Hindawi
Journal of Computer Networks and Communications
Volume 2018, Article ID 6873807, 6 pages
https://doi.org/10.1155/2018/6873807

Research Article

Hybrid Obfuscation Using Signals and Encryption

Bahare Hashemzade and Ali Maroosi

Department of Electrical and Computer Engineering, University of Torbat Heydarieh, Torbat Heydarieh, Iran

Correspondence should be addressed to Ali Maroosi; ali.maroosi@torbath.ac.ir

Received 24 December 2017; Accepted 3 April 2018; Published 30 April 2018

Academic Editor: Zhiyong Xu

Copyright © 2018 Bahare Hashemzade and Ali Maroosi. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Obfuscation of software and data is one of the subcategories of software security. Hence, the outlines of the obfuscation problem and its various methods have been studied in this article. This paper proposes a hybrid of two signals and encryption obfuscation to hide the behaviour program and prevent reconstruction of the normal code by hackers. The usual signal method is strong enough for obfuscation, but its problem is the high complexity because of a lot of call and return instructions. In this study, a new dispatcher was added to the source code to reconstruct the original control flow graph from the hidden one to solve the problem of the signal method. This dispatcher code is encrypted to preclude access by the hacker. In this paper, the potency that makes the obfuscation strong has been increased and the resilience that makes the obfuscation poor has been decreased. The results of a comparison of the similarity among the ambiguous data with its original code and with available efficient methods present a performance advantage of the proposed hybrid obfuscation algorithm.

1. Introduction

With the rapid expansion of the Internet and its influence on all aspects of social, cultural, scientific, economic, and political exchanges, the most important challenge facing cyberspace is the security threats to these exchanges therein. Anything that can lead to a dangerous event has become a security threat in cyberspace. The origin of security threats falls into two categories: people (human factors) and software. Each one of these has its subcategories. In the field of threats posed by human factors, we face five factors, including Red/Black Hat hackers, dissatisfied employees, domestic competitors, foreign competitors, and foreign states, while threats based on software factor, which are applied applications, can be risk factors in two ways and endanger the security of information: vulnerable applications and malwares.

Based on the performance and behaviour of malwares, these can be divided into four groups: virus, worm, Trojan horse, and botnets. Obfuscation is an invasive technique that a malware writer considers to apparently hide his malware. This means that it is done by changing the appearance of the malware source code and maintaining the functional nature of malware. It attempts to be secured by antivirus detection and continues its destructive activities. Obfuscation as an invasive technique can also be used as a defence solution in the field of software and vital information protection against security threats. Malware obfuscation is studied in this research since the access to obfuscation information and software for research is difficult or even impossible because of its confidentiality [1].

Different obfuscation methods have been presented (Figure 1). One of the obfuscation methods is adding the dead code that alters the look of the program code. The implementation of this method is easy, but the disadvantage of this approach is that it is recognized by eliminating additional commands [2–5].

Another obfuscation method is changing the names of the registers so that these will change from one generation to the next one but can be recognized by renaming the registers [6, 7]. Replacement command, which generates the separation code, is a type of obfuscation method [1, 6]. Another form of obfuscation is the shuffle code. In this method, the initial order of commands downs and the cost of detection goes up. But the implementation has problems: it can be recognized by eliminating the nonconditional commands [6, 7]. Code integrating is also an approach for obfuscation, but its disadvantage is that it is difficult to implement. The advantages of this method are the crucial diagnosis and recovery [4, 5].

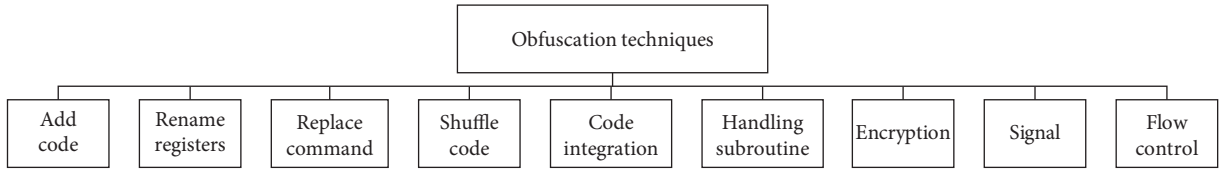


Figure 1: Different approaches for obfuscation.

