Journal of Computing and Security

July 2019, Volume 6, Number 2 (pp. 53–63)

http://www.jcomsec.org

CSE: A Novel Dynamic Obfuscation Based on Control Flow, Signals and Encryption

Bahare Hashemzade [a] Majid Abdolrazzagh-Nezhad [b,*]

[a] *Department of Electrical and Computer Engineering, University of Torbat Heydarieh, Iran.*

[b] *Department of Computer Engineering, Faculty of Engineering, Bozorgmehr University of Qaenat, Qaen, Iran.*

ARTICLE INFO.

Article history:

Received: 1 February 2019

Revised: 26 January 2020

Accepted: 26 January 2020

Published Online: 8 April 2020

Keywords:

Dynamic Obfuscation, Control Flow, Signals, Encryption, Management Table, Spritz, Evaluation Functions.

ABSTRACT

Obfuscation, as one invasive strategy, is considered to be a defense strategy in the field of software and vital information protection against security threats. This paper proposes a new dynamic obfuscation method, called CSE, based on combining a triplet of control flow, signals and encryption of the management table (MT). This triplet exchanges and hides the control graph program. Then, it produces the MT that includes addresses to guide communication between instructions. A type of the stream cipher symmetric encryption (Spritz) applies to encrypt the MT. Also, a multi-objective function (the ability and the resiliency) based on six implementation metrics and two classic objective functions (the cost and the Mishra) are considered to evaluate the proposed obfuscation method. Therefore, the proposed triplet obfuscation method and the multi-objective functions are performed on a small program and a benchmark dataset. The results of our evaluations show that CSE has competitive advantages in comparison with other methods.

© 2019 JComSec. All rights reserved.

1 Introduction

Today internet and cyberspace are increasingly developing and effected on cultural, scientific, economic and political changes. A major cyberspace challenge is the security threats of transportation information. A security threat in cyberspace is a potential risk that can lead to a dangerous event. The origin of security threats falls into two categories: people (human factors) and software. Obfuscation is an invasive strategy that a malware writer considers hiding the software control flow. This means that it is done by changing the appearance of the malware source code maintaining its functional nature and containing its sensitive information [1]. In addition, the invasive strategy can be utilized as a defense solution in the field of software and vital information protection against security threats. Although reviewing the obfuscation literature shows that a universal obfuscator for any type of software does not exist and perfectly secure software obfuscation is impossible, software obfuscation is still used in commercial systems to "raise the bar" for attackers [2].

Since the access to software and obfuscation information for research is impossible, malware obfuscation has been studied in this research. There are many obfuscation algorithms that they mostly utilized nine basic obfuscation methods in their procedure. The methods can be classified into two static and dynamic classes [3–5] (Figure 1).

In the static obfuscation, the internal structures of

* Corresponding author.

Email addresses: b.hashemzadeh@profs.torbath.ac.ir (B. Hashemzadeh), abdolrazzagh@buqaen.ac.ir (M. Abdolrazzagh)

https://dx.doi.org/10.22108/jcs.2020.115402.1017

ISSN: 2322-4460 © 2019 JComSec. All rights reserved.

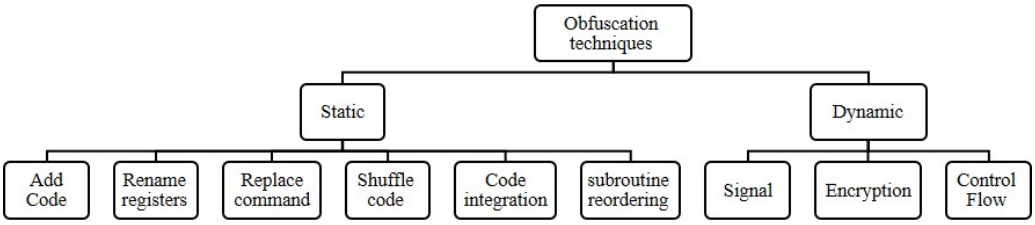

Figure 1. Different Methods to Obfuscation.

the program and control flow are changed to achieve the static program analysis confusion. One of the static methods is adding the dead code [6, 7] that alters the look of the program code (Figure 2). The implementation of this method is easy, but the disadvantage of this approach is that it is recognized by eliminating additional commands.

Another static method is changing the names of the registers (Figure 3), so that these will change from one generation to the next one, but can be recognized by renaming the registers [8–10].

Replacement commands and arithmetic and Boolean expressions (Figure 4) which generate the separation code, is a type of static obfuscation. The replacement increases the complexity of the obfuscated code and = can be applied recursively to increase strength. Another form of static obfuscation is static shuffle code. In this method, the initial order of commands downs and the cost of detection goes up [[11], Figure 1]. But the implementation has a serious challenge: it can be recognized by eliminating the non-conditional commands [8, 12]. Code integrating is also an approach for static obfuscation that the obfuscated program braids itself to the code of its target program. To achieve this aim, firstly manageable objects are produced from its target program, then seamlessly adds itself between them, and reassemble the integrated code into a new generation [13]. Difficult implementation is one of its disadvantages and the crucial diagnosis and recovery are its advantages. The last static method is the subroutine reordering technique [14, 15] that changes the order of the code's subroutines in a random manner (Figure 5). This technique can generate n! different variants, where n is the number of subroutines. Detecting the obfuscated code by changing the subroutine is the disadvantage of this method.

