ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/368574949

Hybrid Obfuscation Technique to Protect Source Code From Prohibited Software Reverse Engineering

Conference Paper · February 2023

CITATIONS
0

READS
38

5 authors, including:

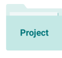
Asmaa Mahfoud Alhakimi
Management and Science University
5 PUBLICATIONS 3 CITATIONS
SEE PROFILE

Abu Bakar bin Md Sultan
Universiti Putra Malaysia
110 PUBLICATIONS 954 CITATIONS
SEE PROFILE

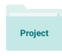
Abdul azim abdul ghani
Universiti Putra Malaysia
207 PUBLICATIONS 1,530 CITATIONS
SEE PROFILE

Norhayati Mohd Ali
Universiti Putra Malaysia
37 PUBLICATIONS 91 CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:



Usability Evaluation Framework for Smartphone Applications View project



Obfuscation to prevent reverse engineering View project

All content following this page was uploaded by Asmaa Mahfoud Alhakimi on 17 February 2023.

The user has requested enhancement of the downloaded file.

https://t.me/learningnets

IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received August 20, 2020, accepted September 12, 2020, date of publication October 2, 2020, date of current version October 23, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3028428

Hybrid Obfuscation Technique to Protect Source Code From Prohibited Software Reverse Engineering

ASMA'A MAHFOUD HEZAM AL-HAKIMI, ABU BAKAR MD SULTAN, ABDUL AZIM ABDUL GHANI, (Member, IEEE), NORHAYATI MOHD ALI, AND NOVIA INDRIATY ADMODISASTRO

Department of Software Engineering and Information System, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia (UPM), Serdang 43400, Malaysia

Corresponding author: Asma'a Mahfoud Hezam Al-Hakimi (selfemoon@gmail.com)

This work was supported by the Universiti Putra Malaysia.

ABSTRACT In this research, a new Hybrid Obfuscation Technique was 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 was 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 classes 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 2008. 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 [2]. 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 [3]. 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. There are two general ways

The associate editor coordinating the review of this manuscript and approving it for publication was Mervat Adib Bamiah.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

https://t.me/learningnets

to protect intellectual property, legally or technically [4]. Legally, such as getting copyrights or signing legal contracts against creating duplicates. And 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 [5].

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 [6], [7].

A determined attacker, after spending enough time to inspect obfuscated code, might locate the functionality to alter the functions and succeed. The renaming obfuscation, layout obfuscation, and source code obfuscation can be attacked by the reversing tools that are able to perform analysis to create a new name for the identifiers that are used in the source file [8].

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 [9]. 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 [10].

Most of the developers have practiced using only one obfuscation technique to protect the code and used the technique for certain 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 [11]. 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 [12].

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 [13].

This article contains several sections, first section is to discuss the related work to obfuscation techniques, then discuss the limitations of the current obfuscation techniques. The contribution is discussed through phases. Experimentation will be discussed in two phases. The first phase was an experiment against the java applications that do not use protection against reverse engineering. The second phase of experimentation is an experiment against the proposed hybrid technique to determine the effectiveness of it. Then a comparison between the two phases was discussed.

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 [14]. Obfuscation consists of code transformations that make a program more difficult to understand by changing its structure [15]. 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 [16]. Obfuscating transformations can be classified according to their target, and the kind of modification they operate on the code [17], [18]. Obfuscation techniques based on the renaming of the identifiers have such techniques that can be classified as a form of layout obfuscation since they reduce the information available to a human reader which examines the target program, or of preventive obfuscation, since they aim to prevent the de-compilation or to produce an incorrect Java source code [6]. 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 bytecode 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

https://t.me/learningnets

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. 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 not prevent reverse engineering.

Reversing tools do not require a long time or days to break the software program. An enhancement of obfuscation techniques was possible by merging certain approaches of obfuscation techniques.

The most effective obfuscation technique that can be merged is string encryption and renaming techniques. String encryption was very beneficial when it was used with a mathematical equation to create a chaos stream while de-compiling. The renaming approach takes two parts, first part where identifiers were converted to junk to hide their meaning, second part where the system keywords were converted to UNICODE to increase the complication of the source file look, and to prevent reading it in case it was stolen.

III. CONTRIBUTION

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 overcome 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 technique can be used to support many languages such as Arabic, English, Chinese, and so on. Using this technique opens a possibility to program using different languages instead of English which increases the level of protection. 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.

TABLE 1. Current obfuscation limitation.

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.
• **Renaming** • **Hiding [20]**	These techniques Aim on hiding the code's function and changing layout. This is harder to understand but not impossible on Reverse Engineering process. These tools might hide the code. However. The Reverse Engineering process is possible.
Key hiding obfuscation [21]	This technique focuses on executable file, it does not focus on the source code and class file. Reversing tools can find the key and break it to get to the source file and perform analysis on the code.
Encryption [22]	This technique focuses on cryptographic techniques to encrypt and pack executable code inside binary file. The limitation is the lawful (export) regulations imposed on programming language vendors. Either they limit key or round sizes, or they leave just stubs for restricted classes. Encryption uses longer keys to provide better security. While the longer key length to slower encryption speed.
Packing[23]	It packs the entire code into one package. The reversing tool can unpack code and create new one that is useful and produces same output as original.
Classes combination obfuscation [24]	Only focuses on hiding the classes by combining them together. Reversing tools can create new classes and uncombine classes. Reversing tools contain a great analysis feature that can find the classes' trees and connection.
Junk code obfuscation	Only focuses on changing the entities names. The renaming aims to confuse the reverser rather than preventing. Reversing tools creates new names for the entities. Then the reverser can use refactor feature to create meaningful names.

https://t.me/learningnets

A. 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 [22]. 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 $\boldsymbol{Char} = \boldsymbol{V}/\mathbf{2} + \boldsymbol{Y} + \boldsymbol{Z}$.

B. UNICODE RENAMING APPROACH

The Unicode transformation was used to rename the system keywords. The purpose of this renaming is to increase the complication of the code in the source file. In this case, when the attacker reads the source file will not be able to get the actual meaning of the code. This approach is very beneficial before reversing, in the case of stealing the source file, the reader will not be able to get the actual meaning of the code, the reader has to translate the entire code to understand the purpose of it. However, the Unicode might be easy to translate, yet the system key words do not carry much meaning as the classes and variables in the functions and methods.

C. 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. 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.

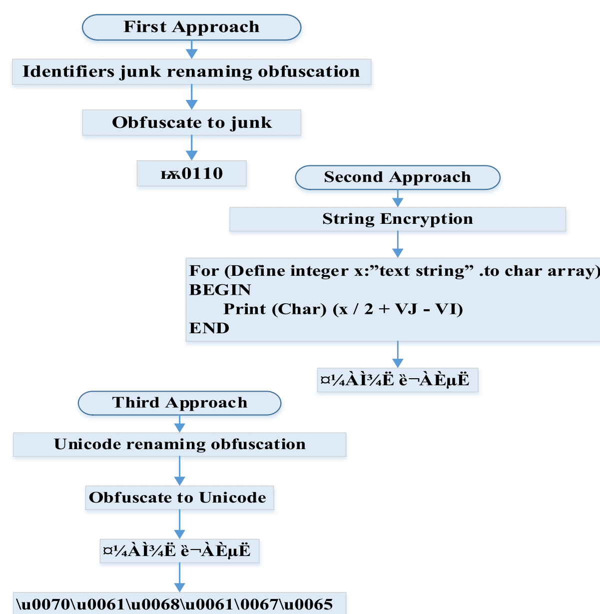


FIGURE 1. Hybrid obfuscation technique.