Displacement and handling of the subroutine is one obfuscation method: its advantage is that it obfuscates the source of the program code and downs the order of the subroutine. The disadvantage of this method is that the code is detected by changing the subroutine [1, 2].

Another type of the obfuscation method is encryption. The advantage of this method is that the main pattern of the program code is hidden. The disadvantage of this method is that the malware could identify it using the code decoding.

The signal method is an obfuscation method in which control flow graph opcodes (operation codes) are hidden [8–10]. The advantage of this method is that it hides the control flow graph of a program and makes the information of the control flow graph of the program difficult [11–13]. The disadvantage of the signal method is the high cost of operations due to the high number of call and return instructions [10, 14–16].

This study proposed a hybrid signal and encryption (proposed S&E) method. In the usual signal method, the control flow graph of the program is hidden. However, the cost of the operation is increased in the usual signal method.

In the proposed method, first, the information of the control flow graph is hidden by the signal method, and then, the dispatcher of the signal, which reconstructs the original control flow graph from the hidden one, is added as new information to the file and encrypted to preclude access by hackers. The hybrid method does not have the disadvantage of the signal method, which has a high cost because of the organization of signals by the operation system. In other words, to preclude overloading of the operating system, it has been suggested that the dispatcher be encoded in the program. A further explanation has come up in the proposed algorithm. In addition, since an elaborate formula to calculate the complexity and resilience of obfuscation techniques is not provided in previous studies, new and transparent formulas are presented in this work.

In this article, the obfuscation outlines of the problem are taken into consideration in Section 2. In Section 3, a hybrid of the signal and encryption method is presented. The results of the implementation of the algorithm are described in Section 4, and finally, in the last section, conclusions have been presented.

2. Description of the Problem

Obfuscation is a set of methods that can be used by malware writers or software to turn one program with the same behaviour but with a different appearance to another one [1]. It consists of three objective functions and six variables, which are described in brief. Three objective functions include (a) potency, (b) resilience, and (c) cost. The potency can be considered a useful measure of the change that causes the encryption purpose of the program to be hidden. Also, the potency is considered an indicator of the obfuscation productivity measure for people. Resilience can be considered an obfuscation productivity measure for machines automatically (in opposition to potency). Cost measures the time complexity.

Among the six variables affecting the objective functions, we can point to μ_1 (program time), the number of operators and operands of the source code. The second variable μ_2 (complexity) specifies the number of conditional statements of the source code and μ_3 (complexity nesting) is the maximum depth of nested statements in the source code. μ_4 (complexity of information) is the undefined variable of the program. μ_5 (complexity fan-in/out) is the number of called functions. The last variable μ_6 (complexity data structure) is the number of defined variables in the program [6].

The potency to change the behaviour of a program that is represented as $T_{\text{pot}}(P)$ is the function that shows the lack of change in the behaviour of the obfuscated program P' to the source program and is affected by the complexity measure function $E(P)$ [6].

3. The Proposed Algorithm

In this study, the goal is the implementation of obfuscation in the signal method and encryption method in order to increase the complexity level and reduce the detection potency. At first, the signal obfuscation method is used so that the tree- and graph-like structures of the program become star structures. For example, in normal status, the applications have a graph-like structure; this graph is created on the basis of the structure of function calls, but in the case of using a signal obfuscation or socket, all requests and communications among the functions are done by sending signals. After a signal is created, the operating system organizes these signals. So, in any communication of functions, first, a signal must be sent to the operating system, and then, the operating system sends target signals into a function (or program). This structure causes the graph structure of the program to turn into a star structure. Though this structure makes the control graph of the program unclear (advantage), it increases the cost of the program (disadvantage). So, it is recommended that the dispatcher of this star structure be in its own program in order to avoid overloading the operating system. In the next step, we encode this part that is there in the program to prevent its hacking (we use obfuscation in the encryption method). The first letter of the word "Signal" and the first letter of the word

"Encrypt" were chosen to name the algorithm "S&E" because this algorithm is a combination of both. The S&E procedure algorithm is in a way such that, in the first step, functions in the source code are run line by line.

