ENHANCING OBFUSCATION TECHNIQUE FOR PROTECTING SOURCE CODE AGAINST SOFTWARE REVERSE ENGINEERING

ASMA MAHFOUDH

FSKTM 2020 6

ENHANCING OBFUSCATION TECHNIQUE FOR PROTECTING SOURCE CODE AGAINST SOFTWARE REVERSE ENGINEERING

By

ASMA MAHFOUDH

Thesis Submitted to the School of Graduate Studies, Universiti Putra Malaysia, in Fulfilment of the Requirements for the Degree of Doctor of Philosophy

December 2019

DEDICATION

This thesis is dedicated to my father MAHFOUDH AL-HAKIMI and my son YUSUF SULTAN

Abstract of thesis presented to the Senate of Universiti Putra Malaysia in fulfilment of the requirement for the degree of Doctor of Philosophy

ENHANCING OBFUSCATION TECHNIQUE FOR PROTECTING SOURCE CODE AGAINST SOFTWARE REVERSE ENGINEERING

By

ASMA MAHFOUDH

December 2019

Chairman : Professor Abu Bakar bin Md Sultan, PhD
Faculty : Computer Science and Information Technology

Obfuscation (Obfu) is a practice to make the programming code complicated to protect the Intellectual Property (IP) and prevent prohibited software Reverse Engineering (RE). Obfuscation involves transforming potentially revealing data, renaming useful classes and variables (identifiers) names to meaningless labels or adding unused or meaningless code to an application binary. Obfuscation is used to convert source code into a program that works the same way but is much harder to read and understand. Obfuscation techniques allow the programmer to customize which part of the code to be obfuscated.

Recently, obfuscation techniques were mostly used to secure the source code; however, none of the current obfuscation techniques satisfy all obfuscation effectiveness criteria to resist the attack of Reverse Engineering. Therefore, IT industry loses tens of billions of dollars annually due to security attacks such as reverse engineering. The obvious amount of lost money of victims has led to many court cases where victim and theft claims the ownership of the program and the winner is who has a good lawyer. Many programming languages are used for programming; Java programming language is known to be most common due to its features, the use of this popular language 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 such intermediate formats to extract sensitive information such as proprietary algorithms present in the software. Hence, there is a need for development of techniques and schemes to obfuscate sensitive parts of software to protect it from reverse engineering attacks.

In this research, we have proposed a new Hybrid Obfuscation Technique to prevent prohibited Reverse Engineering. The proposed 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 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.

Empirical evaluation was conducted to evaluate the proposed Hybrid Obfuscation Technique. It consists of experiment and interview. The experiment contains two phases; first phase was conducted against java applications that do not use any protection to determine the ability of reversing tools to read the compiled code. Second phase was conducted against the proposed technique to evaluate the effectiveness of it. Interview was conducted to get an overview of programming experts towards using Hybrid Obfuscation Technique to prevent prohibited Reverse Engineering. 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 proposed technique can be enhanced in the future to protect games applications and mobile applications that are developed by java; it can improve the software development industry. The proposed technique can be used to support many languages such as Arabic, English, Chinese and so on. There is also a need to develop a tool that contains the three approaches where the developer can customize the protection of the source code.

Abstrak tesis yang dikemukakan kepada Senat Universiti Putra Malaysia sebagai memenuhi keperluan untuk ijazah Doktor Falsafah

MENINGKATKAN TEKNIK OBFUSCASI UNTUK MELINDUNGI KOD SUMBER DARI KEJURUTERAAN TERBAIK PERISIAN

Oleh

ASMA MAHFOUDH

Disember 2019

Pengerusi : Profesor Abu Bakar bin Md Sultan, PhD
Fakulti : Sains Komputer dan Teknologi Maklumat

Obfuscation (Obfu) adalah amalan untuk menjadikan kod pengaturcaraan menjadi rumit untuk melindungi Harta Intelek (IP) dan mencegah perisian terlarang Reverse Engineering (RE). Kekaburan melibatkan mengubah data yang berpotensi mendedahkan, menamakan semula nama kelas dan pemboleh ubah yang berguna (pengecam) menjadi label yang tidak bermakna atau menambahkan kod yang tidak digunakan atau tidak bermakna pada perduaan aplikasi. Obfuscation digunakan untuk menukar kod sumber menjadi program yang berfungsi dengan cara yang sama tetapi jauh lebih sukar untuk dibaca dan difahami. Teknik penyamaran membolehkan pengaturcara menyesuaikan bahagian kod mana yang akan dikaburkan.

Baru-baru ini, teknik penyamaran kebanyakan digunakan untuk mendapatkan kod sumber; namun, tidak ada teknik penyamaran semasa yang memenuhi semua kriteria keberkesanan penyamaran untuk menentang serangan Kejuruteraan Balik. Oleh itu, industri IT kehilangan puluhan bilion dolar setiap tahun kerana serangan keselamatan seperti teknik terbalik. Jumlah wang mangsa yang hilang telah menyebabkan banyak kes mahkamah di mana mangsa dan kecurian mendakwa pemilikan program dan pemenangnya adalah yang mempunyai pengacara yang baik. Banyak bahasa pengaturcaraan digunakan untuk pengaturcaraan; Bahasa pengaturcaraan Java dikenal paling umum kerana ciri-cirinya, penggunaan bahasa popular ini meningkatkan kemampuan penyerang untuk mencuri harta intelek (IP), kerana program sumber diterjemahkan ke format perantaraan yang mengekalkan sebahagian besar maklumat seperti yang bermakna nama pemboleh ubah yang terdapat dalam kod sumber. Penyerang dapat dengan mudah membina semula kod sumber dari format perantaraan seperti itu untuk mengekstrak maklumat sensitif seperti algoritma proprietari yang terdapat dalam perisian. Oleh itu, terdapat keperluan untuk pengembangan teknik dan skema untuk menyamarkan bahagian perisian yang sensitif untuk melindunginya dari serangan kejuruteraan terbalik.

