

Republic of Yemen
**Ministry of High Education &
Scientific Research**
Al – Rayan University
Faculty of Higher Studies



THE IMPACT OF CONTROL FLOW OBFUSCATION TECHNIQUE ON SOFTWARE PROTECTION AGAINST HUMAN ATTACK

**A Thesis Submitted to AL Rayan University to
Complete the Requirements of Obtaining a Master's Degree in Information
Technology**

By

Mohammed Hassan Bin Shamlan.

Supervisors

Dr. Adnan Abdullah Zain

Dr. Mohammed Abdullah Bamatraf.

1441/2020

Republic of Yemen
**Ministry of High Education &
Scientific Research**
Al – Rayan University
Faculty of Higher Studies



THE IMPACT OF CONTROL FLOW OBFUSCATION TECHNIQUE ON SOFTWARE PROTECTION AGAINST HUMAN ATTACK

**A Thesis Submitted to AL Rayan University to
Complete the Requirements of Obtaining a Master's Degree in Information
Technology**

By

Mohammed Hassan Bin Shamlan.

Supervisors

Dr. Adnan Abdullah Zain

Dr. Mohammed Abdullah Bamatraf.

1441/2020

الجمهورية اليمنية

وزارة التعليم العالي والبحث العلمي

جامعة الريان

كلية الدراسات العليا



جامعة الريان
AL-RAYAN UNIVERSITY

تأثير تقنيات تشويش الشفرة المصدرية للبرمجيات على حمايتها ضد هجمات الهندسة العكسية

رسالة مقدمة الى جامعة الريان

لإستكمال متطلبات نيل درجة الماجستير في تقنية المعلومات

إعداد

محمد حسن بن شمالان

إشراف

د. محمد عبدالله بامطرف

د. عدنان عبدالله زين

1441/ 2020

We certify that we have read the present work and that in our opinion it is fully adequate in scope and quality as thesis towards the partial fulfillment of the master degree requirements in

Specialization

From

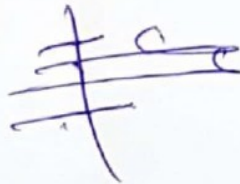
College of

Date

Supervisors

1) Associate Prof. Dr. Adnan Abdullah Zain

Signature:



2)

Signature:

3)

Signature:

Approval of the Proofreader

I certify that the master's dissertation titled ,

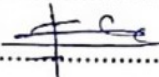
(THE IMPACT OF CONTROL FLOW OBFUSCATION)
TECHNIQUE ON SOFTWARE PROTECTION AGAINST HUMAN ATTACK
submitted by the student MOHAMMED HASSAN BIN GHANILAN

has been linguistically reviewed under my supervision and has become in scientific style and clear from linguistic errors and for that I sign.

Proofreader : Dr. Adnan Abdulla Zain

Academic Title : Associate Prof.

University : ADEN

Signature : 

Date : 18/10/ 2020

Approval of the Scientific Supervisor

I certify that this master's dissertation titled
THE IMPACT OF CONTROL FLOW OBFUSCATION TECHNIQUE
(ON SOFTWARE PROTECTION AGAINST HUMAN...) ATTACK
submitted by the student, MOHAMMED HASSAN BIN SHAMLAN
has been completed in all its stages under my supervision and so
I nominate it for discussion.

Supervisor : Dr. Adnan Abdullah Zain

Signature : 

Date : 18/10/ 2020

ACKNOWLEDGEMENT

Special thanks and praises go to the almighty God for guiding me through this research.

I am extremely grateful to my supervisor, Dr. Adnan Abdullah Zain and Dr. Mohammed Abdullah Bamatraf for their excellent supervision, guidance and encouragement throughout the development of this thesis. I do appreciate their valuable support more than they know.

Abstract

Software developers rely on obfuscation techniques for protecting their source code against reverse engineering attacks. Most of the obfuscation techniques are not based on well-defined measurements to clarify their effectiveness in protecting the source code from both dynamic and static analysis by human subjects. This study presents an experimental technique towards the aim to provide an assessment tool that investigates the impact of control flow obfuscation on software protection against human attacks. The main objective is to estimate how the obfuscation prevents or limits the ability of the attacker to understand and to perform any modification on the source code. An experiment was designed to assess the capabilities of the control flow obfuscation technique with the opaque predicates in preventing or limiting the attacks on source code.

As a result of the statistical analysis used in this study, it is shown that the presence of obfuscation on source code increases seven times the difficulties for the attacker to successfully complete the understanding task. Also, the control flow obfuscation significantly reduces the capability of subjects to correctly perform the understanding tasks while there is no significant difference for modification tasks. Also, it is shown that the presence of obfuscation on source code increases the amount of time needed for subjects to perform modification and understand the source code.

الملخص

يعتمد مطوروا البرمجيات على تقنيات تشويش الشفرات المصدرية للبرمجيات في حماية برامجهم ضد هجمات الهندسة العكسية من قبل المهاجمين. معظم تقنيات التشويش لاتعتمد على قياسات وأدلة واضحة من اجل معرفة مدى فعاليتها وكفائتها في حماية الشفرات المصدرية للبرامج. تقدم هذه الدراسة تحقيقاً عملياً يهدف إلى دراسة تأثير تشويش الشفرات المصدرية للبرامج على حماية البرامج ضد الهجمات البشرية. تم تصميم تجربة عملية تهدف لدراسة وتقييم قدرات تقنيات التشويش في منع أو الحد من قدرة المهاجم على فهم وتنفيذ أي تعديل على الشفرة المصدرية.

تبين نتيجة التحليل الإحصائي المستخدم في هذه الدراسة ، أن وجود التشويش على الشفرة المصدرية للبرنامج يزيد سبعة أضعاف الصعوبات التي يواجهها المهاجم من أجل فهم الشفرة المصدرية للبرنامج بشكل صحيح، بينما لا يؤثر التشويش بشكل كبير في حالة أراد المهاجم التعديل على الشفرة المصدرية للبرنامج. أيضاً ، يتضح أن تشويش الشفرة المصدرية يزيد من مقدار الوقت اللازم للمهاجم لفهم وتعديل الشفرة المصدرية.