3.1. A New Approach to Calculate the Potency Function. The complexity function is defined by affecting six parameters that are not defined precisely in previous studies; therefore, in this article, due to the effect level of each variable listed on the complexity of the program, three complexity functions have been proposed in this study. In that, one of them has been chosen for implementation. According to the present six variables in the previous section and given the lack of a precise definition of the relationship between these variables in their functions, three proposals have been presented in this article. Various methods have been studied, and the following three functions which show better performance compared to the others are selected:

$$E_1(P) = \mu_2 + \mu_3 + \mu_5, \tag{1}$$

$$E_2(P) = \mu_2 + 3\mu_3 + 2\mu_5, \tag{2}$$

$$E_3(P) = 6\mu_2 + 5\mu_3 + 4\mu_5 + 3\mu_1 + 2\mu_4 + \mu_6. \tag{3}$$

The variables are divided into two groups: important and more important groups, due to the complexity effect of making decision variables. The μ_2, μ_3, and μ_5 variables are placed in the more important group, and μ_1, μ_4, and μ_6 are placed in the important variable group. We consider the variables with the same weight in (1) and the sum of these three variables to measure the complexity as $E_1(P)$. The sum of weighted three more important factors to measure the complexity is indicated as $E_2(P)$ in (2). For (3), we consider a sum of important and more important variables as $E_3(P)$. However, the effects of important factors in comparison with the most important factors can be neglected. Thus, in this study, we considered just most important factors for our measurement. Therefore, the complexity can be measured as follows:

$$T_{\text{pot}}(P) = \frac{E_1(P')}{E_1(P)} - 1, \tag{4}$$

where P' is the obfuscated program and P is the original program. If $T_{\text{pot}}(P) > 0$, obfuscation is strong and is disturbing for people, while deobfuscation is very simple for the device. The complexity of the application increases according to some used metrics. Thus, the potency can be considered a useful measure of obfuscation to people.

3.2. A New Approach for Calculation of Resilience. To measure the effectiveness of obfuscation to automatic deobfuscators, resilience is introduced. Resilience takes two parameters into account: programming attempt (the amount of time it takes to build a program that can remove the program from being obscured) (automatic deobfuscators) and attempt of the deobfuscator (run time and memory space required to remove the program from being obfuscated). The potency is in contrast to resilience because the potency focuses on making the application more complex or, in other words, increases the potency, whereas in resilience, it is paid to decrease the resilience because the later the program is identified, the better it is. Resilience of the change of a program behaviour which is displayed as $T_{\text{res}}(P)$ is a function that shows the lack of behaviour change of the obfuscated program P' to the source program P (Martinez [6]). Resilience function is defined by six parameters that are not defined precisely in previous resources. Therefore, in this article, according to the effect level of each variable in the resilience program, two complexity functions have been proposed for the implementation. $T_{\text{res}}(P)$ is affected in an automated manner by the measurement function, execution time, required memory, and amount of time it takes to build a deobfuscator. Two ways are suggested to calculate it as follows.