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].

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

https://t.me/learningnets

TABLE 2. System keywords converted to Unicode.

Java system keywords	Unicode
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

after compiling the source code to class file which contains the bytecode. The following algorithm Displays the transformation to UNICODE. Table 2 displays an example of java system keywords converted to Unicode.

Algorithm 1 Unicode Transformation

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

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.

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 is 17.

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.

$\boldsymbol{Char = V/2 + Y + Z}$.

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 / 2 + vj - 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.

```
void amount()

{

   for ( int ж0110: "¤¾À̀Ì¾Ë ẽ– ÀÈµË".toCharArray()) {
```

FIGURE 2. Code after string encryption.

The approach of using string encryption by applying mathematical equation in array and loops. A (character array) and (For loop) help to convert the messages from English to symbols. Fig 3. demonstrates the actual message before applying string encryption.

https://t.me/learningnets

```
void amount()
  {
         System.out.print("Amount Deposit:");
  }
```

FIGURE 3. Original code before string encryption.

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 transform identifiers into junk. Fig 4. demonstrates the code after converting to junk.

```
捻 ♆ \u003D \u006E\u0065\u0077\u0028\u0029\u003B
ѭ011邏\u003A"□ÌÈËÀ¼¬Æ ¾¤¼¬ "\u002E\u0074\u
0074ѭ011邏\u003A"□ÌÈËÀ¼¬Æ ¾ ¤¼ ¬"
```

FIGURE 4. Code after converting to Junk code.

Algorithm 3 Identifiers Transformation

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

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 confuses 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

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

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:

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

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

https://t.me/learningnets

A. FIRST PHASE OF EXPERIMENT

There are five cases from four different types of software applications that are developed by java. The purpose of the first experiment to demonstrate with evidence the ability of reversing tools to reveal the software's code.

1) PROCEDURAL MATH APPLICATION

Class file was inserted into CAVAJ reversing tool to reveal the original code. Fig 5. demonstrates original and reversed code.

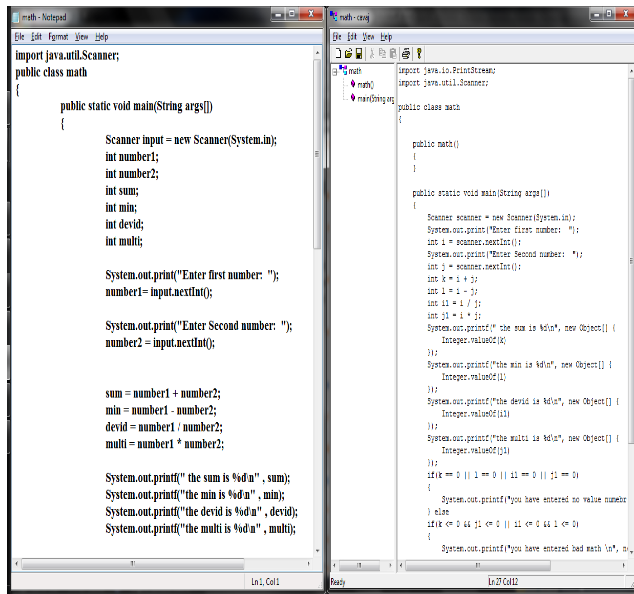


FIGURE 5. Original and reversed code.

The reversing tool has revealed the code of the application with some changes of the identifier's names. Some of the structure and definitions have changed as well. The reversed code has generated same output.

The reversing tool has changed the variables names and the structure of printing; however, the output of the reversed code was same as the original code Table 3 Displays the translation of code before and after reversing.

The next procedure is to calculate the number of lines (LOC), libraries, and methods. The calculation is presented in Table 4. The reversing tool should be able to reveal the code successfully if there is no protection technique used.

Total LOC has increased by four lines as compare to the original code. The reversing tool has added extra libraries.

The reversed code is running efficiently like the original code. To determine how much the reversing tool was able to uncover from the original code, we get total (LOC) of reversed code minus total (LOC) of original code and get the differences. The extra lines define the strength of the reversing tool and the defense level of the original code.

- Total line of reversed code = **48**
- Total lines of original code = **44**
- Difference = **4**

TABLE 3. Code translation.

Original code	Reversed code
Variables names	
number 1	i
number 2	j
sum	k
min	l
devid	i1
multi	j1
Printing structure	
System.out.printf("the sum is %d \n", sum);	System.out.printf("the sum is %d\n, new object [] {Integer.Value of(k)});

TABLE 4. Original code vs reversed code calculation.

Original code		Reversed code	
Objects	**Total**	**Objects**	**Total**
LOC	44	LOC	48
Total libraries	1	Total libraries	2
Total methods	1	Total methods	2

The reversing tools have added up to **four** extra lines of code to the original code. The reversing tool was able to reveal the code, analyze it, and add extra code for better performance. There was not any hidden code. **100%** of the code revealed.

2) CCES, CANCER CARE EXPERT SYSTEM

The class file was inserted to the interface of CAVAJ to reseal the original code. The application was written with Java programming language. The application is based on graphics where there is an interface for the user to use the system, which means, graphic libraries are included in the system. Fig 6 demonstrates the code after reversing.

The reversing tools have discovered the code of the application with some changes of the variables and classes names. Some of the structure and definitions have changed as well.

The reversing tool has produced an error while reversing, however the reversed code is running and provided same output as original code. Table 5 displays the code before and after reversing.

The reversing tool was able to read the code even though there was an error during reversing. This means, the reverser can reverse the unprotected code and determine the meaning of it. Reversing tool was able to do analysis on the original code. The reversing tool has changed the logic of the

https://t.me/learningnets

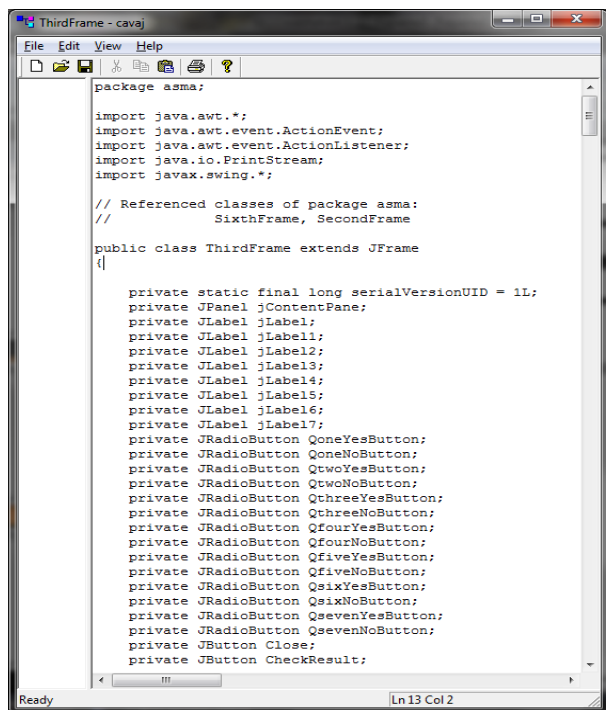


FIGURE 6. Code after reversing.

code, variables names, and structure. With all these changes, the output is same with original code.