Table of Contents

| | |
|-----------------------------|-------------|
| AKNOLGMENT | II |
| ABSTRACT | IV |
| CONTENTS | VI |
| LIST OF FIGURES | VII |
| LIST OF TABLES | VIII |
| LIST OF ABBREVIATION | |

1 INTRODUCTION

| | |
|---|-----------|
| 1.1 THREATS TO SOFTWARE | 1 |
| 1.1.1 PROGRAM ANALYSIS | 2 |
| 1.1.2 REVERSE ENGINEERING | 3 |
| 1.1.3 THREAT MODEL FOR SOFTWARE: UNTRUSTED HOST | 3 |
| 1.2 OBFUSCATION | 5 |
| 1.2.1 OBFUSCATION TECHNIQUES | 5 |
| 1.2.2 OPAQUE PREDICATES | 6 |
| 1.2.3 ADVANTAGES AND DISADVANTAGES OF OBFUSCATION | 7 |
| 1.3 MOTIVATION | 8 |
| 1.4 PROBLEM STATEMENT | 9 |
| 1.5 OBJECTIVES | 9 |
| 1.6 SCOPE AND LIMITATION | 10 |
| 1.7 IMPORTANCE OF THE RESEARCH | 10 |
| 1.8 THESIS LAYOUT | 10 |

2 LITERATURE REVIEW

| | |
|--|-----------|
| 2.1 CONTROL FLOW OBFUSCATION TECHNIQUE: | 12 |
| 2.2 MEASURING THE EFFECTIVENESS OF OBFUSCATION TECHNIQUES | 14 |
| 2.2.1 FIRST APPROACH | 14 |
| 2.2.2 THE SECOND APPROACH | 16 |
| 2.3 REVERSE ENGINEERING OF SOFTWARE | 17 |
| 2.3.1 DEFINITION | 17 |
| 2.3.2 REVERSE ENGINEERING USES | 18 |
| 2.3.3 REVERSE ENGINEERING TOOLS | 19 |
| 2.4 STATISTICAL ANALYSIS | 23 |

| | | |
|-------|---|----|
| 2.4.1 | HYPOTHESIS TESTING | 24 |
| 2.4.2 | STATISTICAL PACKAGE FOR THE SOCIAL SCIENCES (SPSS): | 28 |

3 METHEDODOLOGY **33**

| | | |
|-------|---|----|
| 3.1 | EXPERIMENTAL PLANNING | 33 |
| 3.2 | RESEARCH QUESTIONS | 36 |
| 3.3 | NULL HYPOTHESES | 37 |
| 3.4 | EXPERIMENT MATERIAL AND PROCEDURE | 38 |
| 3.5 | ANALYSIS METHOD | 40 |
| 3.5.1 | MEASUREMENT OF THE SIGNIFICANT DIFFERENCE | 40 |
| 3.5.2 | THE MEASURE OF THE EFFECT SIZE | 41 |

4 RESULTS **43**

| | | |
|-----|--|----|
| 4.1 | TASK RESULTS | 43 |
| 4.2 | SIGNIFICANT DIFFERENCE RESULTS | 43 |
| 4.3 | EFFECT SIZE RESULTS | 45 |
| 4.4 | COMPARE THE OBTAINED RESULTS WITH PREVIOUS STUDIES | 45 |

5 CONCLUSION AND FUTURE WORK **48**

| | | |
|-----|-------------|----|
| 5.1 | CONCLUSION | 48 |
| 5.2 | FUTURE WORK | 49 |

LIST OF REFRENCES

LIST OF PUBLICATIONS

LIST OF FIGURES

| Figure | Page |
|--|-------------|
| 2.1 Ilasm.exe | 19 |
| 2.2 .Net Reflector | 20 |
| 2.3 dnSpy | 21 |
| 2.4 SPSS | 27 |
| 3.1 Activation screenshot | 34 |
| 3.2 Login screenshot | 35 |
| 4.1 Boxplots of the average time for tasks execution | 43 |

LIST OF TABLES

| Table | Page |
|--|------|
| 3.1 Our work vs based work | 32 |
| 3.2 Summarization of the experiment | 33 |
| 3.2 Total subjects and task distribution | 34 |
| 4.1 Correct and wrong tasks | 42 |
| 4.2 comparison with previous studies | 46 |

List of Abbreviations

| Symbol | Nomenclatures |
|---------------|-----------------------------------|
| ANOVA | Analysis of Variance |
| CIL | Common Intermediate Language |
| CF | Control Flow |
| DLL | Dynamic Link Library |
| GUI | Graphics User Interface |
| JIT | Just In Time compiler |
| MANOVA | Multivariate analysis of variance |
| RR | Relative Risk |
| SGX | Software Gard Extension |

Chapter 1

1 Introduction

Computer software is a group of instructions that tells the computer how to work; software developers distribute their applications in many ways. Malicious users are performing reverse-engineering attacks to change the software so that it meets their needs. Generally, these sorts of attacks are proposed performed to overcome the security code which prevents cracking the software and obtains some valuable information to use in an unauthorized manner also copying the functionality, or manipulating and stealing sensitive information or algorithms. A programming language such as C sharp is widely used in software development. C sharp programming language compiling process translate source code into a common intermediate language (CIL) which is assembly to byte-code and during the execution of CIL assembly it passes through just in time (JIT) compiler to generate native code, the attacker can easily attack a byte-code and extract valuable information such as algorithms or registration serial number or cryptographic keys available in the software.

1.1 Threats to Software

Computer software faces many different kinds of threats; including software piracy, tampering, unauthorized modifications, and many other threats. Among these threads, we distinguish two common threats to computer software: reverse engineering and program analysis, this because reverse engineering and program analysis is considered the cornerstone for many different types of attacks on software.

1.1.1 Program Analysis

Program analysis used to analysis the structural or behavior of the software. Those processes of analysis are regarding to a property such as optimization, which focuses on improving the program's performance while reducing the usage of the resource by the program or it may focuses on ensuring that the program perform its tasks correctly.

Analyzing a program can perform in many different sorts depending on status of understudy program, if the analysis performed while the target program are not running or in compile time this is called a static program analysis while if the analysis is performed at run time of the target program this is called dynamic program analysis those are the two kinds of program analysis.