According to the impact of making decision variables on the measure of resilience, we chose two groups of variables such that their impact causes reduction of resilience. According to what has been said in this article, resilience can be measured as follows:

$$\begin{aligned} F_1(P) &= 2\mu_1 + \mu_2 + \mu_4 + \mu_6, \\ F_2(P) &= 3(\mu_1 + \mu_6) + 2(\mu_2 + \mu_3 + \mu_4) + \mu_5, \\ T_{\text{res}}(P) &= \frac{F_1(P')}{F_1(P)} - 1. \end{aligned} \tag{5}$$

If $T_{\text{res}}(P) < 0$, the resilience is low and obfuscation is strong. Note that when $T_{\text{res}}(P) < 0$, then $F_1(P') < F_1(P)$. It means that resilience of the obfuscated program is less than that of the original program and obfuscation is strong.

3.3. Calculating the Cost Function. The program code may require more storage space or more time to finish after changing for obfuscation. This concept is introduced as the cost of changes. The cost of changing the behaviour of an application, which is displayed as $T_{\text{cost}}(P)$, is a function that shows the lack of behaviour change of the obfuscated program P' to the source program P. $T_{\text{cost}}(P)$ is affected by measurement function of complexity of run time $O(P)$. $T_{\text{cost}}(P)$ can be compared in the following ways [6]:

(i) It is very high costly if the implementation of P' requires an exponential amount more than P.

(ii) It is high costly if the implementation of P' requires an amount of $O(n^P)$ more than P where $P > 1$.

(iii) It is of low cost if the implementation of P' requires an amount of $O(n)$ more than P.

(iv) It is of no cost if the implementation of P' requires an amount of $O(1)$ more than P.

The quality of changes is a combination of the obfuscation quality of potency, resilience, and cost, which is displayed as follows:

$$T_{\text{qual}}(P) = \left(T_{\text{pot}}(P), T_{\text{res}}(P), T_{\text{cost}}(P)\right). \tag{6}$$

4. Simulation and Results

The proposed hybrid signal and encryption method was tested on a computer with Intel core 2 Duo CPU and 1 GB RAM. The code is written in C++, and standard data are used for testing and comparing the proposed algorithm. It includes 30 viruses from the VX Heaven public dataset [17]. It consists of 10 viruses from the Second Generation Virus Generator (G2) (published in January 1993) and 20 viruses from the Next Generation Virus Construction Kit (NGVCK).

4.1. How to Evaluate the Data. We use Mishra's method [18] to compare two pieces of code. Mishra proposed a method that allows one to compare two assembly programs by assigning a score to it. It represents that the two programs are similar. The Mishra method involves the following steps:

(1) Two assembly programs X and Y are supposed, and we derive strings of opcodes except description, the empty line (distance), tags, and other orders. The result is identifier sequence lengths n and m, where n and m are the numbers of opcodes in the programs X and Y. Opcodes, respectively, have their identifiers' sequence in each phase.

(2) We compare two identifiers' sequence by considering all sequences (subsequences) of three consecutive opcodes for each step. We count the match of each case regardless of the sequences where all three opcodes are similar and marked in a coordinate graph (x, y).

(3) After comparing two sequences' opcode and marking all the matching coordinates, we gain a plotted graph on a grid of size $n * m$. The numbers of identifiers of the program X on the x-axis are shown and those of the program Y are shown on the y-axis. To reduce interference and random matching, we keep only that part of the line (string) of the length greater than a threshold value (in this study, the threshold is considered to be 5).

(4) As we are doing a continuous correspondence between the two identifiers, the same section of the section-by-line opcode to the core diameter will be formed. If a segment is in the fall core diameter, it is a match. In fact, the places in two identified fields are the same. A diagonal diameter line indicates that the match of opcodes appears in different places in two files.

(5) For each axis, we determine a fraction of opcodes that are covered by one or more segments. A similarity score for two programs is gained from these parts. The similarity metric calculates the similarity between the original program (P) and the obfuscated program (P'). This metric is 1 when P and P' are similar, and this metric is 0 when there is no similarity between P and P'. For example, the similarity score equal to 0.01 shows low similarity and good obfuscation and 0.85 shows high similarity and bad obfuscation.