The next procedure is to calculate the (LOC), libraries, and methods. The calculation is presented in Table 6. the reversing tool should be able to reveal the code successfully if there is no protection technique used.

Total number of lines has increased by thirty-one lines as compare to the original code. Even there are less libraries generated by the reversing tool. The reversed code is running efficiently like the original code. To find out how much the reversing tool was able to reveal from the original code, we get total (LOC) of reversed minus total (LOC) of original code and get the differences. The extra lines define the strength of the reversing tool and the defense level of the original code.

- Total lines of reversed code = **148**
- Total lines of original code = **117**
- Difference = **31**

The reversing tools have added up to **33** extra lines of code to the original code. The reversing tool was able to reveal the code, analyze it, and add extra code for better performance. **100%** of the code was revealed. Fig 7 demonstrates the output of the reversed code.

The reversing tool had the ability to uncover from the original code, we get total (LOC) of reversed code minus total lines of original code and get the differences. Table 7. Displays the differences.

Total number of lines has increased by **56** lines as compare to the original code. Even there are less libraries were revealed by the reversing tool. The reversed code is running efficiently like the original code. The reversed code is quite different from the original, it however provides the same output as original code.

TABLE 5. Original code before and after reversing.

Original code	Reversed code
import javax.swing.JPanel; import javax.swing.JFrame; import javax.swing.JLabel; import java.awt.Rectangle; import java.awt.Color; import java.awt.Font; import javax.swing.Swing. Constants; import awt.event.KeyEvent; import javax.swing. JRadioBtton; import java awt.Point; import java.awt.Dimension; import javax.swing.JButton; import javax. swing. JOptionPane	import java.awt.event. WindowsAdapter; import java.awt.event. WindowsEvent;
Naming and defining structure	
private JPanel jLabel1= null; private JPanel jLabel11= null; private JPanel jLabel12 = null; private JPanel jLabel13 = null; private JPanel jLabel14 = null; private JPanel jLabel15 = null; private JPanel jLabel16 = null; private JPanel jLabel17 = null ;	private JPanel jLabel1; private JPanel jLabel11; private JPanel jLabel12; private JPanel jLabel13; private JPanel jLabel14; private JPanel jLabel15; private JPanel jLabel16; private JPanel jLabel1;
private JRadioButton QoneYesButton = null; private JRadioButton QoneNoButton = null;	private JRadioButton QoneYesButton; private JRadioButton QoneNoButton;
Code structure	
Changed location of (private JPanel jContentPane = null;)	

TABLE 6. Original and reversed code calculation.

Original code		Reversed code	
Objects	**Total**	**Objects**	**Total**
LOC	117	LOC	148
Total libraries	12	Total libraries	4
Total methods	5	Total methods	5

https://t.me/learningnets

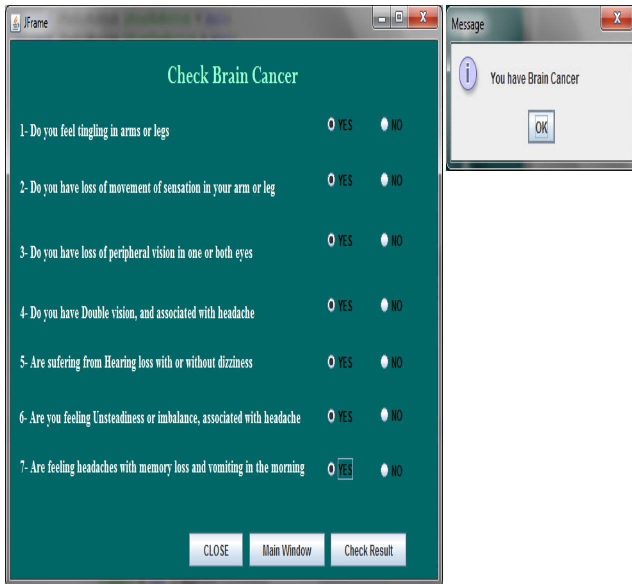


FIGURE 7. Output of reversed code.

TABLE 7. Original and reversed code calculation.

Original code		Reversed code	
Objects	**Total**	**Objects**	**Total**
LOC	421	LOC	477
Total libraries	13	Total libraries	5
Total methods	4	Total methods	4

To find out how much the reversing tools was able to uncover from the original code, we get total (LOC) of reversed code minus total lines of original code and get the differences. The extra lines define the strength of the reversing tool and the defense level of the original code

- Total line of reversed code = **477**
- Total lines of original code = **421**
- Difference = **56**

The reversing tools have added up to **56** extra lines of code to the original code. The reversing tool was able to reveal the code, analyze it, and add extra code for better performance. **100%** of the code was revealed.

B. SECOND PHASE OF EXPERIMENT

In the experimentation, four reversing tools were used against the hybrid obfuscation technique. The purpose of the experiment is to determine the effectiveness of the technique and how much can the reversing tool uncover and read from the obfuscated code.

Four reversing tools were used for this experiment; the tools are CAVAJ, JAD, DJ, and JD. Table 8 Displays the parameters used for every reversing tool. At the end of the experiment, the results were compared with the research

TABLE 8. Parameters used for the second phase experiment.

Reversing tool **Parameters**	**CAVAJ**	**JAD**	**DJ**	**JD**
Output correctness	✓	✓	✓	
Syntax	✓			
Reversed code error	✓	✓		
Flow	✓			
Identifiers names	✓	✓	✓	✓
Methods & classes correctness		✓		
De-Crypt string test	✓			

hypothesis to determine the success of the new proposed hybrid obfuscation technique.

The parameters are distributed among the reversing tools based on their behavior towards the obfuscated code. For example, JD only tested the identifiers names because it can reveal the entire code, therefore, there was no need to test the rest of parameters.

C. FIRST PRESENTATION OF TOOLS WILL BE CAVAJ REVERSING TOOL

1) CAVAJ REVERSING TOOL, (OUTPUT CORRECTNESS) TEST

This reversing tool was used to de-compile hybrid obfuscated code. Fig 8 demonstrates the output of reversed obfuscated code.

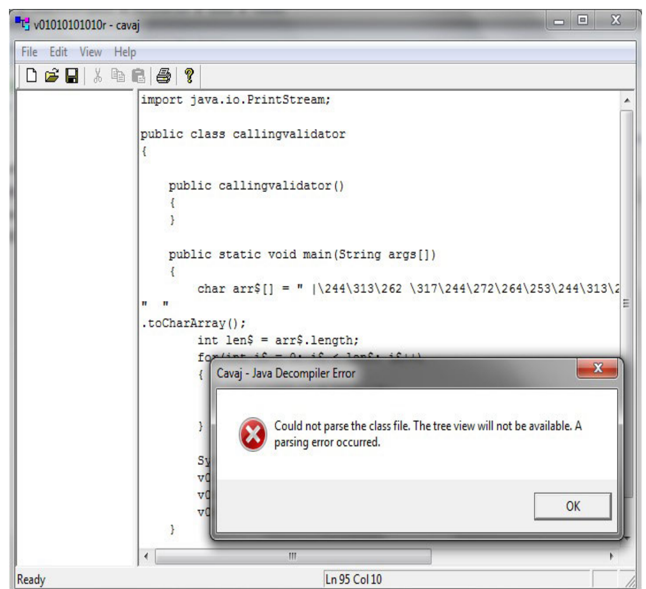


FIGURE 8. Output of reversed code.

https://t.me/learningnets

The reversing tool has generated an error message while reversing the obfuscated code. Due to the string encryption used in the hybrid obfuscation technique, the compiler was not able to read it and translated.