Dalam penyelidikan ini, kami telah mencadangkan Teknik Penyamaran Hibrid baru untuk mencegah Kejuruteraan Terbalik yang dilarang. Teknik yang dicadangkan mengandungi tiga pendekatan; pendekatan pertama adalah penyulitan rentetan. Penyulitan tali adalah mengenai menambahkan persamaan matematik dengan tatasusunan dan gelung pada rentetan dalam kod untuk menyembunyikan maksudnya. Pendekatan kedua adalah menamakan semula kata kunci sistem kepada Unicode untuk meningkatkan kesukaran dan kerumitan kod. Pendekatan ketiga adalah mengubah pengenal kepada kod sampah untuk menyembunyikan makna dan meningkatkan kerumitan kod.

Penilaian empirikal dilakukan untuk menilai Teknik Penyamaran Hibrid yang dicadangkan. Ia terdiri daripada eksperimen dan temu bual. Eksperimen ini mengandungi dua fasa; fasa pertama dijalankan terhadap aplikasi java yang tidak menggunakan perlindungan untuk menentukan kemampuan alat membalikkan membaca kod yang disusun. Fasa kedua dijalankan terhadap teknik yang dicadangkan untuk menilai keberkesanannya. Temu bual dilakukan untuk mendapatkan gambaran keseluruhan pakar pengaturcaraan terhadap penggunaan Teknik Pengabaian Hibrid untuk mencegah Kejuruteraan Terbalik yang dilarang. Percubaan teknik penyamaran hibrid adalah untuk menguji ketepatan output, sintaks, kesalahan kod terbalik, ujian aliran, ujian nama pengenal, kaedah dan ujian ketepatan kelas. Dengan parameter ini adalah mungkin untuk menentukan kemampuan teknik yang dicadangkan untuk mempertahankan serangan.

Teknik yang dicadangkan dapat ditingkatkan pada masa akan datang untuk melindungi aplikasi permainan dan aplikasi mudah alih yang dikembangkan oleh java; ia dapat meningkatkan industri pengembangan perisian. Terdapat juga keperluan untuk mengembangkan alat yang berisi tiga pendekatan di mana pembangun dapat menyesuaikan perlindungan kod sumber. Teknik yang dicadangkan boleh digunakan untuk menyokong banyak bahasa seperti Arab, Inggeris, Cina dan sebagainya. Terdapat juga keperluan untuk mengembangkan alat yang berisi tiga pendekatan di mana pembangun dapat menyesuaikan perlindungan kod sumber.

ACKNOWLEDGEMENTS

First of all, I would like to thank Allah Subhanahu Wa Ta'ala for granting me the strength, patience, guidance, and ability to achieve this study.

I would like to specially thank and acknowledge my Supervisor Professor Dr. Abu Bakar Md Sultan for his guidance, support, and understanding throughout my study. I would like to extend my appreciation to my committee members, Professor Dr. Abdul Azim Abdul Ghani, Dr. Norhayati Binti Mohd Ali, and Dr. Novia Indriaty Admodisastro for their support, guidance, and contribution.

This study would not have been completed without the support, prayers, encouragement and the helping hands from my dearly parents specially my father Mahfoud Al-Hakimi, who have always believed me and supporting me emotionally and financially, who always been in my side whenever I needed him.

I wish to extend my gratitude to my friends, siblings, and lab mates, who always listened to my objections and study issues and never been stingy in providing advises.

I certify that a Thesis Examination Committee has met on 16 December 2019 to conduct the final examination of Asma Mahfoudh on her thesis entitled "Enhancing Obfuscation Technique for Protecting Source Code against Software Reverse Engineering" in accordance with the Universities and University Colleges Act 1971 and the Constitution of the Universiti Putra Malaysia [P.U.(A) 106] 15 March 1998. The Committee recommends that the student be awarded the Doctor of Philosophy

Members of the Thesis Examination Committee were as follows:

Hazura bt. Zulzalil, PhD
Associate Professor
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
(Chairman)

Rodziah binti Atan, PhD
Associate Professor
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
(Internal Examiner)

Wan Nurhayati binti Wan Ab. Rahman, PhD
Associate Professor
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
(Internal Examiner)

Volker Gruhn, PhD
Professor
Faculty of Engineering
University Duisburg-Essen
Germany
(External Examiner)

ZURIATI AHMAD ZUKARNAIN, PhD
Professor Ts. and Deputy Dean
School of Graduate Studies
Universiti Putra Malaysia

Date: 03 September 2020

This thesis was submitted to the Senate of the Universiti Putra Malaysia and has been accepted as fulfilment of the requirement for the degree of Doctor of Philosophy. The members of the Supervisory Committee were as follows:

Abu Bakar bin Md Sultan, PhD
Professor
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
(Chairman)

Norhayati binti Mohd Ali, PhD
Associate Professor
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
(Member)

Novia Indriaty Admodisastro, PhD
Associate Professor
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
(Member)

Abdul Azim B Abd Ghani, PhD
Professor
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
(Member)

ZALILAH MOHD SHARIFF, PhD
Professor and Dean
School of Graduate Studies
Universiti Putra Malaysia

Date: 10 September 2020

Declaration by graduate student

I hereby confirm that:

- this thesis is my original work;
- quotations, illustrations and citations have been duly referenced;
- this thesis has not been submitted previously or concurrently for any other degree at any institutions;
- intellectual property from the thesis and copyright of thesis are fully-owned by Universiti Putra Malaysia, as according to the Universiti Putra Malaysia (Research) Rules 2012;
- written permission must be obtained from supervisor and the office of Deputy Vice-Chancellor (Research and innovation) before thesis is published (in the form of written, printed or in electronic form) including books, journals, modules, proceedings, popular writings, seminar papers, manuscripts, posters, reports, lecture notes, learning modules or any other materials as stated in the Universiti Putra Malaysia (Research) Rules 2012;
- there is no plagiarism or data falsification/fabrication in the thesis, and scholarly integrity is upheld as according to the Universiti Putra Malaysia (Graduate Studies) Rules 2003 (Revision 2012-2013) and the Universiti Putra Malaysia (Research) Rules 2012. The thesis has undergone plagiarism detection software

Signature: _____ Date: _____

Name and Matric No: Asma Mahfoud, GS33200

Declaration by Members of Supervisory Committee

This is to confirm that:

- the research conducted and the writing of this thesis was under our supervision;
- supervision responsibilities as stated in the Universiti Putra Malaysia (Graduate Studies) Rules 2003 (Revision 2012-2013) were adhered to.