Table 1: Results of the proposed hybrid signal and encryption obfuscation (S&E) for different virus groups with potency, resilience, and cost metrics.

	NGVCK			G2		
	Potency	Resilience	Cost	Potency	Resilience	Cost
0	1.34356	−0.01490	O (n)	2.01490	0.84299	O (1)
1	1.62907	−0.18108	O (1)	1.20018	0.63845	O (1)
2	1.45980	−0.07641	O (1)	1.11746	0.83201	O (n)
3	1.34376	−0.20001	O (1)	1.21230	0.64845	O (1)
4	1.92907	−0.10091	O (1)	2.41011	0.02845	O (1)
5	0.60631	−0.11593	O (1)	1.30506	0.83222	O (1)
6	2.10008	−0.20016	O (n)	2.21018	0.65845	O (1)
7	2.14311	−0.18845	O (n)	2.10018	0.84799	O (n)
8	1.12212	−0.15203	O (1)	1.34356	0.66845	O (1)
9	1.10001	−0.14015	O (1)	1.62907	0.84830	O (1)
10	2.15593	−0.10013	O (n)		—	
11	1.12311	−0.14231	O (n)		—	
12	2.02011	−0.20018	O (1)		—	
13	2.13456	−0.02011	O (1)		—	
14	1.18314	−0.13456	O (1)		—	
15	2.04231	−0.18314	O (1)		—	
16	1.20016	−0.11301	O (n)		—	
17	1.11231	−0.01008	O (1)		—	
18	1.18108	−0.04311	O (1)		—	
19	1.35213	−0.03212	O (1)		—	

Table 2: Results of the proposed hybrid signal and encryption obfuscation (S&E) for different virus groups with the Mishra criteria.

	NGVCK	G2
0	0.01490	0.02845
1	0.18108	0.63845
2	0.07641	0.83201
3	0.20001	0.64845
4	0.10091	0.02732
5	0.11593	0.83222
6	0.20016	0.65845
7	0.18845	0.84799
8	0.15203	0.66845
9	0.14015	0.84830
10	0.10013	—
11	0.14231	—
12	0.20018	—
13	0.02011	—
14	0.13456	—
15	0.18314	—
16	0.11301	—
17	0.01008	—
18	0.04311	—
19	0.03212	—

4.2. Results and Discussions. The results of the comparison of the obfuscation viruses in two groups NGVCK and G2 with their original code, based on the potency, resilience, and cost metrics, are presented in Table 1. The comparison results for original and obfuscated codes with the Mishra criteria are also shown in Table 2. The results of presented metrics in Tables 1 and 2 show consistency of the presented metric with the Mishra metric.

Table 3: Comparing the results of the proposed hybrid signal and encryption obfuscation (S&E) with the other algorithms.

		NGVCK	G2
Proposed S&E	Min	0.01490	0.02845
	Max	0.20018	0.43211
	Ave.	0.11746	0.32753
SWOD-CFW (Alam et al. [19])	Min	0.21230	0.62845
	Max	0.41011	0.84864
	Ave.	0.30506	0.74491
ACFG (Alam et al. [20])	Min	0.03452	0.40286
	Max	0.21017	0.73210
	Ave.	0.12101	0.65210
CSD estimator (Toderici and Stamp [21])	Min	0.34356	0.12349
	Max	0.62907	0.73210
	Ave.	0.45980	0.53931
HMM (Austin et al. [22])	Min	0.34376	0.44964
	Max	0.92907	0.96568
	Ave.	0.60631	0.62704
SD (Shanmugam et al. [23])	Min	0.10008	0.18108
	Max	0.14311	0.01490
	Ave.	0.12212	0.10011
Graph (Runwal et al. [24])	Min	0.10001	0.10013
	Max	0.15593	0.40151
	Ave.	0.12311	0.25655
Histogram (Rad et al. [25])	Min	0.02011	0.01491
	Max	0.18314	0.15341
	Ave.	0.13456	0.11230
Sequences (Santos et al. [26])	Min	0.04231	0.03456
	Max	0.20016	0.18314
	Ave.	0.11231	0.11301
Patterns (Shabtai et al. [27])	Min	0.18108	0.01301
	Max	0.35213	0.21630
	Ave.	0.27234	0.15654