The output of the messages in the code was a series of random numbers that have no meaning. The reversing tool has generated an error message notifies that parsing was not possible.

To determine the strength of the reversing tool against the hybrid technique, total (LOC) of the reversed original file will be compared with total (LOC) of obfuscated reversed file.

- Total LOC of original reversed file: **48**
- Total LOC of obfuscated reversed file: **20**
- Difference between original file and reversed file: **48 - 20 = 28**

The reversing tool was not able to discover **28** lines of obfuscated code. There was some series of random numbers were added to the decompiled code that leaded to error during running the code.

2) CAVAJ REVERSING TOOL, SYNTAX TEST

The reversing tool has generated an error message while reversing the obfuscated code. Due to the identifiers renaming used in the hybrid obfuscation technique, the compiler was not able to read the junk and was not able to translate it or analyze it. Fig 9 demonstrates the output of reversed obfuscated code.

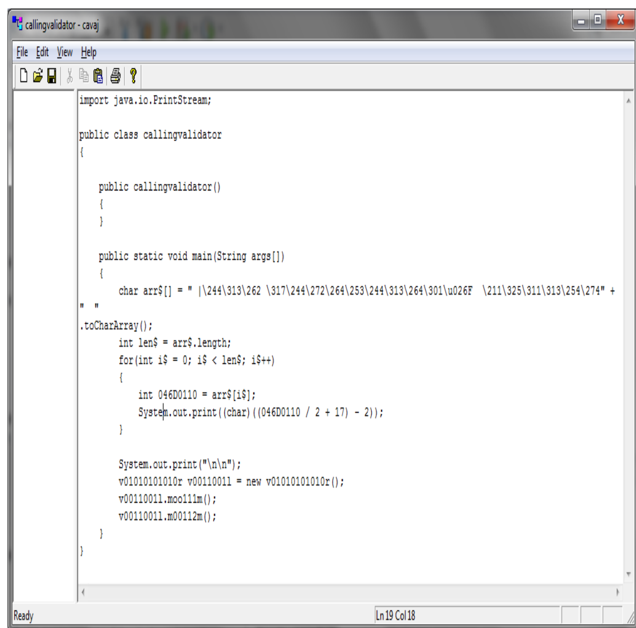


FIGURE 9. Results of reversed obfuscated code.

Table 9 displays the translation of the unreadable output.

The reversing tool was not able to read the encrypted strings, and encrypted identifiers. Due to heavy obfuscation, the overall flow of the code has changed after reversing.

To determine the strength of the reversing tool against the hybrid technique, total number of lines of the reversed original file will be compared with total number of lines of obfuscated reversed file.

- Total LOC of original reversed file: **48**
- Total LOC of obfuscated reversed file: **21**
- Difference between original file and reversed file: **48 - 21 = 27**

The reversing tool did not have the ability to discover **27** lines of obfuscated code. There was a series of random numbers added to the decompiled code that leaded to error Table 11 displays the analysis of the output.

TABLE 9. Unreadable code translation.

Output	Translation
import java .io.PrintStream	Extra library added that has no use
\244\313\262\317\244\264\253\244\313\264\301\u026F\211\325\311\313\254\274	Original message not readable by the tool
V01010101010r v00110011 =new v01010101010r();	Variable names have changed
V00110011.m00111m();	
V00110011.m001112m();	

TABLE 10. Output analysis.

Output	Analysis
Code Flow	Flow of the code has changed
Identifiers names	Identifiers names have changed
Classes trees	The reversing tool is not able to find the connection.
Printed messages / encrypted strings	Encrypted strings were converted to series of number.
Pre-compilers	Extra pre-compilers were added to the code.

3) CAVAJ REVERSING TOOL, COMPILED REVERSED CODE ERROR TEST

NetBeans was not able to run the reversed code, as it contained many errors from the reversing tool. The result of the encrypted string was a series of numbers in an array. NetBeans was not able to read this series of numbers. The public class in the code had an error which was undefined.

To determine the strength of the proposed obfuscated technique, we calculate the total errors appeared during running the reversed code before and after obfuscation. This calculation will help to determine the ability of obfuscation technique to hide the code from reversing tools.

- Total errors of running reversed file before obfuscation: **0**

https://t.me/learningnets

TABLE 11. Reversed code output translation.

Output	Translation
import java .io.PrintStream	Extra library added that has no use to the original program
String a001001a String b0011000185 Int s0111v010m12 M01010010101000n0101088 Int d0011A4iA2viA5d160e Int m100100100i Int n101010r1 Int n01010102	Reversing tool was not able to read the variables after obfuscation
Public void m00111m	System keywords did not change to keep the system running
\0070D\276\313\255\306\256\264\306\311\313\276\314\274\u02EC\25	Encrypted string has been changes to a series of random numbers
for (int i$ = 0;i$<len$;i$++)	Logic has changed after reversing
a001001a	Binary code not readable by the reversing tool

- Total errors of running reversed file after obfuscation: **6**

The code that was generated from the reversing tool was not running as it was supposed to be, therefore it did not produce any output.

4) CAVAJ REVERSING TOOL, FLOW TEST

There was a parsing error during reversing. Some of the code structure was changed from the reversing tool. Furthermore, some variables names were undefined such as int **d0011A4iA2viA5d160e**.

The results of the revising tools, Tree classes have disappeared, run time error occur during running the source file. The reversing tool has changed the flow of the code and structure. Variables and classes names were still obfuscated, the reversing tool was not able to read them correctly.

Parsing the class file was possible by the reversing tool as well. Fig 10 demonstrates the output of reversed obfuscated code.

5) CAVAJ REVERSING TOOL, IDENTIFIERS NAMES TEST

The obfuscated identifiers have changed and remained encrypted. The reversing tool was able to read the mathematical equation which was used for the string encryption; however, it was partially discovered and was relocated in a different location in the code associated with a loop. Fig 11 demonstrates the errors of reversed code. Table 11 displays the translation of the output.

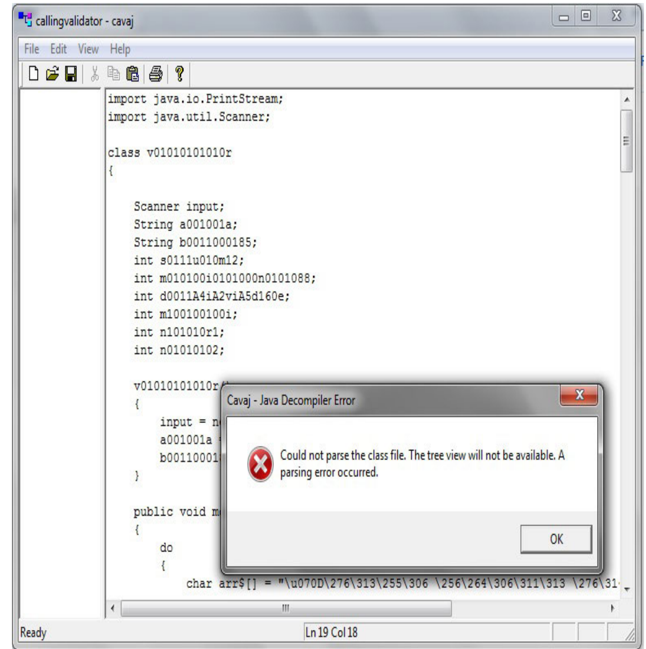


FIGURE 10. Errors of reversed code after compiling.