Signature:	
Name of Chairman of Supervisory Committee:	Professor Dr. Abu Bakar bin Md Sultan
Signature:	
Name of Member of Supervisory Committee:	Associate Professor Dr. Norhayati binti Mohd Al
Signature:	
Name of Member of Supervisory Committee:	Associate Professor Dr. Novia Indriaty Admodisastro
Signature:	
Name of Member of Supervisory Committee:	Professor Dr. Abdul Azim B Abd Ghani

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

RE	Reverse Engineering
CSS	Content Scramble System
IDA	Interactive Disassembler
NOP	No-Operation
DMCA	Digital Millennium Copyright Act
FLIRT	Fast Library Identification and Recognition Technology
OTP	One-Time Password
BCO	Byte Code Obfuscation
API	Application Programming Interface
ART	Anti-Reverse Engineering
OOP	Object Oriented Programming
JDK	Java Development Kit
JRE	Java Runtime Environment
JVM	Java Virtual Machine
NoL	Number of Lines
HOTech	Hybrid Obfuscation Technique
IP	Intellectual property
Obfu	Obfuscation
DMCA	Digital Millennium Copyright Act
TC	Trusted Computing
DO	Data Obfuscation
NeNDS	Nearest Neighbor Data Substitution
EHR	Electronic Health Records

CFG	Control Flow Graph
JD	Java De-compiler
GUI	Graphical User Interface
JBVD	Java Bytecode Viewer and De-compiler
DVD CCA	DVD Copy Control Association
CSS	Content Scramble System
OTP	One-Time Password
LOC	Lines of Codes

CHAPTER 1

INTRODUCTION

1.1 Background

Anti-Reverse Engineering is a collection of algorithms, techniques and mechanisms that help the developer to harden the code in the source file against prohibited reverse engineering (Winograd, Salmani, Mahmoodi, & Homayoun, 2016). With anti-reverse engineering, it is possible to use different techniques and algorithms that complicate and harden the code, most of these techniques are used as countermeasure against prohibited reverse engineering(Rugaber & Stirewalt, 2014). There are different strategies to defend against attacks, and to protect the working of software, such as both legal and technical countermeasures. Copyright and patent are two main approaches to protect software against unlawful copying and stealing of algorithms. Even though copyright protection defends against illegal copying, it does not help in protecting the idea or the implemented algorithms. Software patents help to protect the computer programs inventions including the idea and the algorithm; however, they do not provide solid protection as they are not always enforceable. The major drawback of such patents is the cost(Bergström & Åhlfeldt, 2016). There are usually very expensive to enforce, and therefore unaffordable for small companies. According to the US Digital Millennium Copyright Act (DMCA) and EU Computer Programs Directive legislations, reverse engineering is allowed for the purpose of interoperability between computer programs, if the programs are obtained lawfully(Mohsen & Pinto, 2008). Hence, it is very difficult under these regulations to prevent reverse engineering for the understanding of the inner working of software. Because of these shortcomings, legal protection mechanisms, in most cases, have a small impact on foiling malicious attacks. The software industry has proposed many technical measures to the problem of prohibited reverse engineering attacks(Ceccato, Capiluppi, Falcarin, & Boldyreff, 2015). The processes of these measures are categorized to hardware protection and software protection. Hardware protection techniques leverage the hardware devices capabilities to provide protection, such as secure coprocessors, Trusted Computing (TC), tamper resistance, and smart cards, where secure computation is carried inside the protected hardware despite deployment in a hostile computing environment. However, hardware protection techniques do not provide a complete solution to the malicious attacks, and their logistic challenges such as the difficulty of upgrading hardware usually create difficulties in adapting them to computing infrastructures(Jang et al., 2018). For example, if the hardware protection technique gets compromised by an attacker, it would be very difficult to provide a quick response to patch and fix this problem. It would require a full upgrade and replacement cycle to get the device secure again(Mohsen, 2016).

Software protection techniques provide security by preventing reverse engineering(Sosonkin, Naumovich, & Memon, 2003). Software defence approaches are more flexible as they are fewer platforms dependent, and cheaper, than their hardware counterpart(Kulkarni, 2014). There are different forms of software protection, for example encryption and authentication, these methods help to secure software and

sensitive data, and decrypt the software code on the fly during the execution process. Unfortunately, since the execution requires decryption, the clear (decrypted) code is revealed in the memory during execution, and the attacker can dump the memory and construct the code. Therefore, encryption and decryption have to be conducted by trusted hardware devices, and thus it suffers from the same hardware protection drawbacks(Rugaber & Stirewalt, 2014). One of the defence methodologies that can be effectively used in this case, which is the subject of this research, is code obfuscation. The purpose of code obfuscation is to make the code difficult to read and understand, and hard to analyse by attackers, while preserving the intended functionality of the original program. The basic premise being that if the attacker cannot understand the outcome of the reverse engineering, then it is virtually impossible to usefully alter the reversed engineered code(Hausknecht & Gruicic, 2017). Among the various techniques available for protecting code from different attacks, code obfuscation is one of the most popular alternatives, for preventing from reverse engineering. So, code obfuscation is a largely adopted solution, and many different obfuscation approaches has been proposed(Ceccato et al., 2008). This is also a type of software protection against unauthorized reverse-engineering (Viticchie et al., 2016). For more than a decade code obfuscation is highlighted as the most common anti-reverse engineering and most effective in the software sector. Obfuscation takes many forms; it all depends on the developer to decide which part of the code to protect and what obfuscation method to use. There are several categories of code obfuscation; Layout obfuscation, Data obfuscation, Control obfuscation, and other categories (Dalai, Das, & Jena, 2018). Fig.1.1 Illustrates the types of obfuscation.