1.1.1.1 Static program analysis

This type of analysis is conducted to examine and study the program properties without executing the code. It's useful for code error detection and compiler optimization. Typical static program analysis techniques are: data flow, control flow, alias analysis, type analysis and abstract interpretations [1]. Static analysis is conservative, which means the properties that are found by static analysis techniques are weaker than the ones that may actually be true (over-approximation).

1.1.1.2 Dynamic program analysis

This type of analysis is conducted to examine and study the program properties at the run time of the program. For this analysis to be effective, it must be examine against all possible inputs to cover almost possible outputs. This kind of analysis can be used in software testing for memory error detection ,locating the buggy code, runtime error detection as exceptions and discover security attacks vulnerabilities.

1.1.2 Reverse Engineering

Reverse engineering techniques typically use program analysis tools, in order to perform the reverse process of reconstructing the program source code. It can be perceived as a methodology, which combines both static and dynamic program analysis tools.

The reverse engineering process consists of several stages that aim to produce the source code from the binary code. It starts with disassembly phase, where the machine code is translated to assembly code, and then it ends up with the decompilation phase, which rebuilds the higher level representations of the program from the assembly code. Typically, during this process, static program analysis techniques are employed first, followed as required by dynamic program analysis tools.

1.1.3 Threat Model for Software: Untrusted Host

Program analysis and reverse engineering are the typical attacks on software that are running on an untrusted host. This model of attack is widely known as white-box model, where the adversary or attacker has the host or the system under her/his full control. Aucsmith [2] characterizes three different levels for this model of attack:

1. The attacker uses standard debuggers and system diagnostic tools, no special analysis tools are required.
2. Specialized software analysis tools involved, such as specialized debuggers and sophisticated reverse engineering tools.
3. Specialized hardware analysis tools are employed; these tools include, for example, CPU emulators, and bus logic analyzers.

White-box attacks are a very powerful type of attack especially for open computing platforms such as PCs. The attackers have unlimited access to program binaries; however,

it is assumed that the attackers have very limited knowledge about the software source-code. Having the attacker operating in white-box model, does not rule out the possibility of subjecting the targeted software to black-box attack. In black-box model attacks, the software is considered as an oracle, because the attacker can only analyzing the external behavior of the software, where the internal knowledge of software is not required, or the attacker does not examine it.

Collberg and Nagra [3] propose an attack methodology which resembles attackers' behavior or strategy during the attack process. The attacker goes through five phases: black-box phase, dynamic analysis phase, static analysis phase, editing phase, and scripting phase. The attacker starts with black-box testing in order to reveal the external behavior of the Software; however this stage can be skipped if the attacker has a comprehensive understanding of the software's functionality. Then the attacker moves to the dynamic analysis phase in order to gain more understanding of the internal working of the program. At this stage, the attacker has a high level understanding of the software and how it works. In the static analysis phase, the executable code is checked and investigated directly. This covers the reverse engineering process that we discussed earlier. Now, the attacker is assumed to have very detailed understanding and a complete picture of the whole software design and its implementation. For example, the proprietary algorithms can be exposed, cryptography keys are revealed and vulnerabilities are discovered at this stage; therefore, the confidentiality of software is compromised. Reaching the editing phase, the adversary (attacker) uses the acquired knowledge of the software's inner work to modify its executable, or to integrate it with her/his own software for interoperability purposes. So far the four stages are conducted manually, in practice these phases are not followed in order, they are interleaved with each other using a trial and error process, pattern matching,

running and testing the code multiple times. Finally, when the attacker has fulfilled her/his own goals and has enough confidence in the soundness of her/his work, a scripting code is written in order to automate this process.

1.2 Obfuscation

Obfuscation uses for software intellectual property protection. The new version of the source code which is obfuscated version must equal in function with the original one but harder to recognize or comprehend [4]. Obfuscation is usually used to prevent manual inspection of program internals. The most general strategies are renaming the variables and methods and flattening down the program structures to become more complex. It can conceal the location of a flaw in an obfuscated patch [5]. Obfuscation means that the original code is subjected to a series of transformations by changing its structure, so that it becomes more difficult to understand, while preserving its original functionalities [6].

Definition: let O is an obfuscator which can transform a program P into its obfuscated version $O(P)$ which has the same functionality F as P , such that the $F(P) = F(O(P))$ and such that it is unintelligible for an adversary who is trying to recover P from $O(P)$ [7].

1.2.1 Obfuscation techniques

Code obfuscation techniques are classified into many classes: data obfuscation, control flow, layout, and procedural obfuscation [8].

1.2.1.1 Data Obfuscation

Data Obfuscation alters the name of variables and functions to become much harder to understand by human. Data obfuscation gathers all the transformations that obscure the data structures used in a program. This includes for instance the changes in variable

representation, the conversion from static data to procedural data, the split or the aggregation of variables.

1.2.1.2 Control Flow Obfuscation

In Control Flow Obfuscation technique the flow of control within the program's code is altered, by creating conditional, branching, and iterative constructs which results into valid executable logic, but on decompiling, it yields non-deterministic semantic results. In this technique, the code is divided into different blocks, which are encapsulated in a selective structure with each block in a separate case, with an encapsulated selection. A control variable that represents the program state is used to ensure the correct flow. To overcome static analysis, control flow fattening is used.

1.2.1.3 Layout Obfuscation

Layout Obfuscation Layout obfuscation scrambles a program layout while keeping the syntax intact. For example, it may change the orders of instructions or scramble the identifiers of variables and classes.

1.2.2 Opaque Predicates

To obfuscate the control flow the Opaque predicate is generally used. A predicate is an expression which logically evaluates to either "true" or "false". An Opaque predicate is defined as follows; a predicate p , is named an Opaque predicate, if the outcome of a program P is known before applying the predicate p , which is known to the programmer but not known to the attacker. [9].

1.2.3 Advantages and Disadvantages of obfuscation

The most important advantages of obfuscation are:

1. Protection: obfuscation protects against static and dynamic analysis attacks as it raises the bar for the attacker, i.e. the attacker requires more time or resources to achieve his goal.
2. Diversity: it is possibility to create different instances of the original program.
3. Low cost: obfuscation involves a low maintenance cost due to automation of the transformation process and compatibility with existing systems.
4. Platform independence: obfuscation transformations can be applied on high-level code so that platform independence is preserved.