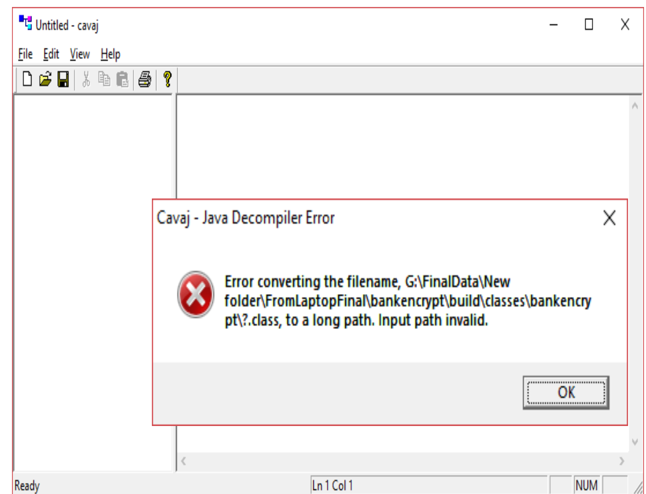


FIGURE 11. Errors of reversed code after compiling.

D. CAVAJ REVERSING TOOL, DE-CRYPT STRING TEST

The reversing tool was not able to parse the file. An error converting the file name. In this case, the string encryption was used heavily in the file and identifiers were renamed to junk. Then there was another layer of obfuscation added while compiling the file to machine language. The transformation process has made reversing difficult.

Fig 12 demonstrates the result of reversing obfuscated code.

To determine the strength of the proposed obfuscated technique, we calculate the total errors appeared during running the reversed code before and after obfuscation. This calculation will help to find out the ability of obfuscation technique to hide the code from reversing tools

https://t.me/learningnets

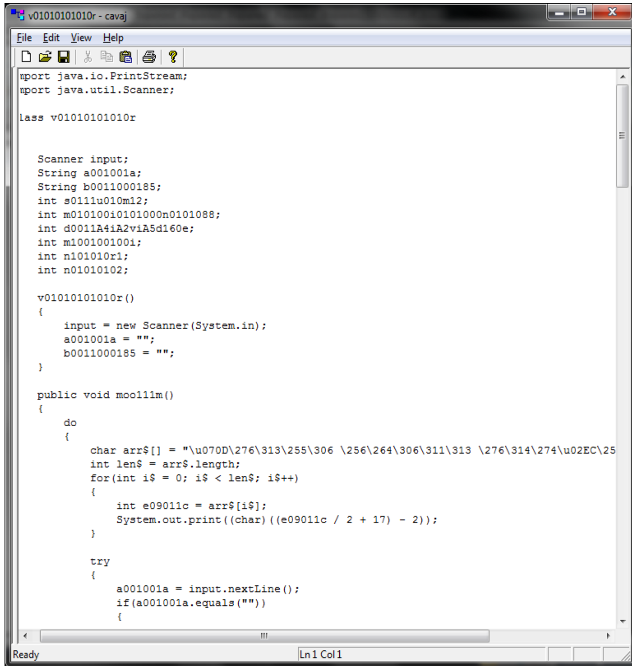


FIGURE 12. Errors of reversed code after compiling.

- Total errors of running reversed file before obfuscation: **0**
- Total errors of running reversed file after obfuscation: **1**

The code that was generated from the reversing tool was not running as it was supposed to be, therefore it did not produce any output.

Summary of CAVAJ Testing: According to the experiment conducted with CAVAJ reversing tool. The result supports the first and second objectives. The reversing tool was not able to read the texts in encrypted strings, chaos stream was generated after reversing.

The proposed hybrid obfuscation technique contains three elements of renaming that are string encryption and renaming that focuses on junk and Unicode. The three elements were mixed in the source file to create confusion while decompiling.

The ordinary obfuscation techniques usually apply one renaming technique. Having an ordinary renaming obfuscation does not protect the code from reversing tools. However, applying the hybrid obfuscation technique has proved to be better than ordinary obfuscation techniques.

After applying the hybrid obfuscation technique and running the CAVAJ reversing tool against it, it was obvious that the reversing tool was not able to read the encrypted strings, and have created a chaos stream as per the first objective of this research. The chaos stream that was created looks ***\0070D\276\ 313\255\306\256\264\306\ 311\3 13\276\ 314\274\u02EC.*** The reversing tool was not able to read the junk code also during reversing, a series of junk code was generated such as:

- **M01010010101000n0101088**
- **Int d0011A4iA2viA5d160e**

Applying hybrid obfuscation technique grants protections whether there was reverse engineering attempted or it has been stolen.

E. SECOND PRESENTATION OF TOOLS WILL BE JAD REVERSING TOOL

1. JAD Reversing Tool, (Output Correctness) Test: There were **7 errors** while reversing. Some of the code statements were discovered but without meaning. There was an invalid method declaration in reversing time. The reversing tool was not able to translate the file name and the obfuscated code.

Fig.13 demonstrates the output of reversed obfuscated code.

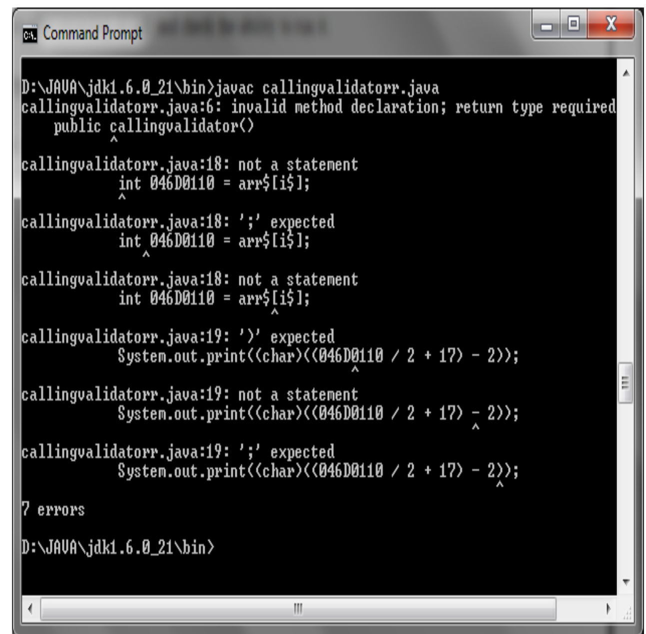


FIGURE 13. Errors of reversed code after compiling.

To determine the strength of the obfuscation technique, we calculate the total errors appeared during reversing before and after obfuscation

- Total errors of running reversed file before obfuscation: **0**
- Total errors of running reversed file after obfuscation: **7**

According to the results, it was clear that the reversing tool was not able to reveal the code **100%** without errors.

1) JAD REVERSING TOOL, COMPILED REVERSED CODE ERROR TEST

The reversing tool have generated **8 errors** during reversing process. Series of random numbers were generated from the encrypted strings in the source code. A series of junk code were generated.

