International Journal of Grid and Distributed Computing
Vol. 14, No. 1, (2021), pp. 11-19

Prevent the Source code from Software reverse Engineering using Hybrid Techniques

K L S Sirisha

Assistant Professor, CSE Dept., Keshav Memorial Institute of Technology, Narayanaguda, Telangana

klssirisha@gmail.com

Abstract

Here a new Hybrid Obfuscation Technique is proposed to prevent prohibited Reverse Engineering. The proposed hybrid technique contains three approaches; first approach is string encryption. The string encryption is about adding a mathematical equation with arrays and loops to the strings in the code to hide the meaning. Second approach is renaming system keywords to Unicode to increase the difficulty and complexity of the code. Third approach is transforming identifiers to junk code to hide the meaning and increase the complexity of the code. An experiment is conducted to evaluate the proposed Hybrid Obfuscation Technique. The experiment contains two phases; the first phase was conducting reverse engineering against java applications that do not use any protection to determine the ability of reversing tools to read the compiled code. The second phase was conducting reverse engineering against the proposed technique to evaluate the effectiveness of it. The experiment of the hybrid obfuscation technique was to test output correctness, syntax, reversed code errors, flow test, identifiers names test, methods, and class's correctness test. With these parameters, it was possible to determine the ability of the proposed technique to defend the attack. The experiment has presented good and promising results, where it was nearly impossible for the reversing tool to read the obfuscated code. Even the revealed code did not perform as well as original and obfuscated code.

INDEX TERMS: *Obfuscation techniques, reverse engineering (RE), anti-reverse engineering, intellectual property, software security, piracy.*

I. INTRODUCTION

Intellectual property theft is one of the most challenging problems of technological era. According to the Business software alliance, global software piracy rate went noticeably high which lead to a loss of $53billion in 2018. Due to the lack of security, software vendors have implemented security algorithms, techniques, and tools, but with the help of reverse engineering tools, software reversers are able to reveal the security algorithms to extract the original code from the source file [1].

IT industry loses tens of billions of dollars due to security attacks such as reverse engineering. Code obfuscation Techniques experienced such attacks by transforming code into patterns that resist the attacks. The use of popular languages such as java increases an attacker's ability to steal intellectual property (IP), as the source program is translated to an intermediate format retaining most of the information such as meaningful variables names present in source code. An attacker can easily reconstruct source code from intermediate formats to extract sensitive information. Hence, there is a need for development of techniques and schemes to obfuscate sensitive parts of software to protect it from reverse engineering attacks [2]. Every organization is having its own intellectual property and it is a big challenge for them to protect their data from software piracy or reverse engineering. Reverse Engineering may damage the software purchaser's business directly. General ways to protect intellectual property, legally or technically. Legally, such as getting copyrights or signing legal contracts against creating duplicates. Technically where the owners implement protection for their software. The better idea is to use obfuscation, which is a novel area of research in the field of software protection, and gaining more importance in this present digital era [3].

Obfuscation is known to be the most common and effective technique to prevent prohibited Reverse Engineering. However, none of the current obfuscation techniques meet and satisfy all the obfuscation effectiveness criteria to resist reverse Engineering [4] [5]. A determined attacker, after spending enough time to inspect obfuscated code, might locate the functionality to alter the functions and succeed. The reversing tools that are able to perform anal can attack the renaming obfuscation, layout obfuscation, and source code obfuscation to create a new name for the identifiers that are used in the source file [6].

All theoretical research on software protection via obfuscation typically points to negative results in terms of the existence of perfect obfuscators. There is no general obfuscation algorithm exists that can hide all information leaked by a variant program based on the notion of a virtual black box [7]. The basic impossibility result states that it is impossible to achieve perfect semantic security where the variant leaks no more information than the input/output relationships of the original program [8].

Most of the developers have practiced using only one obfuscation technique to protect the code and used the technique part only of the code. Having one technique to protect the code is proven not to be effective enough to prevent prohibited reverse engineering these approaches do not help to protect the software when the attacker is the end- user. A determined attacker, after spending enough time to inspect obfuscated code, might locate the functionality to alter the functions. Obfuscation techniques are implemented with other approaches, such as code replacement/update, code-tampering detection, protections updating by that the attackers get a limited amount of time to complete their objective.