The most important disadvantages are:

1. Performance overhead: every transformation introduces an extra cost in terms of memory usage and execution time necessary to execute the obfuscated program.
2. No long term security Obfuscation: by means of code transformations does not provide perfect security that makes analysis infeasible. Instead it makes program analysis harder, though not impossible.

1.3 Motivation

Code obfuscation provides a promising technical approach for protecting software. However, most of the current state-of-the-art obfuscation techniques are not based on well-defined security principles that help to certify their success in protecting software.

The current notion of code obfuscation is based on a fixed metric for program complexity, which is usually defined in terms of syntactic program features, such as code length, number of nesting levels and numbers of branching instructions. There is a need to practically examine and verify the effectiveness of obfuscation transformation based on new quantitative means.

In order to evaluate the quality of software protection such as code obfuscation, we have to capture quantitatively the security of code transformations, and study the code-obfuscation resilience against an adversary, taking into account the adversary's capabilities, such as malicious software reverse engineering, static and dynamic analysis techniques, etc.

1.4 Problem statement

Most of the obfuscation techniques are not based on well-defined measurements to clarify their effectiveness in protecting the source code from both dynamic and static analysis by human subjects. This study targets the lack of experimental evaluation of obfuscation techniques where this problem makes the effectiveness of any proposed obfuscation technique is not empirically proven against analysis attacks.

1.5 Objectives

The goal of this study is to evaluate the effectiveness of the source code control flow obfuscation technique with the opaque predicates for the purpose of evaluating their effectiveness to make the code more difficult for malicious attacks by human subjects. This can be achieved by:-

The objectives of this study are to:-

- Study how the control flow obfuscation technique with opaque predicates increases the time needed to successfully analysis the source code by a human attacker.
- Determine how the control flow obfuscation technique with opaque predicates reduces the attacker's capability to understand and modify the source code.

1.6 Scope and limitation

This study will empirically assist the effectiveness of the control flow obfuscation technique with opaque predicates in protecting the source code against malicious software reverse engineering, static and dynamic analysis attacks. The study will not include any other obfuscation techniques.

1.7 Importance of the research

The results of this study will help the software developers to choose the most appropriate and best obfuscation technique in order to protect their applications. The study will help researchers to find out - through the experimental approach - how obfuscation techniques are powerful and efficient in software protection against static and dynamic analysis attacks by human where a human is considered more dangerous.

1.8 Thesis Layout

This thesis is organized as the following chapters:

Chapter 1: A general introduction of the thesis is provided.

Chapter 2: Presents a thorough literature review on the research area of the thesis with emphasis on the obfuscation effectiveness studies.

Chapter 3: The proposed methodology and the analysis methods are presented and explained.

Chapter 4: In this chapter, the results and their detailed discussions are given.

Chapter 5: In this chapter, the conclusions of the study are presented, and an outline of future work ideas based on this thesis is also given.

Chapter2

2 Literature Review

This chapter presents a survey about the previous works on software protection using code obfuscation. We organized the contents of this chapter into the following sections. Firstly we provide a literature review about control flow obfuscation technique specifically, secondly we presents a survey about the researches for measuring the effectiveness of obfuscation techniques. Lastly we details reverse engineering definition and list some reverse engineering tools.

2.1 Control flow obfuscation technique:

The paper at [10] proposed three models for control flow obfuscation to obfuscate android applications they target a Dalvik byte code level, those proposed models perform a distribution the original control flow of the application by using packing-switch and try-catch constructs, also the combination of these constructs. they presented a register-type separation technique, which solve the Android runtime type conflict problem in the obfuscated applications. The paper previews some analysis that shows the effectiveness of the proposed models.

A new control flow obfuscation scheme called generalized dynamic opaque predicates has been developed at [11]; this approach extends the scope of the application by transforming the program structure like loops, branch and the straight line code. This system does not require the opaque to be adjacent so it is become. the researchers have developed a prototype tool and evaluated it by obfuscating the GNU utilities more resilient to the DE obfuscation techniques.

A novel control flow obfuscation has been proposed at [12], to protect android applications source code, they develop an algorithm to insert an irrelevant code and flatten the control flow which insuring the strength of the obfuscation and minimize its cost. Also they improve some traditional methods of control flow flattening to further reduce the costs of obfuscation, they introduce a method to strength the opaque predicates by convert the identification of the opaque predicates in the whole program into a graph traversal problem. An experiment was conducted to evaluate the proposed methods shows that the proposed methods work well.

A new novel control flow obfuscation technique is proposed in [13], this technique implements Turing machines to simulate the computation of branch conditions. By weaving the original program with Turing machine components, this results a complicated control flow graph, the proposed method provides many advantages such that its ability to obfuscate any program components.

The researchers at [14] introduce a pragmatic control flow obfuscation that gives the developer the ability to customize the tradeoff between the overhead and the achieved complexity. the methods of the application will be obfuscating in different round using tray-catch and packed-switch constructs, where the large method will have obfuscated in sense of many fragments during the early rounds, after each round the cyclomatic complexity based metric is used to calculate the complexly and check it against the desired complexity, this checking process will be done incrementally until the target complexity is reached.

The paper at [15] combines Intel Software Guard Extension (SGX) technology and a program transformation a hardware-assisted control flow obfuscation solution, and propose control flow hider (CF Hider). This mechanize of protection based on moving the conditions of statements into the CF Enclave, which is an opaque and trusted SGX enclave, and inserting fake branch statements into the program to further obfuscate the control flow. CF Hider is the first solution that leverages SGX technology to protect control flow confidentiality, which achieves a high confidentiality guarantee and a low performance overhead.

The study at [16] presented a new control flow technique by rewriting the original code of the program in the continuation passing style. Where the original code is encoded through a function pointers and higher order combinatory, this results a fragmented control flow graph of the original code. Thus the code tempering attack became harder. The paper also presents a prototype of the developed technique to obfuscate C language source code to compare this technique with the other available techniques.

2.2 Measuring the effectiveness of obfuscation techniques

There are two major approaches of research for measuring the effectiveness of obfuscation techniques in protecting the software: one is based on source code metrics like code complexity factors for example number of code lines and variable renaming, the second is based on experimental valuation against a human attack where a human is considered more dangerous.