To compare the obfuscation algorithm level proposed in this paper (S&E) with some of the available efficient methods, the minimum, maximum, and average of the similarity level of obfuscated viruses with the initial code for the proposed S&E and other algorithms are shown in Table 3. Obviously, the minimization of the similarity of obfuscated viruses by the proposed S&E is less than that in other algorithms. However, the proposed S&E has not been successful in reducing the average rate of similarity for viruses of NGVCK and G2 in some cases.

The comparison between the proposed algorithm (S&E) and other algorithms includes sliding window of difference and control flow weight (SWOD-CFW) [19], annotated control flow graph (ACFG) [20], chi-squared distance (CSD) estimator [21], hidden Markov model (HMM) [22], substitution distance (SD) [23], opcode graph similarity [24], opcode histogram [25], opcode sequences [26], and opcode patterns [27]. In general, Table 2 shows the excellence of the proposed S&E, in comparison with the other obfuscated methods.

5. Conclusion

In this study, a hybrid signal and encryption obfuscation method was presented. The proposed algorithm used a signal method to change the tree- and graph-like structure of the program into the star structure and hide the control flow graph of the problem. The problem of the signal method is high number of call and return instructions. This study suggested adding a dispatcher to the program that converts the signal program to the original control flow graph. In this way, the problem of the signal method was solved. This dispatcher was encrypted to keep it secure from hackers. Furthermore, a new approach has been suggested to measure complexity and resilience. Five functions were offered in order to calculate the values of complexity and resilience. The results of the comparison of obfuscated data similarities with the initial codes, based on Mishra's method, represent a performance advantage of the proposed and hybrid algorithm obfuscation.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

References

[1] I. You and K. Yim, "Malware obfuscation techniques: a brief survey," in *Proceedings of the 2010 International Conference on Broadband, Wireless Computing, Communication and Applications*, pp. 297–300, Fukuoka, Japan, November 2010.

[2] C. Barrýa, D. Cordero, C. Cubillos, and M. Palma, "Proposed classification of malware, based on obfuscation," in *Proceedings of the 6th IEEE International Conference on Computers Communications and Control (ICCCC)*, pp. 37–44, Băile Felix-Oradea, Romania, May 2016.

[3] C. K. Behera and D. L. Bhaskari, "Different obfuscation techniques for code protection," *Procedia Computer Science*, vol. 70, pp. 757–763, 2015.

[4] G. Canfora, A. N. Iannaccone, and C. A. Visaggio, "Static analysis for the detection of metamorphic computer viruses using repeated-instructions counting heuristics," *Journal of Computer Virology and Hacking Techniques*, vol. 10, no. 1, pp. 11–27, 2014.

[5] Y. Gao, Z. Lu, and Y. Luo, "Survey on malware anti-analysis," in *Proceedings of the Fifth IEEE International Conference on Intelligent Control and Information Processing (ICICIP)*, pp. 270–275, Dalian, China, August 2014.

[6] S. Martinez, *Source Code Obfuscation by Mean of Evolutionary Algorithms*, MSc thesis, University of Luxembourg, Luxembourg, EuropeUniversity of Luxembourg, 2011.

[7] A. J. Smith, R. F. Mills, A. R. Bryant, G. L. Peterson, and M. R. Grimaila, "REDIR: automated static detection of obfuscated anti-debugging techniques," in *Proceedings of the IEEE International Conference on Collaboration Technologies and Systems (CTS)*, pp. 173–180, Minneapolis, MN, USA, May 2014.