Reversing tools are currently advanced as they can create new code from the obfuscated code that performs the same output even though the original code is obfuscated [9]. It is necessary to enhance the source code obfuscation to use different approaches from the renaming techniques in one source file to increase the confusion and complication [10]. Ordinary obfuscation techniques do not have the ability to prevent reverse engineering, as the reversing tools are very advanced and can analyze the code. Having an ordinary obfuscation technique is equal to not having one at all in the source file. Based on the researchers a merged obfuscation technique is well known to provide better protection that having an obfuscation technique that contains only one approach of protection [11].

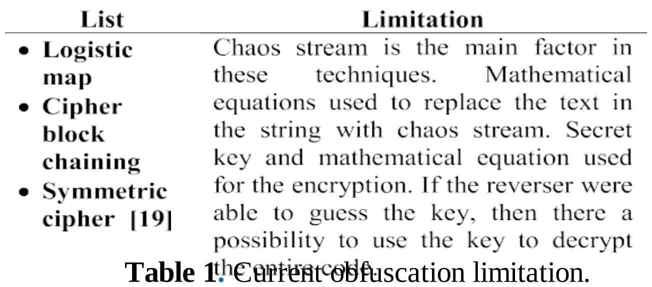
II. RELATED WORK

Anti-reverse engineering techniques are going towards obfuscation due to its power to transform the code into different presentations. Obfuscation opens a room for innovation where the developer can use different languages in programming, a language that only the owner can understand what it is and what it does [12]. Obfuscation consists of code transformations that make a program more difficult to understand by changing its structure [13]. While preserving the original functionalities. The obfuscation process aims to modify the compiled code such that its functionalities are preserved, while its understandability is compromised for a human reader or the de-compilation is made unsuccessful.

Obfuscation methods include code re-ordering, transformation to replace meaningful identifier names in the original code with meaningless random names (identifier renaming), junk code insertions, unconditional jumps, conditional jumps, transparent branch insertion, variable reassigning, random dead code, merge local integers, string encoding, generation of bogus middle-level code, suppression of constants, meshing of control flows and many more.

Several approaches and techniques have been developed, based on the application of different kinds of transformation to the original source (or machine) code [14]. Obfuscating transformations can be classified according to their target and the kind of modification they operate on the code [15] [16]. Such techniques try to hide the structure and the behavior information embedded in the identifiers of a Java program by replacing them with meaningless or confounding identifiers to make more difficult the task of the reverse engineer. It is worth noticing that the information associated with an identifier is completely lost after the renaming.

Furthermore, by replacing the identifiers of a Java byte- code with new ones that are illegal with respect to the Java language specification, such techniques try to make the de-compilation process impossible or make the de-compiler return unusable source code. Such effects will not be easily countered by the existing de-compilation technologies forcing the cracker to spend lots of time to understand and debug the decompiled program manually. According to the power of obfuscation techniques, it is effective to delay Reverse Engineering. However, there are certain limitations that appear in current techniques.



List	Limitation
• Logistic map • Cipher block chaining • Symmetric cipher [19]	Chaos stream is the main factor in these techniques. Mathematical equations used to replace the text in the string with chaos stream. Secret key and mathematical equation used for the encryption. If the reverser were able to guess the key, then there a possibility to use the key to decrypt the entire code.

Table 1. Current obfuscation limitation.

Table 1 displays some of the limitations. The above techniques are common protection. An ordinary user will not be able to break in the software program, but the reverser will be able to break in easily. These techniques do

2.1 String Encryption Approach

A mathematical equation with character array and loops were used to encrypt the strings in the source code. Encrypting the strings create confusion during de-compiling. The reversing tool will not be able to translate the symbols created by the mathematical equation. The compiler will not be able to translate the symbols which were converted to byte code during compiling time. The purpose of the string encryption is to create a chaos stream in the source file and in the reversed file after decompiling. The advantage of string encryption is that the mathematical which was used to create the chaos stream can be used N times in the source code, also several X amounts of mathematical equations can be used in the same source file. The more chaos stream created in the source file the more confusion created during decompiling.