2.2.1 First approach

One of the early researches was conducted by Collberg et al. [17], in which they consider some fundamental metrics in their evaluation such as complexity metric which

indicates how much the difficulty to understand the obfuscation code by the attacker as compared to the original code. The other metric used is the degree of nested statements and execution time of the obfuscated code.

A large scale study was conducted in [18], in order to measure the effects of obfuscation. May metrics were used for the measurements. The changes that occurred to Java codes were quantified in sense of complexity, modularity.

A quantitative model with a set of security metrics derived from the well-known Kolmogorov complexity was presented in [19]. It is reported that most of the obfuscation techniques are not based on well-defined security principles that help to certify their success in protecting software.

In essence, there are two related challenges that arise in code obfuscation security: the lack of a theoretical foundation, and the difficulty of finding consistent and, theoretically and empirically, valid measures of code obfuscation quality.

In [20], an obfuscation approach which yields probabilistic control flow within a given method was given. In this approach, given the same input values, different multiple execution traces are obtained, while preserving the semantics. The scheme presented led to a harder dynamic analysis, which is achieved by embedding an obfuscation graph containing multiple v-paths.

The study at [21], try to evaluate a set of 7 obfuscation methods on 240 Android Packages in case of applying those methods iteratively, they define and trays to find out some structural properties of these methods regarding different complexity metrics. In this research the results show that most of obfuscation methods used in the study are exhibit

stable properties regarding different complexity metrics, while a few obfuscation methods have unstable properties regarding some of the metrics.

At [22] the effects of code obfuscation on Android application in case of similarity check is analyzed because after obfuscated a software it's difficult to detect a software theft or to determine which copy is the original one and which one is the stolen copy, thus adversaries can also use obfuscation techniques to keep hiding stolen software. The empirical measurements were done on five different Android apps with DashO obfuscator. Experimental results show that similarity measures at byte code level are more effective than those at source code level to analyze software similarity.

2.2.2 The second approach

A series of assessments to evaluate the capability of code obfuscation techniques was given in [23]. An empirical approach was adopted. Opaque predicates and identifier renaming were used to decrease the capability of reverse engineer to accurately complete or perform an attack.

In [24], two experiments were conducted to evaluate the renaming, where this technique targets the names of the identifiers in the program code.

The research in [23] and [24] are similar in the sense that they examine the identifier renaming obfuscation technique.

In [25] an experiment that aims to assess the capability in preventing attacks on obfuscated data, was conducted with student pretends as attackers to attack a programs written in C programming language.

A recent large scale study was presented at [26] where an online android developer was participated in an experiment for testing the effectivity of android developers to use

obfuscation for protecting their application. The study indicates that the majority of app developers do not obfuscate their core code, and even when they do they do not use all of the available features. These results might indicate that developers either only obfuscate critical parts of their application or do not fully understand the concept of obfuscation.

2.3 Reverse Engineering of Software

2.3.1 Definition

The Institute of Electrical and Electronics Engineers defined reverse engineering as "the process of analyzing a subject system to identify the system's components and their interrelationships, and to create representations of the system in another form or at a higher level of abstraction", where the "subject system" is the end product of software development. Reverse engineering is a process of examination only: the software system under consideration is not modified (which would make it re-engineering or restructuring). Reverse engineering can be performed from any stage of the product cycle, not necessarily from the functional end product.[32]

There are two components in reverse engineering: redocumentation and design recovery. Redocumentation is the creation of new representation of the computer code so that it is easier to understand. Meanwhile, design recovery is the use of deduction or reasoning from general knowledge or personal experience of the product in order to fully understand the product functionality.[32] It can also be seen as "going backwards through the development cycle".[33] In this model, the output of the implementation phase (in source code form) is reverse-engineered back to the analysis phase, in an inversion of the traditional waterfall model.

Software anti-tamper technology like obfuscation is used to deter both reverse engineering and re-engineering of proprietary software and software-powered systems.

In practice, two main types of reverse engineering emerge. In the first case, source code is already available for the software, but higher-level aspects of the program, perhaps poorly documented or documented but no longer valid, are discovered. In the second case, there is no source code available for the software, and any efforts towards discovering one possible source code for the software are regarded as reverse engineering. This second usage of the term is the one most people are familiar with.

On a related note, black box testing in software engineering has a lot in common with reverse engineering. The tester usually has the API, but their goals are to find bugs and undocumented features by bashing the product from outside [34].

2.3.2 Reverse engineering uses

Reverse engineering is used in a variety of fields such as software design, software testing and programming [35].

- ❖ **Software design**, reverse engineering enables the developer or programmer to add new features to the existing software with or without knowing the source code. Different techniques are used to incorporate new features into the existing software.
- ❖ Reverse engineering is also very beneficial in **software testing**, as most of the virus programmers don't leave behind instructions on how they wrote the code, what they have set out to accomplish etc. Reverse engineering helps the testers to study the virus and other malware code. The field of software testing, while very extensive, is also interesting and requires vast experience to study and analyze virus code.
- ❖ The third category where reverse engineering is widely used is in **software security**. Reverse engineering techniques are used to make sure that the system does not have any major vulnerabilities and security flaws.

The main purpose of reverse engineering is to make the system robust so as to protect it from spywares and hackers. In fact, this can be taken a step forward to Ethical hacking, whereby you try to hack your own system to identify vulnerabilities.

- ❖ **Other purposes** of reverse engineering include security auditing, removal of copy protection or cracking , circumvention of access restrictions often present in consumer electronics, customization of embedded systems such as engine management systems.

2.3.3 Reverse Engineering Tools

As mentioned above, reverse engineering is the process of analyzing the software to determine its components and their relationships. The process of reverse engineering is accomplished by making use of some tools that are categorized into debuggers or disassemblers, hex editors, monitoring, and decompile tools [35]:

- ❖ **Disassemblers:** A disassembler is used to convert binary code into assembly code and also used to extract strings, imported and exported functions, libraries, etc. The disassemblers convert the machine language into a user-friendly format. Different dissemblers specialize in certain things. One example of the intermediate language Disassembler tools is Ildasm.exe as shown in figure 2.1. Ildasm.exe takes a portable executable file that contains intermediate language code and creates a text file suitable as input to Ildasm.exe. This tool is automatically installed with Visual Studio [36].