The dynamic obfuscation methods are designed based on dynamically changing the program code or the execution path at runtime. One of the dynamic obfuscation methods is encryption [16, 17]. Encrypting important codes, modifying function names and utilizing discrete encryption keys to encrypt the program code are different encryption techniques used for dynamic obfuscation (Figure 6). The advantages of this method were that the main pattern of the program code is hidden. The disadvantage of this method is that the malware could identify it using code decoding. The obfuscation utilizing signals is considered to be dynamic obfuscation that it hides the control flow graph opcodes [[18], Figures 1, 2, and 3]. The Advantage of this method is that it makes the information of the control graph of the program difficult. The high cost of operations due to the high number of calls is its disadvantage. The control flow obfuscation, which obfuscates the control graph of the program, is effective resistance the static analysis of the program (Figure 7) [19].

The mentioned basic methods are applied to obfuscate based on developing or combining them. The S&E [1] [4] is a hybridization of signal and encryption methods that consisted of a dispatcher. The S&E can be trapped in an infinite cycle if the designed signal method is not working properly. The REDIR [2] [8] combines "changing the names of the registers by an intermediate representation of binary program" with "subroutine reordering by a rule-based expert system". Identifier renaming, string encryption, and control flow obfuscation are used as a triple combination for obfuscation algorithm in AndrODet [9]. The SWOD-CFW [3] approach [20] is a hybridization of the subroutine reordering and the control flow method. SWOD is a window that represents differences in the opcode distribution graph and the CFW captured the control flow. Since the SWOD's size could be changed based on sliding through an opcode distribution graph, finding an optimal size for the SWOD is a challenge for the SWOD-CFW approach. Also, disassembling and extracting the opcodes from a binary file is an expensive step of this approach. The SD [4] [21], the CSDE [5] [22], the OGS [6] [23], the OH [7] [24] and the HMM

[1] Signals and Encryption
[2] Rule Engine Detection by Intermediate Representation
[3] Sliding Window of Difference and Control Flow Weigh
[4] Substitution Distance
[5] Chi-Squared Distance Estimator
[6] Opcode Graph Similarity
[7] Opcode Histogram

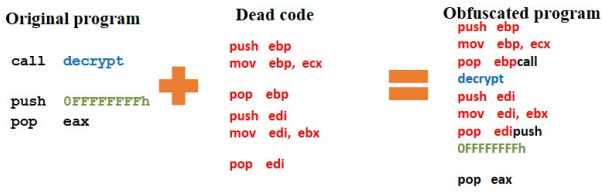

Figure 2. A Sample of the Static Obfuscation Based on Adding Dead Code.

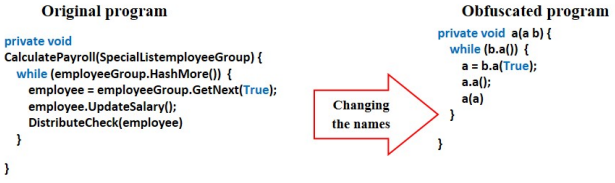

Figure 3. A Sample of the Static Obfuscation Based on Changing the Names of the Registers.

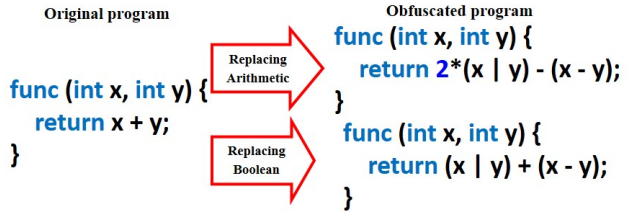

Figure 4. A Sample of the Static Obfuscation Based on Replacing Commands.

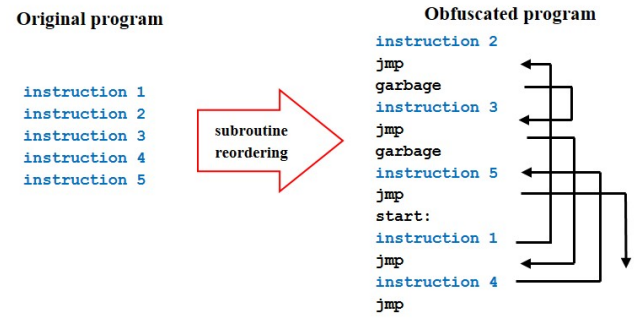

Figure 5. A Sample of the Static Obfuscation Based on the Subroutine Reordering.

[8] [25] are different developed drafts of subroutine reordering method that are based on substitution cipher of opcode graph and hidden Markov models. The frequency histogram of instruction opcodes is considered to assembly subroutine reordering of the source codes in the OH [24].

Concerning the advantages of the dynamic obfuscation techniques, a novel triplet obfuscation method is proposed by combining the signals, the flow control and the encryption techniques which is called the CSE. In our proposed method, first, we combine the control

[8] HMM Hidden Markov Models

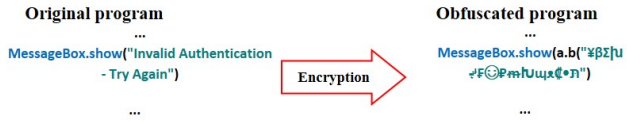

Figure 6. A Sample of the Dynamic Obfuscation Based on the Encryption.