The mathematical equations that were used in the source file for the sake of this research were derived from the concept that Java programming language provides the feature where the mathematical equation can be used to convert the characters into different symbols.

Usually, the equation will contain a fixed value to ensure accurate output. For the sake for this research, the fixed value for the equation is 2 that can be considered as the value of X. There are other two values in the equation that are the values of Y and Z. The values of Y and Z must be carefully declared and assigned to produce the accurate output.

If the value of Y is 17 then the value of Z is 2. If the value of Y is 18 then the value of Z is 3. If the value of Y is 16 then the value of Z is 1.
According to the above conditions, if the value of Y increments by 1 value then the value of Z must increment by 1 value as well. The assigned value of X is 2, it can be changed as well to increment by 1 value, and then the value of Y must decrement by 3 values to get the calculation right for accurate output. The result of calculating the three values must be always 17, therefore the value of X is fixed but it can decrement by 1 value, to increment the value of Y by 1 value as well. To prevent errors the value of X was

fixed at 2. The values of Y and Z can be incremented and decremented accurately to allow using more mathematical equations in the source file.

The final equation as $\boldsymbol{Char} = \boldsymbol{V}/2 + \boldsymbol{Y} + \boldsymbol{Z}$.

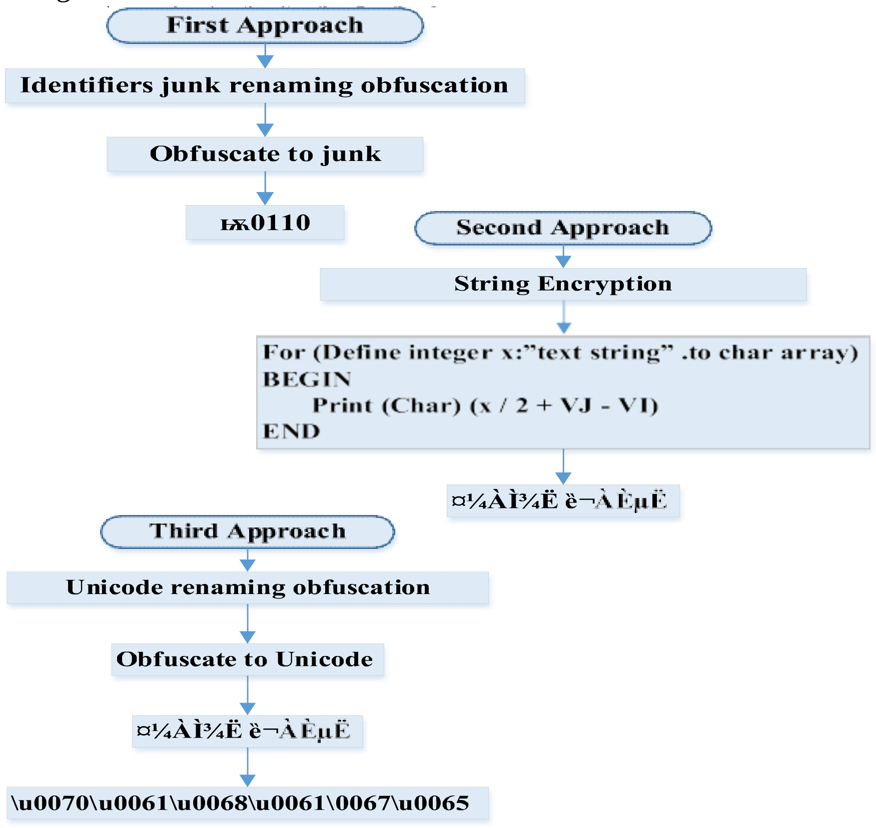
2.2 Identifiers Renaming to Junk Approach

Identifiers will be renamed to junk to hide the meaning of them. The purpose of this conversion is to mislead the reverser while reading the source file. The measurement of this approach was conducted by experiment to validate the effectiveness of the output after obfuscation transformation and to determine the uncovered code after reversing the obfuscated code. The following sections present an explanation in detail about the mechanism of the three approaches.