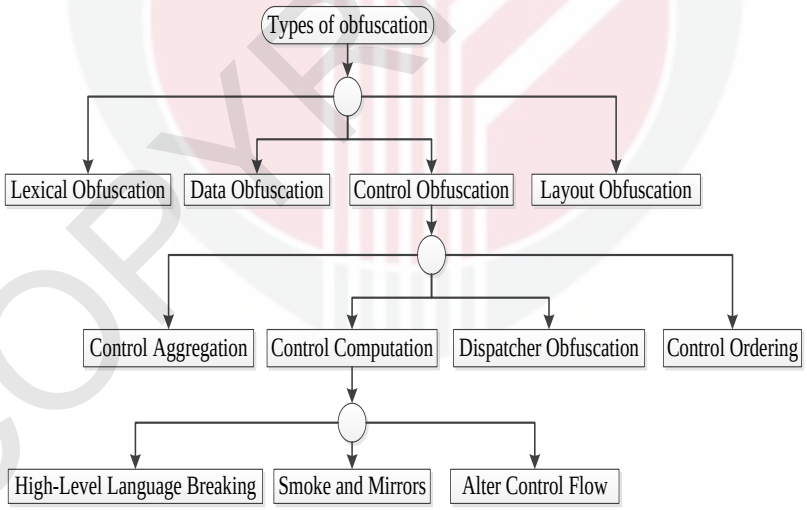

Figure 1.1 : Types of Obfuscation Techniques

Layout obfuscation is the process to hide the source code meaning when this code is disclosed to a third party. This technique is meant to rename the program identifiers and remove the comments from the source code. Source code obfuscation contains renaming

technique. This technique takes several approaches, such as identifiers renaming obfuscation, and string encryption (Borello & Mé, 2008).

Renaming obfuscation technique is known to be effective technique against prohibited reverse engineering. This technique is known to mislead the reverser, it changes the meaning of the code, therefore the reverser will not be able to perform analysis on the code (Tang, Chen, Fang, & Chen, 2009). Identifiers renaming obfuscation is the process to replace the identifiers with meaningless names to unknown language or symbols as new identifiers(Luo, Jiang, & Zeng, 2006). Usually, the identifiers have meaning for a better recognition of the source code structures like classes, methods, variables and so forth. Once an identifier is renamed, it is mandatory to provide consistency across the entire application through replacements of the old names by the new identifiers (Kang, Poosankam, & Yin, 2007).

Programming languages that allow the method overloading, method name obfuscation is made by the same identifier string for the methods having different signatures(Sosonkin et al., 2003). Junk renaming obfuscation is one of the approaches to rename the identifiers to some junk code that is readable by the compiler and produces sufficient output and in the same time not readable by the reverser (Popa, 2014a).

String encryption is another approach of the renaming obfuscation technique; this approach contains mathematical equation in array to generate chaos stream. The compiler will be able to read the code after transformation, but the reverser does not know how to read these symbols (Gong, Luo, Xie, Liu, & Lu, 2016). String encryption is well known to use chaos stream to transform the text in the source file into an unknown symbol in which the reverser is not able to read or understand. One of the common techniques for chaos stream is Cipher technique. this technique applies mathematical equation with arrays and loops to encrypt the strings in the source file (Wagner, 2003).

Data Obfuscation (DO) is a form of data masking where data is purposely scrambled to prevent unauthorized access to sensitive materials. This form of encryption results in confusing data.

Data obfuscation techniques are used to prevent the intrusion of private and sensitive online data, such as Electronic Health Records (EHR). However, issues have stemmed from an inability to prevent attacks. Additionally, there is not a set of standards for (DO) technique. Because of these challenges, researchers have proposed a more robust (DO) technique that is known as Nearest Neighbor Data Substitution (NeNDS), which is favored because of its privacy protection features and ability to sustain data clusters. The same researchers continue to prove that reverse engineering is easily accomplished with geometric transformations related to cluster preservations(C. & M., 2018).

1.2 Problem Statement

Intellectual property theft is one of the most challenging problems of today's technological era. According to Business software alliance global software piracy rate went noticeably high which lead to 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 various reverse engineering tools, software reversers are able to reveal the security algorithms to reveal the original code from the source file (ul Iman & Ishaq, 2010).

IT industry loses tens of billions of dollars annually due to security attacks such as reverse engineering. Code obfuscation techniques counter 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 such intermediate formats to extract sensitive information such as proprietary algorithms present in the software. Hence, there is a need for development of techniques and schemes to obfuscate sensitive parts of software to protect it from reverse engineering attacks(Gomes, 2014).

Software development industries spend billions of dollars annually for preventing from security attacks such as reverse engineering. Every organization is having its own intellectual property and it's 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 to protect the intellectual property, legally or technically(Batchelder & Hendren, 2007). Legally means getting copyrights or signing legal contracts against creating duplicates. And technically means the owners of the software will give the solution for protection with that 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 (C. Kumar & Bhaskari, 2015).

Obfuscation is known to be the most common and effective technique to prevent prohibited Reverse Engineering. However, none of the current obfuscation technique meets and satisfies all the obfuscation effectiveness criteria to resistance the Reverse Engineering (Popa, 2014a). None of the current code obfuscation techniques satisfy all the obfuscation effectiveness criteria such as resistance to reverse engineering attacks and state space increase(Kulkarni, 2014). A determined attacker, after spending enough time to inspect obfuscated code, might locate the functionality to alter and succeed in the malicious purpose. The renaming obfuscation, layout obfuscation, harden the code 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 (C. Kumar & Bhaskari, 2015).

All theoretical research on software protection via obfuscation typically points to negative results in terms of the existence of perfect obfuscators. (Barak et al., 2012) showed that no general obfuscation algorithm exists that can hide all information leaked by a variant program based on the notion of a virtual black box. 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(Su, Wang, Wu, Li, & Huang, 2012). 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 him/herself the end-user, determined attacker, after spending enough time to inspect obfuscated code, might locate the functionality to alter and succeed in her/his malicious purpose. For this reason, 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 very advanced currently as they can create new code from the obfuscated code that performs the same output even though the original code is obfuscated (Kanani, Srivastava, Gandhi, Parekh, & Gala, 2017). Therefore, it is necessary to enhance the source code obfuscation where it is possible to use different approaches from the renaming techniques in one source file to increase the confusion and complication(Hosseinzadeh et al., 2018). Ordinary obfuscation techniques do not have the ability to prevent reverse engineering, as the reversing tools are very advanced and have the ability to 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(Hofheinz, Malone-Lee, & Stam, 2010).

1.3 Research Question

This section presents the research questions for the thesis. The questions attend the investigation that will be conducted by empirical study for this thesis; the questions are as follows;

RQ1- How effective is the use of mathematical equations for string encryption to transform the text to Chaos stream?