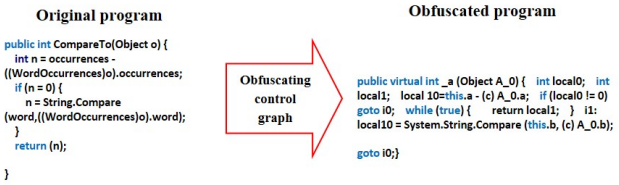

Figure 7. A Sample of the Dynamic Obfuscation Based on Obfuscating Control Flow.

flow and the signals to exchange and hide the control graph program, then a management table (MT), which consists of addresses, is designed to guide communications between functions/instructions. This hybridization avoids trapping in an infinite cycle by the MT. Encrypting the MT by the Spritz as a stream cipher encryption is the final part of the CSE to protect this critical table from hackers. So, the CSE delays the normalization process. The designed objective functions of [4] are utilized to evaluate the ability and the resiliency of the triplet obfuscation method.

In the rest of the article, the obfuscation as the research problem is described in Section 2. Section 3 explains the proposed obfuscation method and Section 4 presents the experimental results and compares them. Finally, conclusions and achievements are discussed in Section 5. Also, an original small program, which is utilized in subsection 4.1, is shown in Appendix A.

2 Description of the Problem

In the current research, obfuscation is considered to be the research problem. The problem is a procedure, which tries to change the appearance of the original program and saving its original behavior from attacks. The evaluation of methodologies, which solve the problem, is done based on four criteria (objective functions) such as ability, flexibility cost, and Mishra. So, this section is divided into two subsections; The obfuscation program and the evaluation criteria (functions).

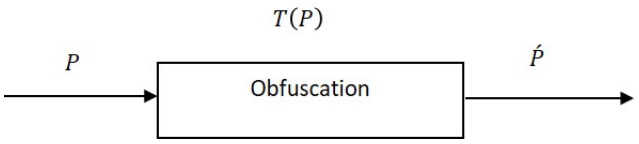

Figure 8. The Overview of Obfuscation.

The first subsection consists of the standard definition of obfuscation and the second one includes classic objective functions and novel proposed evaluation functions that are a part of this research contribution.

2.1 The Obfuscation Program

An obfuscation program is a procedure that considers both the invasive strategy taken by hackers and the defense strategy for protecting important software. An obfuscation transforms a program P into a program P' so that P and P' have the same observable behavior (Figure 8). The original program P and the obfuscated program P' should not differ in their functionality to the user, however, non-visible side effects, like the creation of temporary files are allowed in this loose definition [26].

2.2 The Evaluation Functions

The evaluation functions in [4, 27] are based on six implementation metrics. These metrics are listed as follows:

- Program length (μ_1): is the number of operators and operands of a program. This complexity also called Halstead's complexity is linked to the program's code length.
- Cyclomatic complexity (μ_2): is the number of loops and tests in the program.
- Nesting complexity (μ_3): is the maximum depth of predicate nesting in the module.
- Data flow complexity (μ_4): is the number of variables that are defined outside of the blocks where they are used in (*e.g.* global variables, the same variables used in different blocks).
- Fan-in/out complexity (μ_5): is $fan_{in} \times fan_{out}$, where fan_{in} of a module, M is calculated as the number of modules calling M plus the number of reading effects of M and the fan_{out} of a module M is the number of modules called by M plus the number of write effects of M.
- Data structure complexity (μ_6): is the number of defined variables in the program.

The implementation metrics can be divided into two categories, significant and more significant metrics, based on their definitions and previous performances [27]. The cyclomatic complexity (μ_2), the nesting complexity (μ_3) and the fan-in/out complexity (μ_5) are fallen into the more significant category and the program length (μ_1), the data flow complexity (μ_4) and the data structure complexity (μ_6) are considered in the significant category.

In the current research, instead of using the sum of the above metrics to design classic objective function, three objective functions are considered to evaluate obfuscated programs. These objective functions are ability, resiliency and cost functions that are presented as follows:

2.3 Ability Function

This function (as known as the program's ability) depends on its transformation potency and complexity. So, the ability objective function [27] is formulated as follows:

$$T_{pot} = \frac{E(p')}{E(p)} - 1 \tag{1}$$

Where p and p' are the original and the obfuscated programs, respectively; and E(p) is the complexity measurement function. In the current research, the sum of the more significant metrics is considered as follows:

$$E(p) = \mu_2 + \mu_3 + \mu_5 \tag{2}$$

If T_{pot} (p)>0, obfuscation is strong.

2.4 Resiliency Function

The resiliency function ($T_{res}(P)$) is introduced to measure the effectiveness of obfuscation. It is in contrast to ability because the ability focuses on increasing the complexity of the program, but resiliency tries to decrease the speed of the program identification from its obfuscated program. The resiliency takes two parameters into account: programming attempt (the amount of time it takes to obfuscate a program) and attempt of de-obfuscator (run-time and memory space required to remove the program from being obfuscated). The ($T_{res}(P)$) is one-way if the information is removed from P such that P cannot be reconstructed from P' [27]. Otherwise, a new formula is proposed to calculate it as follows:

The resiliency function ($T_{res}(P)$) was introduced to measure the effectiveness of obfuscation. It is the contrast of the ability because the ability focuses on increasing the complexity of the program, but the resiliency tries to decrease the speed of the program identification from its obfuscated. The resiliency takes two parameters into account: programming attempt (the time of the obfuscated program) and attempt of de-obfuscator (the run-time and memory space required to remove the program from obfuscated). The ($T_{res}(P)$) is one-way if the information is removed from P such that P cannot be reconstructed from P'