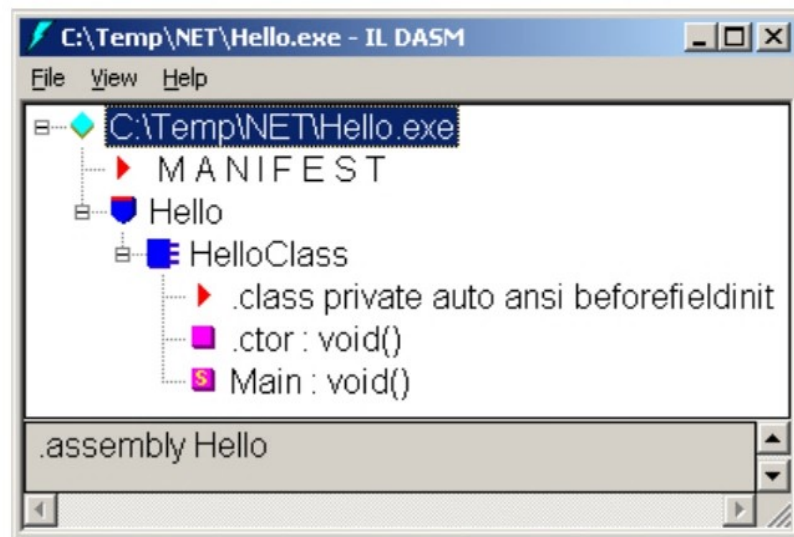


Fig 2.1. Ilasm.exe

- ❖ **Decompiler:** As its name implies, a decompiler performs the opposite operation of a compiler: it transfers compiled byte-code to corresponding high-level source code. By knowing the relationship between the high-level code and its corresponding IL byte-code, a decompiler can identify and convert the IL instructions into their high-level equivalent. In terms of decompilers, for the .NET runtime, the famous useful tool is .NET Reflector. It is considered one of the “Ten Must-Have” utilities for developers by MSDN magazine [37]; this free software provides advanced capabilities such as decompilation, a class browser, and static analysis for executables. Figure 2.2 shows the .NET Reflector user interface

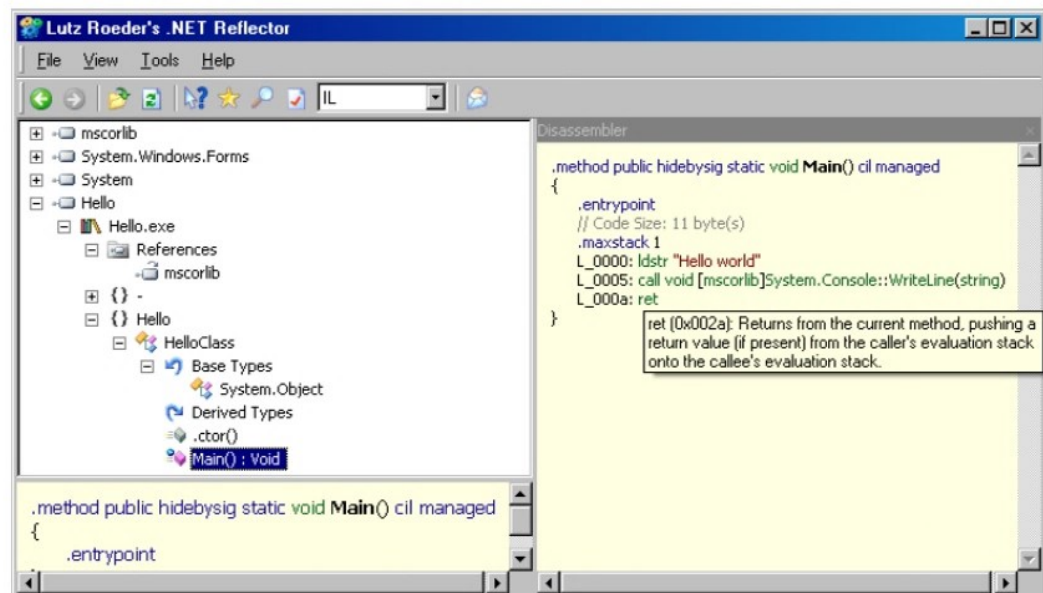


Fig 2.2. .Net Reflector

- ❖ **Debuggers:** This tool expands the functionality of a disassembler by supporting the CPU registers, the hex dumping of the program, view of stack etc. Using debuggers, the programmers can set breakpoints and edit the assembly code at run time. Debuggers analysis the binary in a similar way as the disassemblers and allow the reverser to step through the code by running one line at a time to investigate the results. dnSpy is a debugger and .NET assembly editor. You can use it to edit and debug assemblies even if you don't have any source code available. Figure 2.3 shows the dnSpy user interface.