To determine the strength of the obfuscation technique, we calculate the total errors appeared during reversing before and after obfuscation

- Total errors of running reversed file before obfuscation: **0**

https://t.me/learningnets

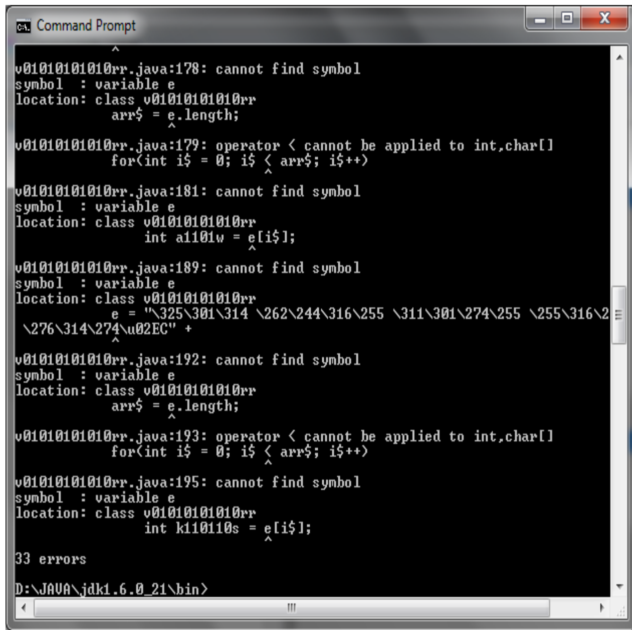


FIGURE 14. Errors of reversed obfuscated code after compiling.

- Total errors of running reversed file after obfuscation: **8**

Looking at the results it was clear that the reversing tool was not able to reveal the code **100%** without errors.

2) JAD REVERSING TOOL, METHODS AND CLASSES CORRECTNESS TEST

The reversing tool, there were **33 errors** while reversing. Some of the code statements were discovered but without meaning. After reversing, it was possible to save the file, but compiling was not possible.

The tool could not read the encrypted strings, it has generated a series of random numbers. Fig 14 demonstrates the output of reversed obfuscated code.

To determine the strength of the obfuscation technique, we calculate the total errors appeared during reversing before and after obfuscation

- Total errors of running reversed file before obfuscation:**0**
- Total errors of running reversed file after obfuscation: **33**

According to the results, it was clear that the reversing tool was not able to reveal the code **100%** without errors

1. JAD Reversing Tool, Identifiers Names Test: The identifier names were converted to numbers. Empty methods were added to the code. Classes trees have disappeared with some reference indicate to them. Fig 15 demonstrates the output of reversed obfuscated code. Table 13 displays the identifiers before and after reversing.

Summary of JAD Testing: As per the experiment, the result answers the second research objective as the reversing tool was not able to read the junk code as seen below:

- A461.F909();
- A461.F90A();
- ᛘ\u002E龜

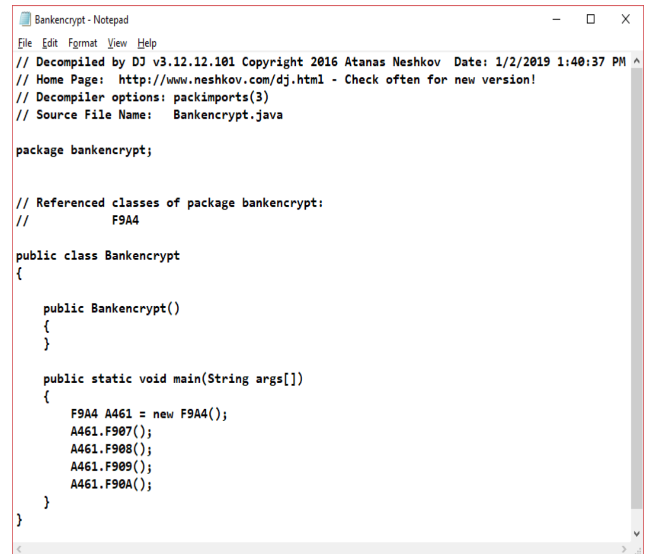


FIGURE 15. Result of reversed obfuscated code.

TABLE 12. Presentation of obfuscated and reversed identifiers.

Identifiers before and after reversing		
Obfuscated code	**Reversed code**	**Original code**

ᛘ\u002E龜\u0028 \u0029\u003B
ᛘ\u002E龜\u0028 \u0029\u003B
ᛘ\u002E契\u0028 \u0029\u003B
ᛘ\u002E金\u0028 \u0029\u003B
	A461.F907();
	A461.F908();
	A461.F909();
	A461.F90A();
		obj1.datain();
		obj1.amount();
		obj1.withdraw() ;
		obj1.balance();

The experiment has proven that applying the hybrid obfuscation technique is evidently better than applying ordinary obfuscation. According to the results from the experiment, the reversing tool was not able to read or determine the meaning of the code.

F. THIRD PRESENTATION OF TOOLS WILL BE DJ REVERSING TOOL

1) DJ REVERSING TOOL, (OUTPUT CORRECTNESS) TEST

DJ Reversing tool java is a tool that reverses the class file developed by java. This tool allows editing the code for other purposes, this tool is used to determine the ability to reverse java class file that contains a hybrid obfuscated technique. The test will determine if the tool is able to read the obfuscated code, and how much can the tool reveal. Fig.16 demonstrates the reversing result.

Main class was reversed by the tool. Evidently the reversing tool was not able to find the class files for reversing.

https://t.me/learningnets

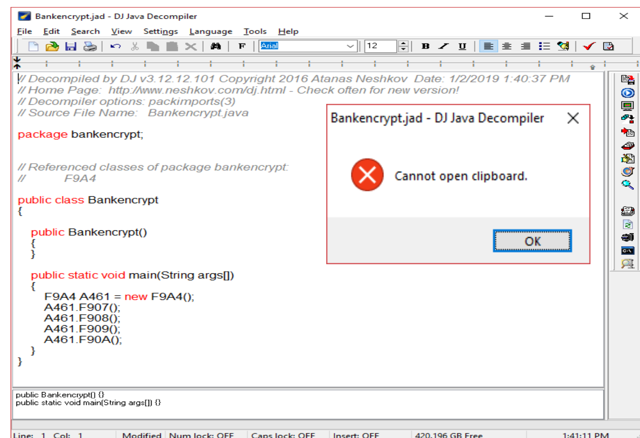


FIGURE 16. After reversing the class file.

TABLE 13. Presentation of objects.

Output Analysis	
Main Class file	Was not found
Variables	No variables have appeared
Classes	No classes have appeared

The tool has generated an error message that notifies that the file could not be opened and was not found. Table 14 displays the objects of the file.

To determine the strength of the proposed obfuscated technique, we calculate the total errors appeared during running the reversed code before and after obfuscation. This calculation will help to find out the ability of obfuscation technique to hide the code from reversing tools

- Total errors of running reversed file before obfuscation: **0**
- Total errors of running reversed file after obfuscation: **1**

The code that was generated from the reversing tool was not running as it was supposed to be, therefore it did not produce any output.

DJ Reversing Tool, Identifiers Names Test: The reversing tool was not able to read the identifiers in the source file. There was no analysis done on the code by the tool. An empty method was added to the code and reference to indicate missing class. Fig 17 demonstrates the output result of the reversing tool. Table 14 displays the changes happened to the identifiers after reversing.

Summary of DJ Testing: The results of the reversing tool answer to the first and second research objectives and third and fourth hypothesis. The reversing tool was not able to read the encrypted strings and junk code. It is however created un-known ciphers. The created cipher looks like following.