[27], else a new formula is proposed to calculate it as follows:

$$T_{res} = \frac{F(p')}{F(p)} - 1 \quad (3)$$

Where p and p' are the original and the obfuscated programs, respectively; and F(p) measures the run-time (the program length), the required memory as well as the amount of time spent to create an automatic de-obfuscator. In the current research, the metrics, which have more influence on de-obfuscating [[27], Figure 2], are considered calculating the F(p). So:

$$F(p) = 2\mu_1 + \mu_2 + \mu_4 + \mu_6 \quad (4)$$

If T_{res} (p)<0. the resiliency is low and obfuscation is strong.

2.5 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 the behavior change of a program P, which is displayed as T_{cost} (P), is a function that shows the lack of behavior change of the obfuscated program P' to the P. The T_{cost} (P) is affected by the measurement function of the complexity of run time O(p). T_{cost} can be compared in the following ways [27]:

- It is very costly if the implementation of P' requires an exponential amount more than P.
- It is very costly if the implementation of the P' requires the amount of O(np) more than P where p>.
- It is cheap if the implementation of P' requires the amount of O(n) more than P.
- It is free if the implementation of P' requires the amount of O(1) more than P.

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

$$T_{qual}(p) = (T_{pot}(p), T_{res}(p), T_{cost}(p)) \quad (5)$$

2.6 Mishra Function

The Mishra criterion [28] is utilized to compare two pieces of assembly program code by assigning a score to it. It represents how much the two programs are similar to each other. So, the highest, lowest and average similarity of the obfuscated code with its original ones is calculated by the criteria in the experimental results section. The Mishra criterion involves the following steps:

- Step1: Suppose two assembly programs X and Y, and their 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, respectively. The opcodes have their identifiers' sequence in each phase.
- Step2: Two identifiers' sequence are compared by considering all sequences (sub-sequences) of three consecutive opcodes for each step. The match of each case is counted regardless of the sequences where all three opcodes are similar and marked in a coordinate graph (x, y).
- Step3: After comparing two sequences' opcode and marking all the matching coordinates, a plotted graph is gained on a grid with size n × m. The x-axis and the y-axis refer to the numbers of identifiers of the program X and the program Y, respectively. Only that part of the line (string) of the length greater than a threshold value (in this research, the threshold is considered to be five) is kept in order to reduce interference and random matching.
- Step4: A continuous alignment is done between the two identifiers; the same section of the section-by-line opcode will be formed to core diameter. If a segment is in fall core diameter, it is a match. In fact, the places in the two identified fields are the same. A diagonal diameter line indicates that the match of opcodes appears in different places in two codes.
- Step5: A fraction of opcodes is determined for each axis that being covered by one or more segments. A similarity score for the two programs is gained from these parts.

3 The Proposed Method

In this section, a new dynamic obfuscation method, which is called the CSE, is proposed based on combining the signals, the control flow and the encryption techniques to increase complexity and decrease resiliency. To achieve this aim, first, hybridization of the control flow and the signals are performed to change the control flow graph and hide it. The signals replace call, jump and return commands during the hybridization, and they define the same structures as the main program functions by handling routines, simultaneously. Then a management table (MT), which consists of addresses, is designed to guide communication between functions/instructions. Finally, encrypting the MT by the Spritz [29] is the final part of the CSE to protect this critical table from hackers and its de-encryption keys embed into the MT to avoid increasing the run-time of obfuscation. The Spritz is a replacement encryption algorithm, which is an improved RC4-like stream cipher encryption from the symmetric encryption methods. Its Pseudo-code is presented in [[29], Figure 9]. The pseudo-code of the

Listing 1: The Pseudo Code of the CSE.

```
1. Start
2. Input: program code P with C++
3. if P's structure is a graph// means P is not obfuscated.
4.  run the control flow technique.
5.  run the signals technique.
6.  produce the MT based on the communications addresses.
7.  encrypt the MT by the Spritz.
8.  embed the encryption keys in the MT.
9.  get the obfuscated program P' with C++
10. evaluate the P' by Tpot, Tres and Tcost
11. End
}
```

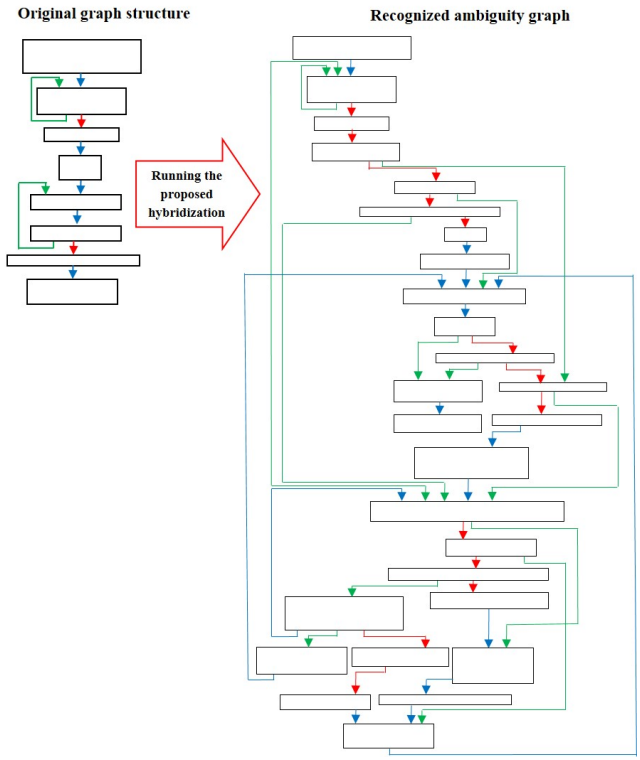

Figure 9. A Sample of Running the Proposed Hybridization-Based on the Control Flow and the Signals.