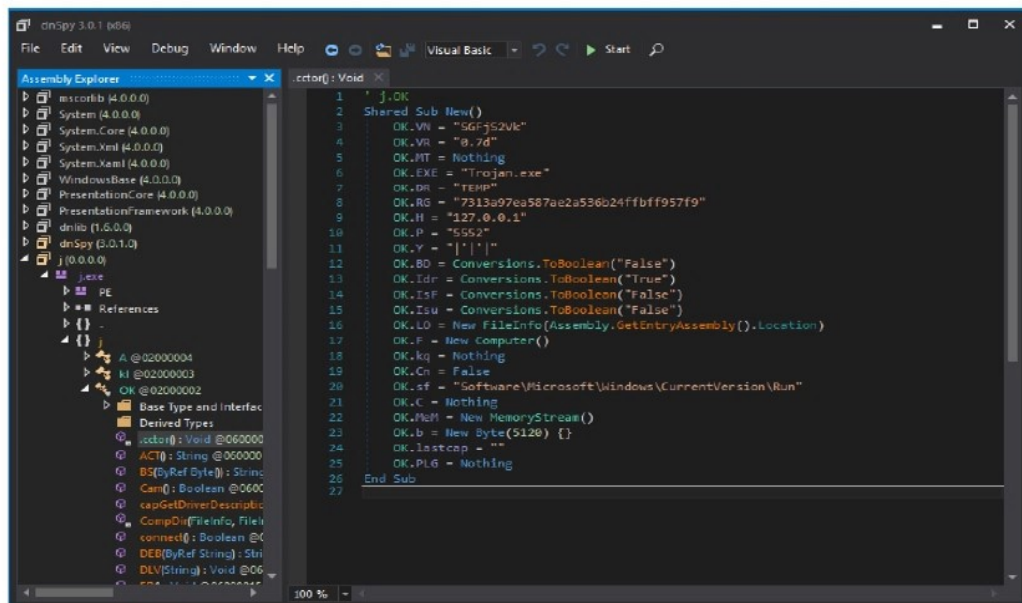


Fig 2.3. .dnSpy

- ❖ **Hex Editors:** These editors allow the binary to be viewed in the editor and change it as per the requirements of the software. There are different types of hex editors available that are used for different functions.
- ❖ **PE and Resource Viewer:** The binary code is designed to run on a windows based machine and has a very specific data which tells how to set up and initialize a program. All the programs that run on windows should have a portable executable that supports the dynamic link libraries the program needs to borrow from.

2.4 Statistical Analysis

Statistics is the discipline that concerns the collection, organization, analysis, interpretation and presentation of data. In applying statistics to a scientific, industrial, or social problem, it is conventional to begin with a statistical population or a statistical model to be studied. Populations can be diverse groups of people or objects such as "all people living in a country" or "every atom composing a crystal". Statistics deals with every aspect of data, including the planning of data collection in terms of the design of surveys and experiments.

Statistical analysis is a component of data analytics. It involves collecting and scrutinizing every data sample in a set of items from which samples can be drawn. A sample, in statistics, is a representative selection drawn from a total population. Statistical analysis can be broken down into five discrete steps, as follows:

- Describe the nature of the data to be analyzed.
- Explore the relation of the data to the underlying population.
- Create a model to summarize understanding of how the data relates to the underlying population.
- Prove (or disprove) the validity of the model.
- Employ predictive analytics to run scenarios that will help guide future actions.

The **goal of statistical analysis** is to identify trends. A retail business, for example, might use statistical analysis to find patterns in unstructured and semi-structured customer data that can be used to create a more positive customer experience and increase sales.

2.4.1 Hypothesis testing

Hypothesis: Research usually starts with a problem. Questions, objectives and hypotheses provide a specific restatement and clarification of the problem statement/research question. Hypothesis is a tentative explanation that accounts for a set of facts and can be tested by further investigation. Hypothesis should be statements expressing the relation between two or more measurable variables. It should carry clear implications for testing the stated relations.

A good hypothesis must be based on a good research question.

- Hypothesis should be simple, specific and stated in advance.
- It must have explanatory power.
- It must state the expected relationship between variables.
- It must be testable.
- It should be consistent with the existing body of knowledge.
- It should be stated as simply and concisely as possible.

Types of Hypotheses: hypotheses are classified into two types: **Null Hypotheses:** is a statement that there is no actual relationship between variables. (H_0 or H_N). A null hypothesis may read, “There is no difference between.....” H_0 states the opposite of what the experimenter would expect or predict. The final conclusion of the investigator will either retain a null hypothesis or reject a null hypothesis in favor of an alternative hypothesis. Not rejecting H_0 does not really mean that H_0 is true. There might not be enough evidence against H_0 . Once the null hypothesis has been stated, it is easy to construct the alternative hypothesis. It is essentially the statement that the null hypothesis is

false. Example can be “There is no significant difference in the anxiety level of children of High IQ and those of low IQ.”

Alternate Hypotheses: is a statement that suggests a potential outcome that the researcher may expect. (H_1 or H_A). It is established only when a null hypothesis is rejected. Often an alternative Hypothesis is the desired conclusion of the investigator. The two types of alternative hypothesis are: Directional Hypothesis and Non-directional Hypothesis.

Directional Hypothesis: It is a type of alternative hypothesis that specifies the direction of expected findings. Sometimes directional hypothesis are created to examine the relationship among variables rather than to compare groups. Directional hypothesis may read, “is more than”, “will be lesser” Example can be “Children with high IQ will exhibit more anxiety than children with low IQ”

Non-directional Hypothesis: It is a type of alternative hypothesis in which no definite direction of the expected findings is specified. The researcher may not know what can be predicted from the past literature. It may read, “...There is a difference between...”

Hypothesis Testing: Hypothesis testing is a statistical technique that is used in a variety of situations. Testing a hypothesis involves

- Deducing the consequences that should be observable if the hypothesis is correct.
- Selecting the research methods that will permit the observation, experimentation, or other procedures necessary to show whether or not these do occur.
- Applying this method and gathering the data that can be analyzed to indicate whether or not the hypothesis is supported.

The following descriptions of common terms and concepts refer to a hypothesis test in which the means of two populations are being compared.

In the field of statistics, a hypothesis is a claim about some aspect of a population. A hypothesis test allows us to test the claim about the population and find out how likely it is to be true. The hypothesis test consists of several components; two statements, the null hypothesis and the alternative hypothesis, the test statistic and the critical value, which in turn gives us the p-value and the rejection region (), respectively.

Test Statistic: The test statistic is the tool researcher use to decide whether or not to reject the null hypothesis. It is obtained by taking the observed value (the sample statistic) and converting it into a standard score under the assumption that the null hypothesis is true. The test statistic depends fundamentally on the number of observations that are being evaluated. It differs from situation to situation. The whole notion of hypothesis rests on the ability to specify (exactly or approximately) the distribution that the test statistic follows.

Significance: The significance level is a measure of the statistical strength of the hypothesis test. It is often characterized as the probability of incorrectly concluding that the null hypothesis is false. The significance level should be specified up front. The significance level is typically one of three values: 10%, 5%, or 1%. A 1% significance level represents the strongest test of the three. For this reason, 1% is a higher significance level than 10%.

Power: Related to significance, the power of a test measures the probability of correctly concluding that the null hypothesis is true. Power is not something that researcher can choose. It is determined by several factors, including the significance level selected and the size of the difference between the things researcher is trying to compare. Unfortunately,

significance and power are inversely related. Increasing significance decreases power. This makes it difficult to design experiments that have both very high significance and power.