RQ2- How effective is to use Junk obfuscation to rename the identifiers to unknown junk?

RQ3- To what extend Hybrid Obfuscation Technique is well performing in terms of renaming the identifiers and system keywords to junk and Unicode?

1.4 Research Hypothesis

$H_{0\,1}$ Standard obfuscation techniques do not significantly decrease the ability of the reverser to change the original code

$H_{0\,2}$ Standard obfuscation techniques significantly decrease the ability of the reverser to change the original code.

$H_{0\,3}$ Hybrid obfuscation techniques do not significantly decrease the ability of the reverser to change the original code

$H_{0\,4}$ Hybrid obfuscation techniques significantly decrease the ability of the reverser to change the original code.

1.5 Research Objectives

Obfuscation technique is very common in the field of software protection, therefore, the objective of this research to combine the most effective techniques and enhance them in one hybrid obfuscation technique. In order to achieve the main objective, below list highlight specific objectives of the research.

- To enhance string encryption obfuscation technique by applying multiple mathematical equations to create series of Chaos stream.
- To enhance renaming obfuscation technique in which two renaming approaches are applied for the transformation. The first approach is renaming identifiers to junk. This approach creates garbage during decompiling. The second approach is to rename system keywords to Unicode. This approach leads to confusion while reading the source file. The two approaches will create extra layer of junk code during reversing.
- To create a series of junk and chaos stream in the reversed file after merging string encryption and renaming approaches.
- To carry out an empirical study to evaluate the effectiveness of the new proposed hybrid obfuscation technique.

1.6 Scope of the Study

The research has the following scopes;

1. This study is focusing on Java applications, more precisely the source file that contains the source code of java applications. This study is focusing on Java

programming language due to its popularity in the software development industry.

2. This study is limited to Layout and Data Obfuscation techniques.
3. This study uses only four reversing tools that are CAVAJ, JAD, DJ, and JD. These tools are used in this research due to their popularity to most researches and developers in the software development industry. These tools are free of use and well known of the great ability to reverse the software.
4. Other applications such as android, mobile, and web were not considered in the study.

1.7 Contribution to Study

This research proposes a new Hybrid Obfuscation Technique that merges renaming obfuscation and string encryption. The string encryption contains multiple mathematical equations to generate chaos stream that change the form of the code during decompiling. Renaming obfuscation contains two transformations, firstly, Unicode transformation for system keywords to create a confusing look while reading the source code. Secondly, contains junk character transformation to transform the original identifiers to junk which leads to a complicated look. The junk transformation confuses the compiler during decompiling. There is an extra advantage added to the proposed obfuscation technique; that is the compiler translates the entire code to byte code. Byte code is an extra layer of protection; normal user doesn't have the ability to read it. This byte code is only readable by the machine.

1.8 Organization of Thesis

The thesis is formed into seven chapters that descripted as follows;

Chapter 1: Background

This chapter discusses the essentials of the research that include background, problem statement, research questions, objectives, research scope, and research organization.

Chapter 2: Literature Review

This chapter focuses of the current and latest reviews related to the research scope. This chapter discusses the issues and limitation of current obfuscation techniques and guides the researcher towards forming final and best technique possible as compare with the similar current techniques.

Chapter 3: Research Methodology

This chapter presents the flow of the research methodology in conducting the thesis. The main purpose of the research is to propose a Hybrid Obfuscation Technique to prevent prohibited reverse engineering for java's applications source file, and to select a test strategy to validate the effectiveness of the proposed technique. To achieve the research purpose, a sequence of activities and stages should be planned consequently.

Chapter 4: Hybrid Obfuscation Technique

This chapter presents the new proposed Hybrid Obfuscation Technique to protect the code in the source file. The chapter presents the architecture of the proposed technique, the output of the obfuscated code, and discussion on the performance of it.

Chapter 5: Empirical Evaluation

This chapter presents the empirical study that was conducted in this research. Experiment contains two phases; first phase was conducted with reversing tools against the applications that are not protected to get evidence of the need for protection. Second phase of the experiment was conducted with reversing tools against hybrid obfuscated code, to get evidence of the technique effectiveness. Data set for the experiment were four types of applications; procedure math application, image logo, expert system, and obfuscated java application. The reversing tools that were used for reversing are CAVAJ, JAD, DJ, and JD.

Chapter 6: Qualitative Analysis

This chapter focuses on interview. The interview was conducted with four experts who are involved in the research and development of software applications.

Chapter 7: Conclusion

This research focuses on the contributions of the proposed hybrid obfuscation technique. Then it discusses the suggested future works in which improves the protection techniques. This chapter discusses the needed recommendation to improve the protection techniques.

REFERENCES

Akinyode, B. F., & Khan, T. H. (2018). Step by step approach for qualitative data analysis. *International Journal of Built Environment and Sustainability*, *5*(3), 163–174.

Angyal, L., Lengyel, L., & Charaf, H. (2006). An Overview of the State-of-The-Art Reverse Engineering Techniques. *7th International Symposium of Hungarian Researchers on Computational Intelligence An*, 507–516.

Avidan, E., & Feitelson, D. G. (2015). From Obfuscation to Comprehension. *IEEE International Conference on Program Comprehension*, *2015-Augus*, 178–181.

Badier, H., Lann, J. C. Le, Coussy, P., & Gogniat, G. (2019). Transient Key-based Obfuscation for HLS in an Untrusted Cloud Environment. *Proceedings of the 2019 Design, Automation and Test in Europe Conference and Exhibition, DATE 2019*, 1118–1123.

Baker, S. I. B., & Al-Hamami, A. H. (2017). Novel Algorithm in Symmetric Encryption (NASE): Based on feistel cipher. *Proceedings - 2017 International Conference on New Trends in Computing Sciences, ICTCS 2017*, *2018-January*(3), 191–196.

Balachandran, V., & Emmanuel, S. (2013). *Software Protection with Obfuscation and Encryption*. (March 2016).

Batchelder, M., & Hendren, L. (2007). Obfuscating Java: The most pain for the least gain. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *4420 LNCS*, 96–110.

Bergström, E., & Åhlfeldt, R. M. (2016). Foundations and Practice of Security. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *9482*(October), 268–276.