[8] V. Balachandran, N. W. Keong, and S. Emmanuel, "Function level control flow obfuscation for software security," in *Proceedings of the Eighth IEEE International Conference on Complex, Intelligent and Software Intensive Systems (CISIS)*, pp. 133–140, Birmingham, UK, July 2014.

[9] V. Balachandran, D. J. Tan, and V. L. Thing, "Control flow obfuscation for android applications," *Computers and Security*, vol. 61, pp. 72–93, 2016.

[10] C. K. Behera and D. L. Bhaskari, "Code obfuscation by using floating points and conditional statements," in *Proceedings of the 4th International Conference on Frontiers in Intelligent Computing: Theory and Applications (FICTA)*, pp. 569–578, Bhubaneswar, India, September 2016.

[11] M. S. Islam, M. R. Islam, A. S. Kayes, C. Liu, and I. Altas, "A survey on mining program-graph features for malware analysis," in *Proceedings of the International Conference on Security and Privacy in Communication Systems*, vol. 24, pp. 220–236, Beijing, China, September 2014.

[12] H. Ma, X. Ma, W. Liu, Z. Huang, D. Gao, and C. Jia, "Control flow obfuscation using neural network to fight concolic testing," in *Proceedings of the International Conference on Security and Privacy in Communication Systems*, vol. 24, pp. 287–304, Beijing, China, September 2014.

[13] A. Pawlowski, M. Contag, and T. Holz, "Probfuscation: an obfuscation approach using probabilistic control flows," *Detection of Intrusions and Malware, and Vulnerability Assessment*, vol. 7, pp. 165–185, 2016.

[14] V. Balachandran, S. Emmanuel, and N. W. Keong, "Obfuscation by code fragmentation to evade reverse engineering," in *Proceedings of the 2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, vol. 5, pp. 463–469, San Diego, CA, USA, October 2014.

[15] S. S. Das, *Code Obfuscation Using Code Splitting with Self-Modifying Code*, Ph.D. disseration, National Institute of Technology Rourkela, Odisha, India, 2014.

[16] J. Schneider and T. Locher, "Obfuscation using encryption," pp. 1–11, 2016, http://arxiv.org/abs/1612.03345.

[17] W. Wong and M. Stamp, "Hunting for metamorphic engines," *Journal in Computer Virology*, vol. 2, no. 3, pp. 211–229, 2006.

[18] P. OKane, S. Sezer, and K. McLaughlin, "Obfuscation: the hidden malware," *IEEE Security and Privacy Magazine*, vol. 9, no. 5, pp. 41–47, 2011.

[19] S. Alam, I. Sogukpinar, I. Traore, and R. N. Horspool, "Sliding window and control flow weight for metamorphic malware detection," *Journal of Computer Virology and Hacking Techniques*, vol. 11, no. 2, pp. 75–88, 2015.

[20] S. Alam, R. N. Horspool, I. Traore, and I. Sogukpinar, "A framework for metamorphic malware analysis and real-time detection," *Computers and Security*, vol. 48, pp. 212–233, 2015.

[21] A. H. Toderici and M. Stamp, "Chi-squared distance and metamorphic virus detection," *Journal of Computer Virology and Hacking Techniques*, vol. 9, pp. 1–14, 2013.

[22] T. H. Austin, E. Filiol, S. Josse, and M. Stamp, "Exploring hidden Markov models for virus analysis: a semantic approach," in *Proceedings of the 46th Hawaii International Conference on System Sciences (HICSS)*, pp. 5039–5048, Maui, HI, USA, January 2013.

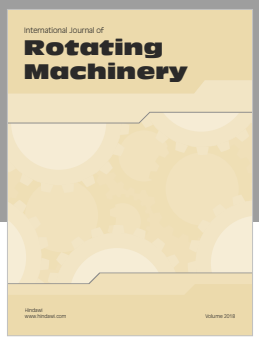
[23] G. Shanmugam, R. M. Low, and M. Stamp, "Simple substitution distance and metamorphic detection," *Journal of Computer Virology and Hacking Techniques*, vol. 9, no. 3, pp. 159–170, 2013.