- A461.F907();
- A461.F908();
- ᛘ\u002E龜

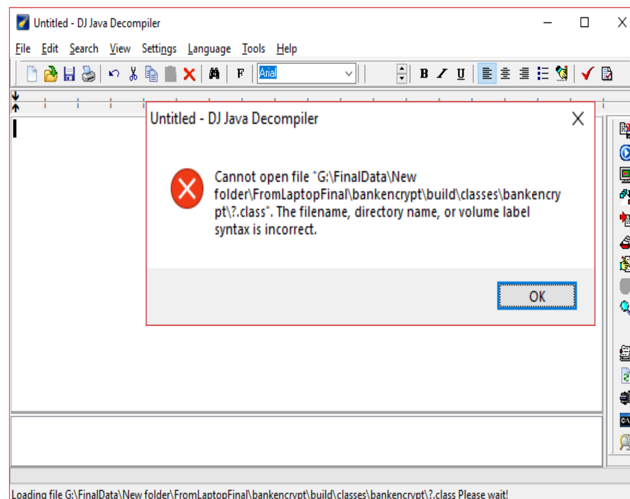


FIGURE 17. Result of reversed obfuscated code.

TABLE 14. Obfuscated Identifiers before and after reversing.

Identifiers before and after reversing		
Obfuscated code	**Reversed code**	**Original code**
ᛘ \u002E龜\u0028 \u0029\u003B	A461.F907();	obj1.datain();
ᛘ \u002E龜\u0028 \u0029\u003B	A461.F908();	obj1.amount();
ᛘ \u002E契\u0028 \u0029\u003B	A461.F909();	obj1.withdraw();
ᛘ \u002E金\u0028 \u0029\u003B	A461.F90A();	obj1.balance();

G. FOURTH PRESENTATION OF TOOLS WILL BE JD REVERSING TOOL

1) JD REVERSING TOOL, IDENTIFIERS NAMES TEST

DJ Reversing tool java is a tool that reverses the class. This tool allows editing the code for other purposes, it is used to determine the ability to reverse java class file that contains a hybrid obfuscated technique. The test will determine if the tool is able to read the obfuscated code, and how much can the tool read and discover. The reversing tool was able to read the code, however, was not able to read the encrypted strings. It has converted the strings to an unknown language. The variables were converted. A mathematical equation was revealed but with some encrypted string. Table 15 displays the translation of the reversed code.

To determine the strength of the reversing tool against the hybrid obfuscation technique, total number of lines of the reversed original file will be compared with total number of lines of obfuscated reversed file.

- Total LOC of original reversed file: **48**
- Total LOC of obfuscated reversed file: **49**
- Difference between original file and reversed file: $\mathbf{48 - 49 = 1}$

https://t.me/learningnets

TABLE 15. Translation of obfuscated reversed code.

Identifiers names		
	After reversing code	**Original code**
Variables	String ?; String ?; int ?; double ?; double ?; double ?; double ?; double ? = 2000.0D;	String name, typeac; int accno; double total, amount, amountout, running, balance=2000;
	After revers code	**Original code**
Classes	public class?	public class bank
	After reversing code	**Obfuscated code**

Identifiers	for(int ?011? :
	"¤??ÀÌ¾ËË?¬"
	.toCharArray()){
	System.out.print((
	char)(?011?/2+17
	-2));}System.out
	.print("\n"); this.?
	=
	this.?.nextLine();}
		\u0066\u006F\u007
		2\u0028\u0069\u00
		6E\u0074ıĸ011邏
		\u003A" ◻ÌÈËÀ¼¬
		Æ ¾ ¤¼¬"\u002E\
		u0074\u006F\u004
		3\u0 068\u0061\u0
		072\u 0041\u0072\
		u0072\u0061\u0079\
		u0028 \u0029

The reversing tool was able to discover the code despite the complication of it. It however has changed most of the code and could not read some of symbols in the code. To determine the strength of the proposed obfuscated technique, we calculate the total errors that appeared during running the reversed code before and after obfuscation. This calculation will help to find out the ability of obfuscation technique to hide the code from reversing tools:

- Total errors of running reversed file before obfuscation: **0**
- Total errors of running reversed file after obfuscation: **1**

The code that was generated from the reversing tool was not running as it was supposed to be, therefore it did not produce any output. The generated code from the reversed code is copied into NetBeans for further testing. After running the code, an error appeared, and there was no results or output. Fig.18 demonstrates the error message that appeared after compiling the reversed code.

Summary of JD Testing: The result of the experiment conducted with the JD reversing tool proves that the first and second research objectives are successfully met. The experiment of the tool proves that the research questions are met where it was possible to merge the three approaches proposed in the research as discussed in the first chapter.

```
package bankencrypt; import java.io.PrintStream; import
java.util.Scanner; public class ? { Scanner ? = new
Scanner(System.in); Scanner ? = new Scanner(System.in); String
?; String ?; int ?; double ?; double ?; double ?;
double?;double?=2000.0D;void?(){for(int?011?:"?ÌËËÀ¼¬Æ¾¤
¼¬".toChar Array()){System.out.print ((char)(?011? / 2 + 17 -
2));}System.out.print("\n"); this.? = this.?.nextLine(); for (int
?011? : "¤??ÀÌ¾Ë ¾Ì¼§¾Æ".toCharArray()){ System.out.print
((char)(?011? / 2 + 17 - 2)); }
System.out.print("\n");this.?=this.?.nextInt();
for(int?011?:"¤??ÀÌ¾ËËÕ?¬".toCharArray()){System.out.print((
char)(?011?/2+17-2));} System.out.print("\n"); this.? =
this.?.nextLine();} void ?()
{for(int?011?:"¤¼ÀÌ¾Ë?¬?ÀÈµË".toCharArray()){System.out.pr
int((char) (?011? / 2 + 17 - 2));} System.out.print("\n"); this.? =
this.?.nextDouble(); this.?
+=this.?;for(int?011?:"?ÌÆÆ¬¾Ë?¤?¤¾?¬".toCharArray()){Syst
em.out.print((char)(?011?/2+17-2));} System.out.print("\n");
System.out.println(this.?);}void
?(){for(int?011?:"ÐµË??Æ¤Ð".toCharArray()){System.out.print(
(char) (?011?/2+17-2));}System.out.print ("\n") ;this.? =
this.?.nextDouble(); }void? (){this.?=(this.?-
this.?);for(int?011?:"?¤?¤¾?¬".toCharArray()){System.out.
print((char)(?011?/2+17-2));}
System.out.print("\n");System.out.println (this.?);}}
```

FIGURE 18. Result of reversed obfuscated code.

According to the research hypothesis, the experiments have proven that the reversing tools were not able to translate or de-compile the obfuscated code due to its complication and the great merge of the three approaches of the string encryption and renaming techniques.

All reversing tools have generated errors during de-compiling due to the hard encryption used in the source code. The generated code from the reversing tools has generated errors as well during compiling.

Summary of Experiment: The Hybrid Obfuscation Technique was effective to protect the code. The reversing tools were not able to read and translate the encrypted strings. Renaming to junk in the obfuscation technique was effective as the reversing tool has converted the junk to a series of random numbers and symbols.

The reversing tool was able to read the system keywords only. Furthermore, the reversing tool has added methods, pre-processors while parsing the file. The reversing tool was not able to analyze the obfuscated code to get the appropriate output.

This means that the Hybrid Obfuscation Technique is effective to protect the source file from prohibited reverse engineering. The third objective of this research was successfully met, according to the experimentation, a series of junk and chaos was created after reversing the obfuscated code. The extreme chaos was generated due to the merge of string encryption and renaming approaches in one source file which has led to confusion while reversing as the reversing tool was not able to translate or read or analyze the code.