Binshamlan, M. H., Bamatraf, M. A., & Zain, A. A. (2019). The Impact of Control Flow Obfuscation Technique on Software Protection Against Human Attacks. *2019 1st International Conference of Intelligent Computing and Engineering: Toward Intelligent Solutions for Developing and Empowering Our Societies, ICOICE 2019*, (Cil), 2–6.

Borello, J. M., & Mé, L. (2008). Code obfuscation techniques for metamorphic viruses. *Journal in Computer Virology*, *4*(3), 211–220.

Budhkar, S. (2011). *Reverse Engineering Java Code to Class Diagram : An Experience Report*. *29*(6), 36–43.

C., G., & M., S. (2018). Study for Best Data Obfuscation Techniques using Multi-Criteria Decision-Making Technique. *International Journal of Computer Applications*, *180*(43), 50–57.

Ceccato, M., Capiluppi, A., Falcarin, P., & Boldyreff, C. (2015). A large study on the effect of code obfuscation on the quality of java code. *Empirical Software Engineering*, *20*(6), 1486–1524.

Ceccato, M., Di Penta, M., Nagra, J., Falcarin, P., Ricca, F., Torchiano, M., & Tonella, P. (2008). Towards experimental evaluation of code obfuscation techniques. *Proceedings of the ACM Conference on Computer and Communications Security*, 39–45.

Ceccato, M., Di Penta, M., Nagra, J., Falcarin, P., Ricca, F., Torchiano, M., & Tonella, P. (2009). The effectiveness of source code obfuscation: An experimental assessment. *IEEE International Conference on Program Comprehension*, 178–187.

Chan, J. T., & Yang, W. (2004). Advanced obfuscation techniques for Java bytecode. *Journal of Systems and Software*, *71*(1–2), 1–10.

Chen, H., Yuan, L., Wu, X., Zang, B., Huang, B., & Yew, P. (2009). *Control flow obfuscation with information flow tracking*. 391.

Chiba, S. (2000). *Load-Time Structural Reflection in Java*. 313–336.

Cimato, S., De Santis, A., & Petrillo, U. F. (2005). Overcoming the obfuscation of Java programs by identifier renaming. *Journal of Systems and Software*, *78*(1), 60–72.

Cimitile, A., Martinelli, F., Mercaldo, F., Nardone, V., & Santone, A. (2017). Formal methods meet mobile code obfuscation identification of code reordering technique. *Proceedings - 2017 IEEE 26th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE 2017*, 263–268.

Clarke, D. A. (2010). on the Reliability of Zeus-3D. *Astrophysical Journal Supplement Series*, *187*, 119–134.

Dalai, A. K., Das, S. S., & Jena, S. K. (2018). A code obfuscation technique to prevent reverse engineering. *Proceedings of the 2017 International Conference on Wireless Communications, Signal Processing and Networking, WiSPNET 2017*, *2018-Janua*, 828–832.

Darwish, S. M., Guirguis, S. K., & Zalat, M. S. (2010). Stealthy code obfuscation technique for software security. *Proceedings, ICCES'2010 - 2010 International Conference on Computer Engineering and Systems*, 93–99.

Di Troia, F., Visaggio, C. A., Austin, T. H., & Stamp, M. (2017). Advanced transcriptase for JavaScript malware. *2016 11th International Conference on Malicious and Unwanted Software, MALWARE 2016*, 121–128.

Feiyuan, W., Jiying, Z., & El Saddik, A. (2005). Copyright protection of J2EE web applications through watermarking. *Canadian Conference on Electrical and Computer Engineering, 2005*, 1782–1785.

George, N. (2008). Reverse Engineering : Anti-Cracking Techniques. *Program*, 1–19.

Ghosh, S., & Kelly, J. L. (2008). Bytecode fault injection for Java software. *Journal of Systems and Software*, *81*(11), 2034–2043.

Gomes, N. D. (2014). *Software Piracy : An Empirical Analysis Software Piracy : An Empirical Analysis*.

Gong, D., Luo, X., Xie, X., Liu, F., & Lu, B. (2016). Random table and hash coding-based binary code obfuscation against stack trace analysis. *IET Information Security*, *10*(1), 18–27.

Hausknecht, K., & Gruicic, S. (2017). Anti-computer forensics. *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2017 - Proceedings*, 1233–1240. https://doi.org/10.23919/MIPRO.2017.7973612

Hofheinz, D., Malone-Lee, J., & Stam, M. (2010). Obfuscation for cryptographic purposes. *Journal of Cryptology*, *23*(1), 121–168.

Hosseinzadeh, S., Rauti, S., Laurén, S., Mäkelä, J., Holvitie, J., Hyrynsalmi, S., & Leppänen, V. (2018). *Diversi fi cation and obfuscation techniques for software security : A systematic literature review*. *104*(July), 72–93.

Jang, J., De, A., Vontela, D., Nirmala, I., Ghosh, S., & Iyengar, A. (2018). Threshold-defined Logic and Interconnect for Protection against Reverse Engineering. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *PP*(c), 1.

Junod, P., Rinaldini, J., Wehrli, J., & Michielin, J. (2015). Obfuscator-LLVM-Software Protection for the Masses. *Proceedings - International Workshop on Software Protection, SPRO 2015*, 3–9.

Kanani, P., Srivastava, K., Gandhi, J., Parekh, D., & Gala, M. (2017). Obfuscation: Maze of code. *2017 2nd International Conference on Communication Systems, Computing and IT Applications, CSCITA 2017 - Proceedings*, 11–16.

Kang, M. G., Poosankam, P., & Yin, H. (2007). Renovo. *Proceedings of the 2007 ACM Workshop on Recurring Malcode - WORM '07*, 46.

Kasmi, M. A., Mostafa, A., & Lanet, J. L. (2014). Methodology to reverse engineer a scrambled java card virtual machine using electromagnetic analysis. *International Conference on Next Generation Networks and Services, NGNS*, 278–281.

Kienle, H. M., & Müller, H. A. (2010). Rigi-An environment for software reverse engineering, exploration, visualization, and redocumentation. *Science of Computer Programming*, *75*(4), 247–263.