CSE is shown in Listing 1 and a small program, as a sample, is presented and obfuscated in Appendix A.

Note that each program has a graph structure which is created based on the structure of call functions. Applying the signals obfuscation causes all requests and communications between functions/instructions to be carried out using send signals. So, one signal should send to the operating system (OS) for each communication, and then the OS sends the mentioned signal to the target function. The procedure allows the graph structure of the original program to become a star structure. Therefore, when converting the program into an assembler file after executing and loading on the computer memory, the program becomes ambiguity. In Figure 9, the original graph structure, which is close to the presented sample in Appendix A, and its recognized graph after running the hybridization method of the control flow and the signals are shown.

4 The Experimental Results and Comparison

The proposed CSE obfuscation method was performed on a processor with Intel Core i5 7200-2.71 GHz, 8GB RAM, and 64 bits operating system. CSE's code is written in C++ and evaluated based on the implementation metrics (The cyclomatic complexity (μ_2), the nesting complexity (μ_3) and the fan-in/out complexity(μ_5) are fallen into the more significant category and the program length (μ_1), the data flow complexity (μ_4) and the data structure complexity (μ_6) are considered in the significant category) and four objective functions (ability $T_{pot}(p)$, resiliency $T_{res}(p)$, cost T_{cost} (p) and Mishra). To compare CSE's behavior with the state-of-the-art obfuscation methods, ten obfuscation methods are considered such as the S&E [4], the SWOD-CFW [20], the ACFG [9] [30], the CSDE [22], the HMM [25], the SD [21], the OGS [23], the OH [24], the OSe [10] [31] and the OP [11] [32]. Due to the unavailability of the source codes of the methods, the considered objective functions, such as Tpot(p), Tres(p) and Tcost(p), cannot be compared in these methods. Therefore, this section is divided into two subsections to evaluate the considered objective functions and the CSE.

In subsection 4.1, a small program is considered obfuscating by the SCE and BUZATU [33], which its source code is available. The behaviors of the methods are evaluated by T_{pot}(p), T_{res}(p) and T_{cost}(p). The SCE compares with the state-of-the-art obfuscation methods on a benchmark dataset included 30 viruses from VX Heaven public dataset [34], which was applied in previous studies [[20]-[25], [30]-[32]]. Then, the SCE evaluated by Mishra's criterion [28] in subsection 4.2.

4.1 Objective Functions Evaluation

A small program, as a sample, is presented and obfuscated by the CSE in Appendix A. Its original program consists of three functions including "Matrix mul matrix", "Parse matrices" and "Main". These functions are evaluated based on the implementation metrics and the results are shown in Table 1. Also, its obfuscated program includes 11 functions; "Matrix mul matrix", "Parse matrices", "Main", "GJWMBANBWH", "IMREJOKYGN", "CWFBNBRSZ", "AUKAJYWUDU", "VUUPXQDQCA", "YLHNWSGIPR", "MWRLMCGIRG" and "CXTHQSKROZ". These functions are also evaluated based on the im-

[9] Annotated Control Flow Graph
[10] Opcode Sequences
[11] Opcode Patterns

Table 1. The Metrics Evaluations for the Original Codes.

Module	μ_1	μ_2	μ_3	μ_4	μ_5	μ_6
Matrix mul matrix	41	4	3	50	5	12
Parse matrix	83	5	4	30	2	21
Main	89	2	1	71	71	24
Total	213	11	8	151	65	57

Table 2. The Metrics Evaluations for the Obfuscated Codes Based on the CSE.

Module	μ_1	μ_2	μ_3	μ_4	μ_5	μ_6
Matrix mul matrix	12	16	3	15	2	12
Parse matrix	10	4	4	10	1	5
Main	8	10	3	15	117	12
GJWMBANBWH	12	2	2	8	5	4
IMREJOKYGN	8	4	4	9	1	8
CWFBNBRSZ	8	4	4	8	8	8
AUKAJYWUDU	8	4	4	8	4	8
VUUPXQDQCA	8	4	4	8	8	8
YLHNWSGIPR	8	4	4	8	4	8
MWRLMCGIRG	8	4	4	8	4	8
CXTHQSKROZ	3	2	1	5	7	2
TOTAL	93	58	37	93	74	83

Table 3. The Comparison of the Objective Functions for the CSE and the BUZATU Methods.