First section discusses the Unicode renaming obfuscation. Second section discusses string encryption obfuscation. Third section discusses the mathematical equation used to encrypt the strings in the source file. Fourth section discusses the identifiers renaming to junk obfuscation. Fifth section discusses the possibility to merge the three approaches in one source file to create preventative transformation and Displays the results of the merging.

FIGURE 1. Hybrid obfuscation technique.

Fig 1 demonstrates the Hybrid Obfuscation Technique. The proposed technique has changed the form of the code and complicated the look of it. The complication of the codes has created confusion while reading the source file and while reversing the class file.


First Approach
Identifiers junk renaming obfuscation
Obfuscate to junk
ꓘ0110
Second Approach
String Encryption
For (Define integer x:"text string" .to char array)
BEGIN
Print (Char) (x / 2 + VJ - VI)
END
¤¼ÀÌ¾Ë è¬À ÈµË
Third Approach
Unicode renaming obfuscation
Obfuscate to Unicode
¤¼ÀÌ¾Ë è¬À ÈµË
\u0070\u0061\u0068\u0061\0067\u0065

III. Proposed Approaches

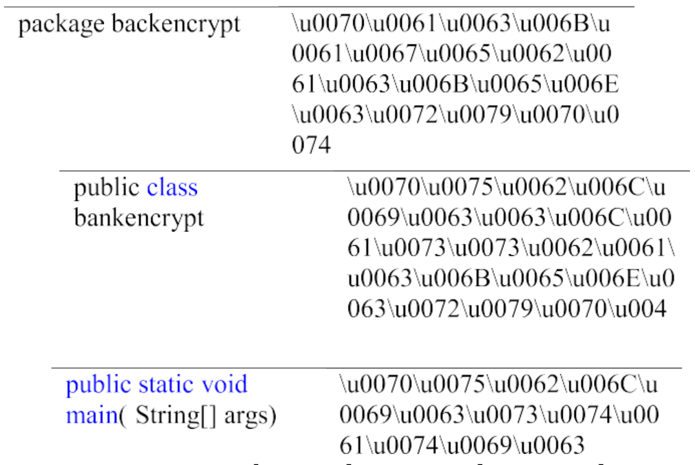
The contribution of this research is to introduce a new hybrid obfuscation technique to overcome the obfuscation of Java programs based on the renaming of the identifiers and string encryption. The proposed technique was based on the hybrid renaming of the identifiers in the source file to create extreme confusion for both the reversing tools the human examining the source file without permission.

Independently of the obfuscating renaming strategy used, it was possible to contrast the obfuscation by renaming the identifiers and string encryption in two phases, to start over- come the preventive obfuscation, then to add type information to the identifiers in the source code to contrast layout obfuscation. In the first phase, the renaming of the hybrid obfuscation technique contains two sections. The first section is to rename the identifiers to junk code to hide the meaning and increase complexity and confuse the de-compiler while reversing. The second section is to replace system keywords with UNICODE.

The second phase, the string encryption, where a group of random mathematical equations is inserted into the strings to encrypt them. A framework of transformation was implemented to present the steps of the hybrid obfuscation technique. The proposed hybrid obfuscation technique includes three phases of renaming. In these phases, three renaming approaches were applied in the source file. The proposed hybrid obfuscation technique aims to confuse or mislead the reverser as much as possible while reading the reversed code after obfuscation. The technique should produce the same output as the original code. The following sections discuss the phases of the hybrid obfuscation technique.