Kim, M. J., Lee, J. Y., Chang, H. Y., Cho, S. J., Park, M., Park, Y., & Wilsey, P. A. (2010). Design and performance evaluation of binary code packing for protecting embedded software against reverse engineering. *ISORC 2010 - 2010 13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, *1*, 80–86.

Koteshwara, S., Kim, C. H., & Parhi, K. K. (2018). Functional encryption of integrated circuits by key-based hybrid obfuscation. *Conference Record of 51st Asilomar Conference on Signals, Systems and Computers, ACSSC 2017*, *2017-Octob*, 484–488.

Kulkarni, A. (2014). A New Code Obfuscation Scheme for Software Protection. *2014 IEEE 8th International Symposium on Service Oriented System Engineering*, 409–414.

Kulkarni, A., & Metta, R. (2014). A code obfuscation framework using code clones. *22nd International Conference on Program Comprehension, ICPC 2014 - Proceedings*, 295–299.

Kumar, C., & Bhaskari, D. L. (2015). *Different Obfuscation Techniques for Code Protection*. *70*, 757–763.

Kumar, R., & Vaishakh, A. R. E. (2016). Detection of Obfuscation in Java Malware. *Physics Procedia*, *78*(December 2015), 521–529.

Leahy, P. (2017). What Is Unicode? *ThoughtCo*, 1. Retrieved from https://www.thoughtco.com/what-is-unicode-2034272

Lehman, E., Leighton, F. T., & Meyer, A. R. (2013). Mathematics for Computer Science. *Proofs*, *3*(2), 1–848. Retrieved from http://courses.csail.mit.edu/6.042/spring12/mcs.pdf

Li, A., Zhang, Y., Zhang, J., & Zhu, G. (2015). A token strengthened encryption packer to prevent reverse engineering PE files. *Proceedings of 2015 International Conference on Estimation, Detection and Information Fusion, ICEDIF 2015*, (ICEDIF), 307–312.

Li, Q. (2009). *Research on Reverse Engineering Technology of.pdf*.

Liang, Z., Li, W., Guo, J., Qi, D., & Zeng, J. (2017). A parameterized flattening control flow based obfuscation algorithm with opaque predicate for reduplicate obfuscation. *Proceedings of 2017 International Conference on Progress in Informatics and Computing, PIC 2017*, 372–378.

Lin, Z., Zhang, X., & Xu, D. (2010). Reverse engineering input syntactic structure from program execution and its applications. *IEEE Transactions on Software Engineering*, *36*(5), 688–703.

Luo, H., Jiang, J., & Zeng, Q. (2006). Code obfuscation for software protection. In *Jisuanji Gongcheng/Computer Engineering* (Vol. 32).

Maskur, M., Sari, Z., & Miftakh, A. S. (2018). Implementation of obfuscation technique on PHP source code. *International Conference on Electrical Engineering, Computer Science and Informatics (EECSI), 2018-Octob*, 738–742.

Meeker, F. M., Wright, B. C., Kann, R. L., Stern, J. R. S., & Lemley, M. A. (2004). United States Court of Appeals for the Federal Circuit. *Biotechnology Law Report*, *23*(6), 770–775.

Memon, J. M., Shams-ul-Arfeen, Mughal, A., & Memon, F. (2006). Preventing reverse engineering threat in java using byte code obfuscation techniques. *Proceedings - 2nd International Conference on Emerging Technologies 2006, ICET 2006*, (November), 689–694.

Mohsen, R. (2016). *Quantitative Measures for Code Obfuscation Security*. (May).

Mohsen, R., & Pinto, A. M. (2008). *Algorithmic Information Theory for Obfuscation Security*. 1–12.

Ohdo, T., Tamada, H., Kanzaki, Y., & Monden, A. (2013). An instruction folding method to prevent reverse engineering in java platform. *SNPD 2013 - 14th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, 517–522.

Peng, Yanru, Chen, Y., & Shen, B. (2019). An adaptive approach to recommending obfuscation rules for Java bytecode obfuscators. *Proceedings - International Computer Software and Applications Conference*, *1*, 97–106.

Peng, Yong, Liang, J., & Li, Q. (2016). A control flow obfuscation method for Android applications. *Proceedings of 2016 4th IEEE International Conference on Cloud Computing and Intelligence Systems, CCIS 2016*, 94–98.

Pizzolotto, D., & Ceccato, M. (2018). Obfuscating Java programs by translating selected portions of bytecode to native libraries. *Proceedings - 18th IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2018*, 40–49.

Popa, M. (2011). Techniques of Program Code Obfuscation for Secure Software. *Journal of Mobile, Embedded and Distributed Systems*, *III*(4). Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.870.4798&rep=rep1&type=pdf

Popa, M. (2014a). *Techniques of Program Code Obfuscation for Secure Software*. (December 2011).

Popa, M. (2014b). *Techniques of Program Code Obfuscation for Secure Software*. (January).

Programme, E. E., Petronas, U. T., & Leepile, T. M. (2006). *A STUDY AND IMPLEMENTATION OF ENCRYPTION , WITH (Electrical & Electronics Engineering) A STUDY AND IMPLEMENTATION OF ENCRYPTION , WITH (Electrical & Electronics Engineering)*.

Qin, S., Wang, Z., Wang, Y., & Xu, K. (2016). A method of JavaScript path obfuscation based on collatz conjecture. *Proceedings - 2015 12th Web Information System and Application Conference, WISA 2015*, 330–333.

Rao, M. S. B., & Giridharakula, D. S. (2013). *International Journal of Advances in Computer Science and Technology Available Online at http://warse.org/pdfs/2013/ijacst06282013.pdf CHAOTIC ALGORITHMS USED FOR ENCRYPTION AND DECRYPTION ON MOVING IMAGES*. *2*(8), 160–164.

Real-time, M. A. T., Ef-, P. C. C. S., & Butaha, M. A. (2017). *Crypto-Compression Systems for Efficient Embedded To cite this version : Thèse de Doctorat*.

Rugaber, S., & Stirewalt, K. (2014). Model-Driven Reverse Engineering. *IEEE Software*.

Samuelson, P. (2002). Reverse engineering under siege. *Communications of the ACM*, *45*(10), 15–20.

Samuelson, P., & Scotchmer, S. (2002). The law and economics of reverse engineering. *Yale Law Journal*, *111*(7), 1575–1663.