To summarize the results of the experiments that were conducted before and after obfuscation, we calculate the LOC of the original file before and after reversing, we calculate total errors appeared during running the reversed file before and after obfuscation, then we find the difference to determine the

https://t.me/learningnets

TABLE 16. Calculation summary of first and second experiment.

	Before obfuscation		After obfuscation
Reversing tool	LOC original file	LOC reversed file	LOC reversed file
CAVAJ	44	48	20
JAD	56	60	21
DJ	117	148	16
	421	477	15
JD	48	49	49

TABLE 17. Errors summary of first phase experiment cases.

Reversing tool	Testing element	Reversed file before obfuscation	Reversed file after obfuscati on
CAVAJ	Compiled reversed code error test	0	6
	De-Crypt String test	0	1
JAD	Output correctness	0	7
	Compiled reversed code error test	0	8
	Methods and classes correctness test	0	33
DJ	Output correctness test	0	1
JD	Identifiers names test	0	1

strength. Table 16 displays the summary of the experiments before and after obfuscation.

Based on the results of the reversing tools, they were not able to discover fully functioning code, in all cases the reversing tools have generated a series of chaos and random numbers and symbols while attempting to translate the obfuscated code.

The code that was generated from the reversing tools did not provide an output, there was always an error while trying to compile the obfuscated code after reversing. Table 17 displays the summary of errors that occurred for the four tested cases.

The code that was generated by the reversing tools did not provide any output after it has been obfuscated. Errors always occur while running the reversed code after obfuscation. The minimum errors occur while running code was one error. Even though some of the cases had one error only, yet there was no output generated.

The experiments have supported the research hypothesis, where protection is highly needed to protect the code and more specifically hybrid obfuscation technique is more protective than standard obfuscation techniques. It was proved that not having protection or applying standard obfuscation technique decreases the security of the code, while applying a hybrid obfuscation technique have successfully protected the 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 framework for automation of the presented technique and to provide as a plug-in to support developers to customize the method of obfuscation. Also, 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 to be 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.

Merging of different renaming techniques and mathematical equations for encryption plus the extra layer added from the compiler during compiling have created a huge amount of junk code while reversing as the reversing tool was not able to determine or trace or analyze the obfuscated code.

ACKNOWLEDGMENT

Asma'a Mahfoud Hezam Al-Hakimi would like to thank and acknowledge her Supervisor Professor Dr. Abu Bakar Md Sultan for her guidance, support, and understanding during this research. She would also like to extend her appreciation

https://t.me/learningnets

to my committee members, Professor Dr. Abdul Azim Abdul Ghani, Dr. Norhayati Binti Mohd Ali, and Dr. Novia Indriaty Admodisastro for their support and contribution.

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] P. Samuelson and S. Scotchmer, "The law and economics of reverse engineering," *Yale Law J.*, vol. 111, no. 7, pp. 1575–1663, 2002.

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

[4] M. Batchelder and L. Hendren, "Obfuscating Java: The most pain for the least gain," in *Compiler Construction* (Lecture Notes in Computer Science), vol. 4420. Berlin, Germany: Springer-Verlag, 2007, pp. 96–110.

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

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

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

[8] 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. Netw. (WiSPNET)*, Mar. 2017, pp. 828–832.

[9] 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.

[10] 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 tansformations," in *Proc. 7th Int. Conf. Comput. Sci. Edu. (ICCSE)*, Jul. 2012, pp. 1093–1097.

[11] 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.

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.

[17] D. Pizzolotto and M. Ceccato, "[Research paper] obfuscating java programs 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.

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

[19] S. I. Bani Baker and A. H. Al-Hamami, "Novel algorithm in symmetric encryption (NASE): Based on feistel cipher," in *Proc. Int. Conf. New Trends Comput. Sci. (ICTCS)*, Oct. 2017, pp. 191–196.

[20] Z. Y. Wang and W. M. Wu, "Technique of Javascript code obfuscation based on control flow tansformations," *Appl. Mech. Mater.*, vols. 519–520, pp. 389–392, Feb. 2014.

[21] H. Badier, J.-C.-L. Lann, P. Coussy, and G. Gogniat, "Transient key-based obfuscation for HLS in an untrusted cloud environment," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2019, pp. 1118–1123.

[22] M. Maskur, Z. Sari, and A. S. Miftakh, "Implementation of obfuscation technique on PHP source code," in *Proc. 5th Int. Conf. Electr. Eng., Comput. Sci. Informat. (EECSI)*, Oct. 2018, pp. 738–742.

[23] M.-J. Kim, J.-Y. Lee, H.-Y. Chang, S. Cho, Y. Park, M. Park, and P. A. Wilsey, "Design and performance evaluation of binary code packing for protecting embedded software against reverse engineering," in *Proc. 13th IEEE Int. Symp. Object/Compon./Service-Oriented Real-Time Distrib. Comput.*, May 2010, pp. 80–86.

[24] X. Guangli and C. Zheng, "The code obfuscation technology based on class combination," in *Proc. 9th Int. Symp. Distrib. Comput. Appl. Bus., Eng. Sci.*, Aug. 2010, pp. 479–483.

[25] P. Leahy, *What is Unicode*. New York, NY, USA: ThoughtCo, 2017, p. 1.

[26] X. Zhou and J. Xie, "Evaluating obfuscation performance of novel algorithm-to-architecture mapping techniques in systolic-array-based circuits," in *Proc. Asian Hardw. Oriented Secur. Trust Symp. (AsianHOST)*, Oct. 2017, pp. 127–132.

ASMA'A MAHFOUD HEZAM AL-HAKIMI was born in Egypt. She received the Diploma degree in computer programming from the University of Science and Technology Sanaa Yemen, in 2006, the bachelor's degree in computer studies from Northumbria Newcastle University, U.K., in 2008, the master's degree in software engineering from Staffordshire University, U.K., in 2011, and the Ph.D. degree in software engineering from Universiti Putra Malaysia. She started working as a Graphic Designer in the media line, then in networking field. She has worked for five years as a Lecturer in Asia Pacific (APU), Malaysia.

ABU BAKAR MD SULTAN received the bachelor's degree in computer science from Universiti Kebangsaan Malaysia, in 1993, and the master's degree in software engineering and the Ph.D. degree in artificial intelligence from Universiti Putra Malaysia (UPM). He is currently a Professor with the Faculty of Computer Science and Information Technology, UPM. His research interests include optimization and search-based software engineering (SBSE).

ABDUL AZIM ABDUL GHANI (Member, IEEE) received the B.Sc. degree in mathematics/computer science from Indiana State University, the M.Sc. degree in computer science from the University of Miami, and the Ph.D. degree in software engineering from the University of Strathclyde. He is currently a Professor with the Department of Software Engineering and Information System, Universiti Putra Malaysia. His research interests include software measurements, software testing, and software quality.

NORHAYATI MOHD ALI is currently an Associate Professor with the Department of Software Engineering and Information System, Universiti Putra Malaysia. Her main research interest includes software engineering.

NOVIA INDRIATY ADMODISASTRO is currently an Associate Professor with the Department of Software Engineering and Information System, Universiti Putra Malaysia. Her main research interests include software engineering, service-engineering, and human–computer interaction.

https://t.me/learningnets

View publication stats