3.1 First Approach Unicode Renaming Obfuscation

Unicode is a standard design that uniquely encodes characters written in any language. Unicode uses Hexadecimal to express the character. Unicode is the standard for encoding, representation, and handling of text on computers. There are 136,755 are defined in the Unicode which grant an opportunity to use it widely [25].



package backencrypt	\u0070\u0061\u0063\u006B\u0061\u0067\u0065\u0062\u0061\u0063\u006B\u0065\u006E\u0063\u0072\u0079\u0070\u0074
public class bankencrypt	\u0070\u0075\u0062\u006C\u0069\u0063\u0063\u006C\u0061\u0073\u0073\u0062\u0061\u0063\u006B\u0065\u006E\u0063\u0072\u0079\u0070\u004
public static void main(String[] args)	\u0070\u0075\u0062\u006C\u0069\u0063\u0073\u0074\u0061\u0074\u0069\u0063

TABLE 2. System keywords converted to Unicode

This research has used Unicode to rename the system keywords, the conversion will lead to confusion, where by the reader will not be able to extract the meaning of the code without manually translating it or using a reversing tool

Algorithm 1 Unicode Transformation

```
BEGIN
Get initial code
Get system keyword codes
Apply UNICODE transformation
END
```

3.2 Second Approach String Encryption Obfuscation

String encryption is well known in most of the programming languages such as C/C, Java, Python, C#, and PHP. There are several methods to encrypt the strings. This section discusses the string encryption obfuscation technique. Java programming language allows mathematical equations to be used with arrays and loops to encrypt the strings in the source file to create a chaos stream as a method to hide the meaning of it.

Current string encryption such as symmetric Cipher inserts only one mathematical equation to hide the meaning. The encryption of the mathematical equation is however readable by the compiler and confuses the reader. For the proposed hybrid obfuscation technique, mathematical string encryption was used to encrypt the strings in the source file. The mathematical encryption was inserted with character array in for loop in the source file.

The purpose of the mathematical equation is to confuse the reader and complicate the look of the code. Increasing the number of the string encryptions and mathematical equations in the source file will complicate the process of decompiling. after compiling the source code to class file which contains the byte code. The following algorithm displays the transformation to UNICODE. Table 2 displays an example of java system keywords converted to Unicode.

3.3 Mathematical Equation to Encrypt Strings

The equation which was used to encrypt the strings in the source code is associated with beneficial attributes associated with non-beneficial attributes Y indicates the ideal (best) value of the considered attribute among the values of the attribute for different alternatives the fixed and best value for the equation is 2 this value will not be changed. In the case of beneficial attributes (i.e., those of which higher values are desirable for the given application), Y indicates the higher value of the attribute, and the highest value which will be used for the equation.

In the case of non-beneficial attributes, Z indicates the lower value of the attribute. Z indicates the lowest value of the considered attribute among the values of the attribute for different alternatives, the lowest value which will be used is 2. In the case of beneficial attributes, Z indicates the lower value of the attribute. In the case of non-beneficial attributes, Y indicates the higher value of the attribute. Below equation displays the string encryption transformation.

$Char = V/2+Y+Z$.

The following algorithm displays the steps of string encryption.

Algorithm 2 String Encryption

```
BEGIN
Get initial code Start String
For (Define integer x:''text string''.to char array) BEGIN
Print (Char) $(x / 2vj - vi)$ END
END
Print new line
END
```

According to the above algorithm, there is a possibility to use a different mathematical equation for string encryption to transform the characters to different symbols, Fig. 2. demonstrates an example of code after applying string encryption in the source file.

3.4 Third Phase Identifiers Renaming to Junk Obfuscation

The third phase, which was used in the new proposed hybrid obfuscation technique, was renaming identifiers to junk obfuscation. The main purpose of the junk renaming is to create a complicated code that is difficult to understand and difficult to get a meaning out of it.

The renaming junk obfuscation works for confusing the reversing tool that leads to wrong analysis, therefore produce wrong codes. Junk conversion creates an opportunity to create a variety of languages while developing the software for the sake of protection. The class file contains the junk code after compiling the source file.

After using the junk conversion, the converted code will be converted again to junk in the class file which increases the level of protection. The following algorithm is to trans- form identifiers into junk. Fig 4. demonstrates the code after converting to junk.