Schrittwieser, S., & St, P. (2016). *Protecting Software through Obfuscation : Can It Keep Pace with Progress in Code Analysis ?* *49*(1), 1–40.

Sebastian, S. A., Malgaonkar, S., Shah, P., Kapoor, M., & Parekhji, T. (2016). A study & review on code obfuscation. *IEEE WCTFTR 2016 - Proceedings of 2016 World Conference on Futuristic Trends in Research and Innovation for Social Welfare*, 1–6.

Solomonoff, R. J. (2009). Algorithmic probability: Theory and applications. *Information Theory and Statistical Learning*, 1–23.

Sosonkin, M., Naumovich, G., & Memon, N. (2003). Obfuscation of design intent in object-oriented applications. *DRM 2003: Proceedings of the Third ACM Workshop on Digital Rights Management*, 142–153.

Su, Q., Wang, Z. Y., Wu, W. M., Li, J. L., & Huang, Z. W. (2012). Technique of source code obfuscation based on data flow and control flow tansformations. *ICCSE 2012 - Proceedings of 2012 7th International Conference on Computer Science and Education*, (Iccse), 1093–1097.

Sun, Y. (2003). *How to Render Mathematical Symbols in Java*. (March).

Taha, M. A., Ef-, P. C. C. S., & Butaha, M. A. (2017). *Crypto-Compression Systems for Efficient Embedded To cite this version : Thèse de Doctorat*.

Tang, Z., Chen, X., Fang, D., & Chen, F. (2009). Research on java software protection with the obfuscation in identifier renaming. *2009 4th International Conference on Innovative Computing, Information and Control, ICICIC 2009*, (2007), 1067–1071.

Tang, Z., Kuang, K., Wang, L., Xue, C., Gong, X., Chen, X., … Wang, Z. (2017). SEEAD: A semantic-based approach for automatic binary code de-obfuscation. *Proceedings - 16th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, 11th IEEE International Conference on Big Data Science and Engineering and 14th IEEE International Conference on Embedded Software and Systems*, 261–268.

ul Iman, M., & Ishaq, A. F. M. (2010). Anti-reversing as a tool to protect intellectual property. *Engineering Systems Management and Its Applications (ICESMA), 2010 Second International Conference On*, 1–5. Retrieved from http://www.scopus.com/inward/record.url?eid=2-s2.0-77956582376&partnerID=40&md5=f69ddcf5fec86ec558ad4e2dbdc90f5c

Vijayakumar, A., Patil, V. C., Holcomb, D. E., Paar, C., & Kundu, S. (2017). Physical Design Obfuscation of Hardware: A Comprehensive Investigation of Device and Logic-Level Techniques. *IEEE Transactions on Information Forensics and Security*, *12*(1), 64–77.

Viticchie, A., Regano, L., Torchiano, M., Basile, C., Ceccato, M., Tonella, P., & Tiella, R. (2016). Assessment of source code obfuscation techniques. *Proceedings - 2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation, SCAM 2016*, 11–20.

Wagener, G., Dulaunoy, A., & Engel, T. (2008). An instrumented analysis of unknown software and malware driven by free libre open source software. *SITIS 2008 - Proceedings of the 4th International Conference on Signal Image Technology and Internet Based Systems*, 597–605.

Wagner, N. R. (2003). The Laws of Cryptography with Java Code. *Available Online at Neal Wagner's Home Page*, 1–334. Retrieved from http://www.cs.utsa.edu/~wagner/lawsbookcolor/laws.pdf

Wang, P., Bao, Q., Wang, L., Wang, S., Chen, Z., Wei, T., & Wu, D. (2018). *Software protection on the go*. 26–36.

Wang, Z., Jia, C., Liu, M., & Yu, X. (2012). Branch obfuscation using code mobility and signal. *Proceedings - International Computer Software and Applications Conference*, 553–558.

Wang, Z. Y., & Wu, W. M. (2014). Technique of javascript code obfuscation based on control flow tansformations. *Applied Mechanics and Materials*, *519–520*(Iccse), 389–392.

Winograd, T., Salmani, H., Mahmoodi, H., & Homayoun, H. (2016). Preventing design reverse engineering with reconfigurable spin transfer torque LUT gates. *Proceedings - International Symposium on Quality Electronic Design, ISQED*, *2016-May*, 242–247.

Xiang, G., & Cai, Z. (2010a). The code obfuscation technology based on class combination. *Proceedings - 9th International Symposium on Distributed Computing and Applications to Business, Engineering and Science, DCABES 2010*, (60970064), 479–483.

Xiang, G., & Cai, Z. (2010b). The code obfuscation technology based on class combination. *Proceedings - 9th International Symposium on Distributed Computing and Applications to Business, Engineering and Science, DCABES 2010*, (60970064), 479–483.

Xu, W., Zhang, F., & Zhu, S. (2012). The power of obfuscation techniques in malicious JavaScript code: A measurement study. *Proceedings of the 2012 7th International Conference on Malicious and Unwanted Software, Malware 2012*, 9–16.

Yasin, A., Nasra, I., Yasin, A., & Nasra, I. (2016). *Dynamic Multi Levels Java Code Obfuscation Technique (DMLJCOT)*. (10), 140–160.

You, I. (2010). *Malware Obfuscation Techniques : A Brief Survey*. 297–300.

Zhang, L., Meng, H., & Thing, V. L. L. (2019). Progressive Control Flow Obfuscation for Android Applications. *IEEE Region 10 Annual International Conference, Proceedings/TENCON*, *2018-Octob*(October), 1075–1079.

Zhang, W., Wang, K., & Meng, J. (2015). Research on application of functional FMECA in reverse engineering optimization. *Proceedings of 2015 the 1st International Conference on Reliability Systems Engineering, ICRSE 2015*, 1–5.

Zhang, X., He, F., & Zuo, W. (2008). An inter-classes obfuscation method for Java program. *Proceedings of the 2nd International Conference on Information Security and Assurance, ISA 2008*, 360–365.

Zhou, X., & Xie, J. (2018). Evaluating obfuscation performance of novel algorithm-to-architecture mapping techniques in systolic-array-based circuits. *Proceedings of the 2017 Asian Hardware Oriented Security and Trust Symposium, AsianHOST 2017*, *2018-May*, 127–132.