Objective functions	T_{pot}(p)	T_{res}(p)	T_{cost}(p)
The CSE	1.01	-0.34	O(n)
BUZATU	0.56	4.3	O(n)

plementation metrics and the results are shown in Table 2. In addition, the original sample program is obfuscated by the BUZATU obfuscation method [33] and the metrics are calculated for it. Table 3 compares the values of the objective functions for the obfuscated program based on the CSE and BUZATU. Due to $T_{pot}(CSE)>T_{pot}$ (BUZATU), the ability of the CSE is higher than the BUZATU. Since T_{res} (CSE)$<T_{res}$ (BUZATU), the resiliency of the CSE is lower than the BUZATU. Therefore, the obfuscation with the CSE is stronger than the BUZATU.

4.2 The CSE Evaluation

A benchmark dataset included 30 viruses from VX Heaven public dataset [34], which was applied in the previous studies [[20]-[25], [30]-[32]], and Mishra criteria [28], are considered for the comparison of obfuscation algorithms. These datasets consist 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). In Table 4, the Mishra criteria are calculated to compare "the obfuscated viruses with the CSE" with "their original ones".

In Table 5, the minimum, the maximum and the average of the Mishra criteria (the similarity level) are presented for the considered obfuscation algorithms and the CSE. The CSE and the SD can achieve the best results for the NGVCK and the G2 based on Mishra criteria, respectively.

5 Conclusion

Although static and dynamic obfuscations of software threats are invasive strategies to obscure the original software from detection, the invasive strategy can be utilized as a defense strategy in the field of

Table 4. The Mishra Criteria of "the Obfuscated Viruses With the CSE" and "Their Original Ones".

	NGVCK	G2
1	0.04345	0.83955
2	0.17108	0.61004
3	0.05915	0.79361
4	0.21081	0.61993
5	0.09149	0.02845
6	0.11593	0.83222
7	0.18637	0.65845
8	0.20005	0.80075
9	0.01003	0.62721
10	0.14015	0.73286
11	0.10013	-
12	0.14231	-
13	0.20018	-
14	0.02011	-
15	0.13456	-
16	0.17006	-
17	0.10009	-

Table 5. Comparison of the CSE With the State of the Art Obfuscation Algorithms.

Obfuscation method		NGVCK	G2	Obfuscation method		NGVCK	G2
CSE	Min	0.01003	0.02845	SD	Min	0.10008	0.01490
	Max	0.21081	0.83955		Max	0.14311	0.18108
	Ave.	0.10906	0.65431		Ave.	0.12212	0.18108
SWOD-CFW	Min	0.21230	0.62845	OGS	Min	0.10001	0.10013
	Max	0.41011	0.84864		Max	0.15593	0.40151
	Ave.	0.30506	0.74491		Ave.	0.12311	0.25655
ACFG	Min	0.03452	0.40286	OH	Min	0.02011	0.01491
	Max	0.21017	0.73210		Max	0.13456	0.15341
	Ave.	0.12101	0.65210		Ave.	0.18314	0.11230
CSDE	Min	0.34356	0.12349	OSe	Min	0.04231	0.03456
	Max	0.62907	0.73210		Max	0.20016	0.18314
	Ave.	0.45980	0.53931		Ave.	0.11231	0.11301
HMM	Min	0.34376	0.44964	OP	Min	0.18108	0.01301
	Max	0.92907	0.96568		Max	0.35213	0.21630
	Ave.	0.60631	0.62704		Ave.	0.27234	0.15654
S&E	Min	0.01490	0.02845				
	Max	0.20018	0.84830				
	Ave.	0.11746	0.68458				

software and vital information protection against security threats. Reviewing the previous studies show that dynamic obfuscation is effective in resisting the static program analysis. So, a new dynamic obfuscation method, which is called the CSE, was proposed in this paper. The proposed CSE consisted of a triple combination of the control flow and the signals to change and hide the control flow graph and encryption of the management table (MT) by the Spritz as a stream cipher encryption. Six implementation metrics were described and two objective functions, including the ability T_{pot} (P) and the resiliency T_{res} (P), were considered evaluating the proposed obfuscation method. The cost and the Mishra functions were the other considered criteria for the evaluation and the comparison in the paper.

A small program, which shown in Appendix A, and a benchmark dataset included 30 viruses from VX Heaven public dataset were considered to experiment and evaluate the proposed method. The investigated sample and its obfuscated program with the CSE were compared with the BUZATU obfuscation by calculating the ability, the resiliency, and the cost objective functions. It is shown that the ability of the CSE is higher than the BUZATU and the resiliency of the CSE is lower than the BUZATU. Therefore, the obfuscation with the CSE is stronger than the BUZATU. Also, the comparison of the CSE and nine state-of-the-art obfuscation methods based on the Mishra criteria represents the competitive advantages of the proposed CSE. The research achievement is proposing a new dynamic obfuscation method (the CSE) that hides the control flow graph, decreases high-cost operations and can be effective in the static program analysis.

References

[1] D. Maiorca, D. Ariu, I. Corona, M. Aresu, and G. Giacinto. Stealth attacks: An extended insight into the obfuscation effects on android malware. *Computers & Security*, 51(1):16–31, 2015. doi:10.1016/j.cose.2015.02.007.

[2] S. Schrittwieser and S. Katzenbeisser. Code obfuscation against static and dynamic reverse engineering. In *International workshop on information hiding*, pages 270–284. Springer, Berlin, Heidelberg, 2011. ISBN 978-3-642-24177-2. doi:10.1007/978-3-642-24178-9_19.