Algorithm 3 Identifiers Transformation

```
BEGIN
Get initial code Get identifiers
Transformation Begin Convert characters to junk Transformation END
END
```

3.5 Applying Hybrid Obfuscation Technique in the Source Code

Java development is based on object orientation whereas the compiler runs the application based on components, unlike structured programs that are developed by C programming language. Therefore, obfuscating the code will not create problem while compiling into machine language or byte code. To use this hybrid obfuscation technique, certain steps must be followed; first step is to use the object junk renaming obfuscation. This conversion must be done first to prevent confusion and errors when the obfuscation process is running. Second step is to use string encryption obfuscation; this technique must be done secondly, for the developer to encrypt all the strings at once. Final step is Unicode renaming obfuscation technique. Carrying out the Hybrid obfuscation technique increases the security level of the code and complicates the reversing process. The string encryption makes the obfuscation technique more effective in terms of securing the code, as it contains so many symbols that help to confuse the de-compiler while parsing and analysis. The following snippet displays the code before and after using obfuscation.

The Hybrid Obfuscation Technique is effective as it con- fuses and the reversing tool while reversing the class file. The reversing tool has translated the junk code to another junk code, and it has translated the encrypted strings to random meaningless numbers. The reversing tool could not perform an analysis of the obfuscated code. Reversing tools have produced errors and illogical code after reversing the obfuscated code. This Hybrid Obfuscation Technique was tested to evaluate the effectiveness and correctness of the code with four reversing tools.

IV. Experiment Procedure

The experiment consists of two phases. First phase is to test the applications that are not using any protection technique to determine the need of java applications for protection. The applications that are used for the experiment are procedural application, image application, object-oriented application, and an obfuscated application [26]. The parameters used for the experiment are:

a. Output correctness
b. Syntax and flow
c. Compiling testing
d. Identifiers names

Second phase of experiment was an attempt to reverse the code after inserting the hybrid obfuscation technique into the source code. The reversing tools used for the experiment are CAVAJ, JAD, DJ, and JD. The parameters for the experiment are

a. Output correctness
b. Syntax
c. Error testing (Reversing code compiling test)
d. Flow test
e. Identifiers names.
f. Decrypt string test

g. During the experiment, we have calculated total lines of code (LOC) of the reversed code before and after obfuscation, total errors of compiled reversed file before after obfuscation. This calculation has numerically determined the strength of the proposed hybrid obfuscation technique. the hypothesis of the experiment is:

h. *H1:* Hybrid obfuscation techniques do not significantly decrease the ability of the reverser to change the original code.

i. *H2:* Hybrid obfuscation techniques significantly decrease the ability of the reverser to change the original code.

V. CONCLUSION

Due to the increasing piracy of the software, a novel attempt is made to implement a hybrid obfuscation technique in this research. Typically, after obfuscation, the complexity of the code increases according to logically as well as structurally because of the insertion, removal, or rearrangement of the code. The proposed technique presented has been found to be effective.

The future work is aimed at the development of a frame- work for automation of the presented technique and to pro- vide as a plug-in to support developers to customize the method of obfuscation. The aim has been set to implement the proposed idea for large-scale software protection and improvement.

Implementing a hybrid obfuscation technique is highly recommended and proved successful. Currently, most reveres and companies are very much interested to reverse complicated software applications rather than implementing fresh ones. Implementing the hybrid obfuscation technique will make them struggle to understand the obfuscated code, as it requires a long time to get a meaning out of it.

The proposed technique was evaluated empirically with the experiment. Four reversing tools were used for the experiment to determine the ability to discover the code and analyze it. An interview was conducted with programming experts from the industry. Results from the experiment and interview supported the research's hypothesis and objectives. According to the experiments, the proposed technique has shown promising results, where the objectives of the research are met, where a chaos stream was created during reversing, junk code was generated from the reversing tool, and an extra layer of garbage created from the reversing tool as a result of the inability to read the obfuscated code.

REFERENCES

[1] M. ul Iman and A. F. M. Ishaq, ''Anti-reversing as a tool to protect intellectual property,'' in *Proc. 2nd Int. Conf. Eng. Syst. Manage. Appl.*, Apr. 2010, pp. 1–5.

