$

5 Experiments and evaluation

We conducted experiments in which malicious JavaScript codes were collected and a total of 210 scripts used as experimental datasets (105 learning datasets and 105 test datasets). Each dataset consisted of a total of three groups. The first group contained 35 malicious JavaScript codes; the second group contained malicious behaviors that were not malicious but were similar; and the last group consisted of 35 normal JavaScript codes.

Table 7 shows the results of experiments conducted using the SVM based on the datasets. The malicious code detection result for Group A shows a high detection rate of about 94%; Group B, 83%; and Group C, 15%. In general, Groups A and B show high detection rates. On the other hand, the Group C result shows a positive error of 15%, which can be considered a low detection rate. Group B, the group for detecting variants of malicious codes, has a high detection level, which implies that the proposed method has a high detection rate for similar and variant malicious behavior forms.

For malicious codes corresponding to Group A, all three methods exhibit similar results. However, for the codes similar to the malicious codes subject to comparison as in Group B, the results of the malicious code detection method

Table 7: A Result of Comparison between the Proposed Method and Existing Techniques

Group	Proposed Detection Method		A's Vaccine		B's Web Browser	
	Detected	Detection Rate	Detected	Detection Rate	Detected	Detection Rate
A	33	94.2%	34	97.1%	31	88.5%
B	29	82.8%	28	80%	20	57.1%
C	5	14.2%	8	22.8%	12	34.2%

using the conceptual graph proposed in this paper show a higher detection rate than those of other methods. This implies that the concept based similarity measurement method using conceptual graphs is more suitable for detecting the codes similar to malicious behaviors than the existing pattern matching techniques. In addition, the search result for Group C, which shows the lowest positive error, signifies that the proposed method is suitable for malicious code detection.

In the table 7, group A is the detection result based on the well known malware codes and result is indicated similar result between proposed method and others. The group B is the result based on the little known malware codes and proposed method is better than others. Finally, group C the detection result based on general codes (not malware codes) and proposed method is lower than others.

The existing malicious code detection methods used by A and B are signature based approaches that use a string of malicious codes. However, the malicious script analysis method proposed in this paper, which uses conceptual graphs created from the patterns of malicious behaviors, combines the signature based detection method and the behavior based detection methods. As shown in the experimental results, it exhibits a better detection rate than that of malicious code detection methods that rely solely on the signature based approach. This suggests that the proposed method is not only suitable for detection of malicious codes, but is also more efficient than other detection methods as it combines the advantages of more than two malicious code detection methods.

6 Conclusions and future work

This paper presented a method for detecting unknown malicious codes that are similar to or variants of existing malicious codes, using conceptual graphs and SVM. In general, conventional malicious code detection methods create patterns only through analysis of malicious codes, and attempt to detect malicious codes by comparing those patterns. In contrast, our proposed method creates malicious patterns not only through analysis of malicious codes but also using the concept of malicious behaviors and the relationships between those behaviors and malicious codes.

Furthermore, it uses SVM, which exhibits outstanding machine learning performance, for malicious code detection. It combines the advantages of signature based and behavior based detection methods. The results of the experiments verify that the proposed malicious script code analysis and detection method effectively detects both malicious scripts and unknown malicious codes. It is necessary to collect more diverse patterns in order to improve the performance of the system. However, as the number of patterns increases, positive errors, in which normal codes are identified as malicious codes, may occur. In order to prevent such a drawback, a method that defines the concepts and the relationships of the codes more accurately and in more detail is required.

Thus, in future studies, we plan to improve the malicious code detection accuracy by collecting more patterns and thereby increasing the number of patterns. In addition, research into the definition of malicious behaviors through the relationships among malicious codes will be conducted.

Acknowledgements

This study was supported by research fund from Chosun University, 2014.

References

[Ajay, 12] Ajay, K.: Modern Malware and APT: What You May be Missing and Why; AtlSecCon (2012).

[Baget, 03] Baget, J.: Simple conceptual graphs revisited: Hypergraphs and conjunctive types for efficient projection algorithms; Springer-Verlag, 2746, (2003), 229-242.

[Chen et al., 09] Chen, Y., Liu, F., Vanschoenwinkel, B., Manderick, B.: Splice Site Prediction using Support Vector Machines with Context-Sensitive Kernel Functions; Journal of Universal Computer Science, 15, 13(2009), 2528-2546.

[Choi et al., 11] Choi, J., Kim, H., Choi, C., Kim, P.: Efficient Malicious Code Detection Using N-Gram Analysis and SVM; 14th Network-Based Information Systems, (2011), 618-621.