[3] C. Barría, D. Cordero, C. Cubillos, and M. Palma. Proposed classification of malware, based on obfuscation. In *2016 6th International Conference on Computers Communications and Control (ICCCC)*, pages 37–44. IEEE, 2016. ISBN 978-1-5090-1735-5. doi:10.1109/ICCCC.2016.7496735.

[4] B. Hashemzade and A. Maroosi. Hybrid Obfuscation Using Signals and Encryption. *Journal of Computer Networks and Communications*, 2018, 2018. doi:10.1155/2018/6873807.

[5] S. Hosseinzadeh, S. Rauti, S. Laurén, J. Mäkelä, J. Holvitie, S. Hyrynsalmi, and V. Leppänen. Diversification and obfuscation techniques for software security: A systematic literature review. *Information and Software Technology*, 104(8):72–93, 2018. doi:10.1016/j.infsof.2018.07.007.

[6] M. Christodorescu and S. Jha. Static analysis of executables to detect malicious patterns. Technical report, Wisconsin University-Madison on Departement of Computer Sciences, 2006.

[7] S. Romano, C. Vendome, G. Scanniello, and D. Poshyvanyk. A multi-study investigation into dead code. *IEEE Transactions on Software Engineering*, 46(1):71 – 99, 2018. ISSN 1939-3520. doi:10.1109/TSE.2018.2842781.

[8] 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 *2014 International Conference on Collaboration Technologies and Systems (CTS)*, pages 173–180. IEEE, 2014. ISBN 978-1-4799-5158-1. doi:10.1109/CTS.2014.6867561.

[9] O. Mirzaei, de J. M. Fuentes, J. Tapiador, and L. Gonzalez-Manzano. AndrODet: An adaptive Android obfuscation detector. *Future Generation Computer Systems*, 90:240–261, 2019. ISSN 1939-3520. doi:10.1016/j.future.2018.07.066.

[10] S. Alrabaee, L. Wang, and M. Debbabi. BinGold: Towards robust binary analysis by extracting the semantics of binary code as semantic flow graphs (SFGs). *Digital Investigation*, 18:S11–S22, 2016. doi:10.1016/j.diin.2016.04.002.

[11] J. Ge, S. Chaudhuri, and A. Tyagi. Control flow based obfuscation. In *Proceedings of the 5th ACM workshop on Digital rights management*, pages 83–92. ACM, 2005. ISBN 978-1-4799-5158-1. doi:10.1145/1102546.1102561.

[12] I. You and K. Yim. Malware Obfuscation Techniques: A Brief Survey. In *2010 International conference on broadband, wireless computing, communication and applications*, pages 297–300. IEEE, 2010. ISBN 978-1-4244-8448-5. doi:10.1109/BWCCA.2010.85.

[13] E. Konstantinou and S. Wolthusen. Metamorphic virus: Analysis and detection. Technical report, Royal Holloway University of London, 2008.

[14] A. Cimitile, F. Martinelli, F. Mercaldo, V. Nardone, and A. Santone. Formal methods meet mobile code obfuscation identification of code reordering technique. In *2017 IEEE 26th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pages 263–268. IEEE, 2017. ISBN 978-1-

5386-1760-1. doi:10.1109/WETICE.2017.23.

[15] C. Chen, M. Hasan, A. Ghassami, S. Mohan, and N. Kiyavash. REORDER: Securing Dynamic-Priority Real-Time Systems Using Schedule Obfuscation. *arXiv preprint arXiv:1806.01393*, 2018.

[16] Z. Guo, X. Xu, M. M. Tehranipoor, and D. Forte. EOP: An Encryption-Obfuscation Solution for Protecting PCBs Against Tampering and Reverse Engineering. *arXiv preprint arXiv:1904.09516*, 2019.

[17] N. Bitansky and V. Vaikuntanathan. Indistinguishability obfuscation from functional encryption. *Journal of the ACM (JACM)*, 65(6):240–261, 2018. ISSN 1-37. doi:10.1145/3234511.

[18] I. V. Popov, S. K. Debray, and G. R. Andrews. Binary Obfuscation Using Signals. In *USENIX Security Symposium*, pages 275–290, 2007.

[19] 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) 2015*, pages 569–578. Springer, New Delhi, 2015. ISBN 978-81-322-2693-2. doi:10.1007/978-81-322-2695-6_48.

[20] 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*, 11(2):75–88, 2015. ISSN 2263-8733. doi:10.1007/s11416-014-0222-y.

[21] G. Shanmugam, R. M. Low, and M. Stamp. Simple substitution distance and metamorphic detection. *Journal of Computer Virology and Hacking Techniques*, 9(3):159–170, 2013. ISSN 2263-8733. doi:10.1007/s11416-013-0184-5.

[22] A. H. Toderici and M. Stamp. Chi-squared distance and metamorphic virus detection. *Journal of Computer Virology and Hacking Techniques*, 9(1):1–14, 2013. ISSN 2263-8733. doi:10.1007/s11416-012-0171-2.

[23] N. Runwal, R. M. Low, and M. Stamp. Opcode graph similarity and metamorphic detection. *Journal in computer virology*, 8(1-2):37–52, 2012. ISSN 2263-8733. doi:10.1007/s11416-012-0160-5.

[24] B. B. Rad, M. Masrom, and S. Ibrahim. Opcodes histogram for classifying metamorphic portable executables malware. In *2012 International Conference on e-Learning and e-Technologies in Education (ICEEE)*, pages 209–213. IEEE, 2012. ISBN 978-1-4673-1679-8. doi:10.1109/ICeLeTE.2012.6333411.