[2] N. D. Gomes, ''Software piracy? An empirical analysis software piracy?: An empirical analysis,'' Univ. De Combra, Coimbra, Portugal, Tech. Rep., 2014.

[3] C. K. Behera and D. L. Bhaskari, ''Different obfuscation techniques for code protection,'' *Procedia Comput. Sci.*, vol. 70, pp. 757–763, Jan. 2015.

[4] M. Popa, ''Techniques of program code obfuscation for secure software,'' *J. Mobile, Embedded Distrib. Syst.*, vol. 3, no. 4, pp. 205–219, 2011.

[5] A. Kulkarni and R. Metta, ‘‘A code obfuscation framework using code clones,’’ in *Proc. 22nd Int. Conf. Program Comprehension ICPC*, 2014, pp. 295–299.

[6] A. K. Dalai, S. S. Das, and S. K. Jena, ‘‘A code obfuscation technique to prevent reverse engineering,’’ in *Proc. Int. Conf. Wireless Commun., Signal Process. Net. (WiSPNET)*, Mar. 2017, pp. 828–832.

[7] P. Junod, J. Rinaldini, J. Wehrli, and J. Michielin, ‘‘Obfuscator-LLVM - software protection for the masses,’’ in *Proc. IEEE/ACM 1st Int. Workshop Softw. Protection*, May 2015, pp. 3–9.

[8] S. Qing, W. Zhi-yue, W. Wei-min, L. Jing-liang, and H. Zhi-wei, ‘‘Technique of source code obfuscation based on data flow and control flow tans- formations,’’ in *Proc. 7th Int. Conf. Comput. Sci. Edu. (ICCSE)*, Jul. 2012, pp. 1093–1097.

[9] P. Kanani, K. Srivastava, J. Gandhi, D. Parekh, and M. Gala, ‘‘Obfuscation: Maze of code,’’ in *Proc. 2nd Int. Conf. Commun. Syst., Comput. Appl. (CSCITA)*, Apr. 2017, pp. 11–16.

[10] S. Hosseinzadeh, S. Rauti, S. Laurén, J. M. Mäkelä, J. Holvitie, S. Hyrynsalmi, and V. Leppänen, ‘‘Diversi fi cation and obfuscation techniques for software security?: A systematic literature review,’’ *Inf. Softw. Technol.*, vol. 104, pp. 72–93, Jul. 2018.

[11] D. Hofheinz, J. Malone-Lee, and M. Stam, ‘‘Obfuscation for cryptographic purposes,’’ *J. Cryptol.*, vol. 23, no. 1, pp. 121–168, Jan. 2010.

[12] T. Winograd, H. Salmani, H. Mahmoodi, and H. Homayoun, ‘‘Preventing design reverse engineering with reconfigurable spin transfer torque LUT gates,’’ in *Proc. 17th Int. Symp. Qual. Electron. Design (ISQED)*, Mar. 2016, pp. 242–247.

[13] M. H. BinShamlan, M. A. Bamatraf, and A. A. Zain, ‘‘the impact of control flow obfuscation technique on software protection against human attacks,’’ in *Proc. 1st Int. Conf. Intell. Comput. Eng. (ICOICE)*, Dec. 2019, pp. 2–6.

[14] Y. Peng, J. Liang, and Q. Li, ‘‘A control flow obfuscation method for Android applications,’’ in *Proc. 4th Int. Conf. Cloud Comput. Intell. Syst. (CCIS)*, Aug. 2016, pp. 94–98.

[15] D. Pizzolotto and M. Ceccato, ‘‘[Research paper] obfuscating java pro- grams by translating selected portions of bytecode to native libraries,’’ in *Proc. IEEE 18th Int. Work. Conf. Source Code Anal. Manipulation (SCAM)*, Sep. 2018, pp. 40–49.

[16] S. Cimato, A. De Santis, and U. Ferraro Petrillo, ‘‘Overcoming the obfuscation of java programs by identifier renaming,’’ *J. Syst. Soft.*, vol. 78, no. 1, pp. 60–72, Oct. 2005.