[Christodorescu and Jha, 03] Christodorescu, M., Jha, S.: Static Analysis of Executables to Detect Malicious Patterns; 12th USENIX Security Symposium, 1, (2003), 12-12.

[Cova et al., 10] Cova, M., Kruegel, C., 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, (2010), 281-290.

[Dell SecureWorks, 12] Dell SecureWorks.: Advanced Presistent Threats: Higher Education Security Risks; `http://www.secureworks.com/resources/articles/featured\_articles/20120709-hcr/`

[Elshoush and Osmank, 11] Elshoush, H., Osmank, I.: Alert correlation in collaborative intelligent intrusion detection systems - A survey; Applied Soft Computing In Press, 11, (2011), 4349-4365.

[Fredrikson et al., 10] Fredrikson, M., Jha, S., Christodorescu, M., Sailer, R. Yan, X.: Synthesizing Near-Optimal Malware Specifications from Suspicious Behaviors; In Proc. of 2010 IEEE Symposium on Security and Privacy, (2010), 45-60.

[Giura and Wei, 12] Giura, P., Wei, Wang.: A Context-Based Detection Framework for Advanced Persistent Threats; Cyber Security 2012 International Conference, (2012), 69-74.

[Gregoire, 09] Gregoire, J.: JavaScript and Visual Basic Script threats: Different scripting languages for different malicious purposes; 18th International EICAR Conference, (2009).

[Hensman, 04] Hensman, S.: Construction of Conceptual Graph Representation of Texts; HLT-SRWS '04 Proceedings of the Student Research Workshop, (2004), 49-54.

[Ho et al., 13] Ho, T., Kang, H., Kim, S.: Graph-based KNN Algorithm for Spam SMS Detection; Journal of Universal Computer Science, 19, 16(2013), 2404-2419.

[Karalopoulos et al., 04] Karalopoulos, A., Kokla, M., Kavouras, M.: Geographic Knowledge Representation Using Conceptual Graphs; 7th AGILE Conference on Geographic Information Science, (2004), 511-521.

[Kruegel et al., 09] Kruegel, C., Kirda, E., Mutz, D., Robertson, W., Vigna, G.: Polymorphic worm detection using structural information of executables; In RAID '06: Proc. 8th International Symposium on Recent Advances in Intrusion Detection, LNCS, (2006), 207-226.

[Laskov and Srndic, 11] Laskov, P., Srndic, N.: Static detection of malicious javascript bearing pdf documents; In ACSAC '11: Proc. 27th Annual Computer Security Applications ConferenceAnnual Computer Security Applications Conference. IEEE Computer Society, (2011), 373-382.

[Legg et al., 13] Legg, P., Moffat, N., Nurse, J., Happa, J., Agrafiotis, I., Goldsmith, M., Creese, S.: Towards a Conceptual Model and Reasoning Structure for Insider Threat Detection; Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications(JoWUA), 4, 4(2013), 20-37.

[Likarish et al., 09] Likarish, P., Jung, E., Jo, I.: Obfuscated Malicious Javascript Detection using Classification Techniques; Malicious and Unwanted Software, (2009), 47-54.

[Liu et al., 13] Liu, F., Wang, J., Bai, H.: YaVNC - A Virtual Application Solution for Smartphone; Journal of IT Convergence Practice, 1, 4(2013), 39-49.

[Mishne and Rijke, 04] Mishne, G., Rijke, M.: Source Code Retrieval using Conceptual Similarity; Conf. Computer Assisted Information Retrieval, (2004), 539-554.

[Shahzad and Lavesson, 13] Shahzad, R., Lavesson, N.: Comparative Analysis of Voting Schemes for Ensemble-based Malware Detection; Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications(JoWUA), 4, 1(2013), 98-117.

[Shen et al., 14] Shen, Y., Chien, R., Hung, S.: Toward Efficient Dynamic Analysis and Testing for Android Malware; Journal of IT Convergence Practice, 2, 3(2014), 14-23.

[Wang and Chiang, 09] Wang, J., Chiang, J.: An Efficient Data Preprocessing Procedure for Support Vector Clustering; Journal of Universal Computer Science, 15, 4(2009), 705-721.

[Wei and Erik, 09] Wei, Y., Erik, W.: Toward Automatic Discovery of Malware Signature for Anti-virus Cloud Computing; Complex Sciences, (2009), 724-728.

[Zhang and Yu, 01] Zhang, L., Yu, Y.: Learning to Generate CGs from Domain Specific Sentences; Proc. of ICCS'01, (2001), 44-57.

[Zhong et al., 02] Zhong, J., Zhu, H., Li, J., Yu, Y.: Conceptual Graph Matching for Semantic Search; Proc. of ICCS'02, (2002), 92-106.