[25] T. H. Austin, E. Filiol, S. Josse, and M. Stamp. Exploring hidden markov models for virus analysis: a semantic approach. In *2013 46th Hawaii International Conference on System Sciences*, pages 209–213. IEEE, 2013. ISBN 978-1-4673-5933-7. doi:10.1109/HICSS.2013.217.

[26] C. Collberg, C. Thomborson, and D. Low. A taxonomy of obfuscating transformations. Technical report, Department of Computer Science, University of Auckland, 1997.

[27] S. Martinez. Source code obfuscation by mean of evolutionary algorithms. *Internship Report, University of Luxemborg August, 2012*, 2011.

[28] P. OKane, S. Sezer, and K. McLaughlin. Obfuscation: The hidden malware. *Journal in computer virology*, 9(5):41 – 47, 2011. ISSN 1540-7993. doi:10.1109/MSP.2011.98.

[29] R. L. Rivest and J. C. Schuldt. Spritz-a spongy RC4-like stream cipher and hash function. *IACR Cryptology ePrint Archive*, 2016.

[30] S. Alam, R. N. Horspool, I. Traore, and I. Sogukpinar. A framework for metamorphic malware analysis and real-time detection. *computers & security*, 48:212–233, 2015. doi:10.1016/j.cose.2014.10.011.

[31] 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*, 231: 64–82, 2013. doi:10.1016/j.ins.2011.08.020.

[32] A. Shabtai, R. Moskovitch, C. Feher, S. Dolev, and Y. Elovici. Detecting unknown malicious code by applying classification techniques on opcode patterns. *Security Informatics*, 1(1), 2012. doi:10.1186/2190-8532-1-1.

[33] P. OKane, S. Sezer, and K. McLaughlin. Methods for obfuscating Java programs. *Journal of Mobile, Embedded and Distributed Systems*, 4(1):25–30, 2012.

[34] W. Wong and M. Stamp. Hunting for metamorphic engines. *Journal in Computer Virology*, 2(3):211–229, 2006. ISSN 1772-9890. doi:10.1007/s11416-006-0028-7.

A Appendix I

The original program as a small sample is considered as follow:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void matrix mul matrix(size_t N, double C[4*N][4*N],
      double A[4*N][4*N], double [4*N][4*N]) }
 size_t i , j ,k;
 for ( i=0; i<4*N; i++) }
     for ( j=0; j<4*N; j++) }
         for(k=0;k<4*N;k++)
}
C[i ][ j]+=A[i][k] * B[k][ j ];
         {
```

```
    {
{
{
void parse matrix(int n, char * s,
          double a[4*n][4*n])}
int sl = strlen(s) ;
char buffer \cite{687};
int i=-1;
int j = 0;
int p = 0;
int k = 0;
for (k =0;k<sl;k++){
    if (s [k] == '['){
       j = 0;
       i++;
    }
else if (s [k] == ' '){
     buffer [p] = '\0' ;
     a[ i ][ j ] = atof(buffer) ;
     p =0;
     j++;
}
else if (s [k] == ']'){
     buffer [p] = '\0' ;
     a[ i ][ j ] = atof(buffer) ;
     p =0;
}else{
    buffer [p] = s[k ];
     p++;
    }
  }
}
int main(int argc, char ** argv) {
    int n = atoi(argv\cite{1}) ;
    double (*a)[4*n][4*n],
           (*b)[4*n][4*n],(* c)[4*n][4*n];
    a=malloc(sizeof(double)*16*n*n);
    b=malloc(sizeof(double)*16*n*n);
    c=calloc(sizeof(double),16*n*n);
    int i=0;
    int j = 0;
    parse matrix(n,argv \cite{2},(*a)) ;
    parse matrix(n,argv \cite{3},(* b));
    matrix mul matrix(n,*c,*a,*b);
    for(i=0;i<4*n;i++)
     printf ("%f ",(*c) [ i ][ i ]) ;
    free (a);
    free (b);
    free (c) ;
  return 0;
  }
```

Bahare Hashemzade is a lecturer and researcher at Electrical and Computer Engineering, University of Torbat Heydarieh since 2016. She has graduated from M.Sc. degree in Information Science from Birjand University in 2015. Her interest fields are information technology, obfuscation and data mining.

Majid Abdolrazzagh-Nezhad is a lecturer and researcher as an assistant professor at the computer engineering department of the University of Bozorgmehr Qaenat since 2013. He was dean of the faculty of computer science between 2013 to 2016. Abdolrazzagh-Nezhad guides Master and Ph.D. students of Islamic Azad University of Birjand since 2016. He has graduated at the Ph.D. degree in Computer Science from Information Science and Technology, Faculty of the National University of Malaysia (UKM) in 2013. Also, he received his Master's degree in Operation Research from the University of Sistan and Blochestan in 2007 and received his bachelor's degree from the University of Birjand in 2004. His interest fields are artificial intelligence, optimization, data mining, scheduling and uncertain systems. He is a young Professionals member of the Institute of Electrical and Electronics Engineering (IEEE) and reviewer of valid journals such as Information Sciences, Applied Soft Computing, Soft Computing, International Journal of Production Research, IEEE Transaction on Industrial Electronics and IEEE Transaction on Industrial Informatics.