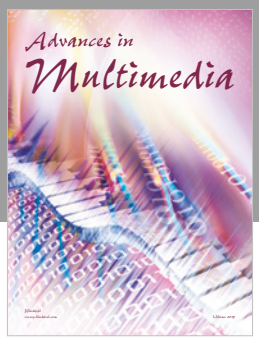
[24] N. Runwal, R. M. Low, and M. Stamp, "Opcode graph similarity and metamorphic detection," *Journal in Computer Virology*, vol. 8, no. 1-2, pp. 37–52, 2012.

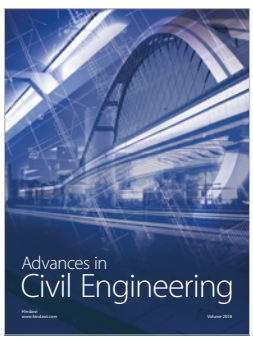
[25] B. B. Rad, M. Masrom, and S. Ibrahim, "Opcodes histogram for classifying metamorphic portable executables malware," in *Proceedings of the International Conference on E-Learning and E-Technologies in Education*, pp. 209–213, Lodz, Poland, September 2012.

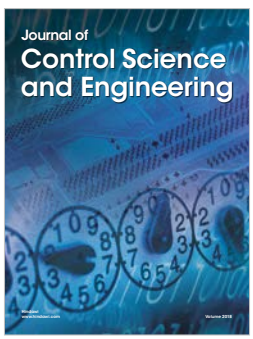
[26] I. Santos, F. Brezo, X. Ugarte-Pedrero, and P. G. Bringas, "Opcode sequences as representation of executables for data-mining-based unknown malware detection," *Information Sciences*, vol. 231, pp. 64–82, 2013.

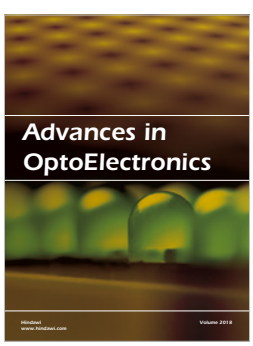
[27] A. Shabtai, R. Moskovitch, C. Feher, S. Dolev, and Y. Elovici, "Detecting unknown malicious code by applying classification techniques on opcode patterns," *Security Informatics*, vol. 1, no. 1, pp. 1–22, 2012.

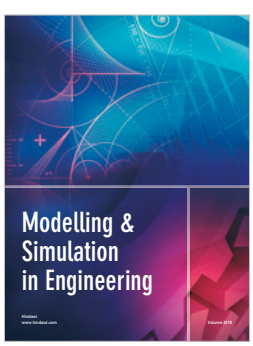


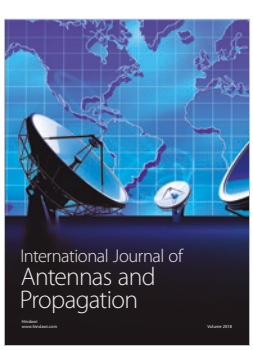


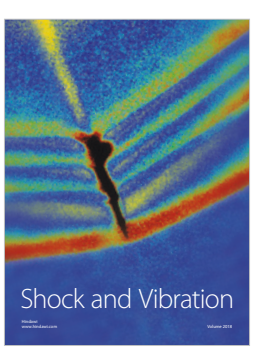


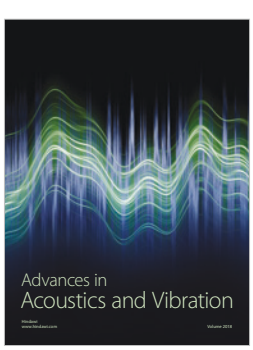












Hindawi

Submit your manuscripts at
www.hindawi.com