Critical Value: The critical value is the standard score that separates the rejection region () from the rest of a given curve. The critical value in a hypothesis test is based on two things: the distribution of the test statistic and the significance level. The critical value(s) refer to the point in the test statistic distribution that give the tails of the distribution an area (meaning probability) exactly equal to the significance level that was chosen.

Decision: Your decision to reject or accept the null hypothesis is based on comparing the test statistic to the critical value. If the test statistic exceeds the critical value, you should reject the null hypothesis. In this case, you would say that the difference between the two population means is significant. Otherwise, you accept the null hypothesis.

P-Value: It is the area to the left or right of the test statistic. The p-value of a hypothesis test gives another way to evaluate the null hypothesis. The p-value represents the highest significance level at which particular test statistic would justify rejecting the null hypothesis. For example, if the significance level of 5% is chosen, and the p-value turns out to be .03 (or 3%), it would be justified in rejecting the null hypothesis.

One-Tailed Test: One-tailed test alludes to the significance test in which the region of rejection appears on one end of the sampling distribution. It represents that the estimated test parameter is greater or less than the critical value. When the sample tested falls in the region of rejection, i.e. either left or right side, as the case may be, it leads to the acceptance of alternative hypothesis rather than the null hypothesis.

Two-tailed Test The two-tailed test is described as a hypothesis test, in which the region of rejection or say the critical area is on both the ends of the normal distribution. It determines whether the sample tested falls within or outside a certain range of values. Therefore, an alternative hypothesis is accepted in place of the null hypothesis, if the calculated value falls in either of the two tails of the probability distribution.

2.4.2 Statistical Package for the Social Sciences (SPSS):

Statistical Package for the Social Sciences is a software package which is used in statistical analysis of data. It was developed by SPSS Inc. to edit and analyze all sorts of data and acquired by IBM in 2009. In 2014, the software was officially renamed IBM SPSS Statistics. The basic application of this program is to analyze scientific data related with the social science. This data can be used for market research, surveys; data mining, etc. figure 2.4 shows the main screen of the SPSS software.

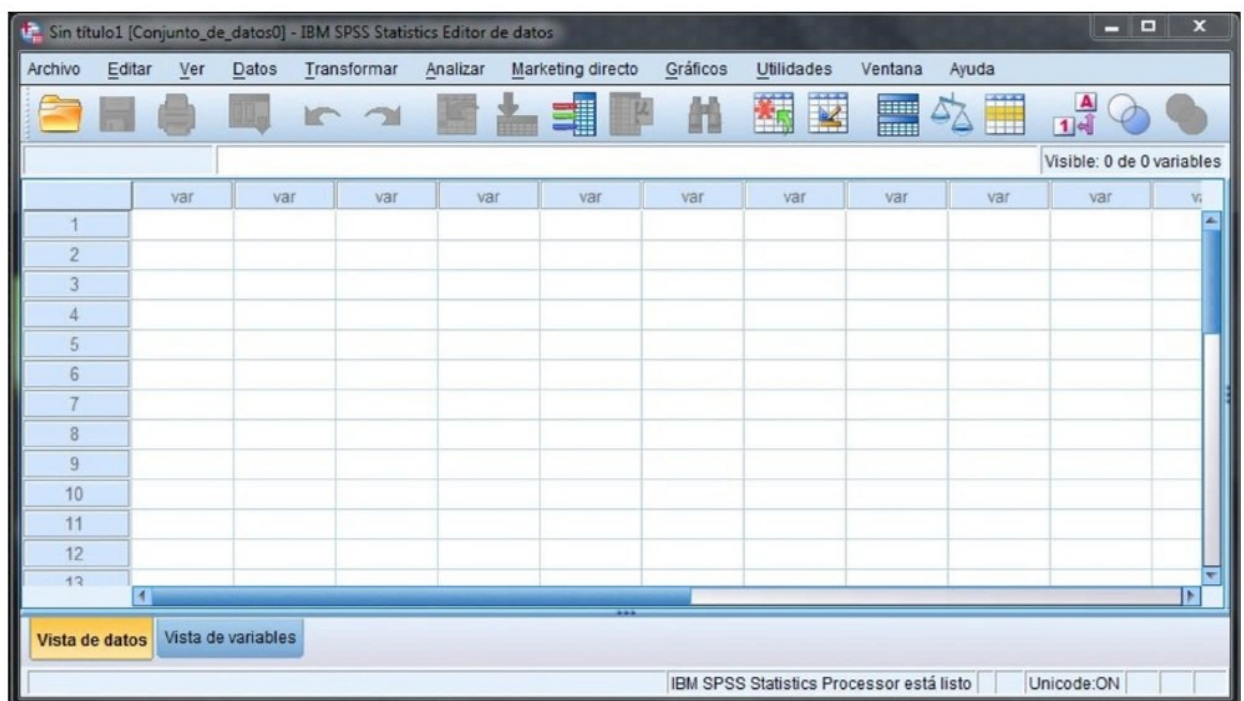


Fig 2.4 SPSS

Features and Benefits of SPSS:

1. It is an easy to navigate GUI (graphics user interface) with a wide range of options for users. It can generate syntax which can be edited or saved.
2. SPSS assignment help experts suggest that it makes data management easy with its features of recording and transforming scores, selecting cases on the basis of score values, combining the files by adding wither cases or variables.
3. SPSS is easy to learn command language wherein explanations for keywords and subcommands are available online.
4. Spatio-Temporal Prediction is a technique that creates linear models when data has been garnered over a period of time from different locations.

2.4.2.1 The statistical methods which are used in SPSS are:

Descriptive Statistics: SPSS help professionals suggest that descriptive statistics describe the basic features of the acquired data in a study along with providing summaries of sample and the measures. Frequencies, cross tabulation, descriptive radio statistics come under this.

Bivariate Statistics: It is a form of quantitative statistical analysis that determines the empirical relationship between two variables (often denoted as X, Y). Some of its tools are ANOVA, means, correlation, nonparametric tests.

Other than the above-mentioned statistical methods that can be leveraged in SPSS, there is a prediction for identifying groups, including cluster analysis (K-means, two-step, hierarchical), factor analysis, and Numeral outcome prediction such as linear regression.

2.4.2.2 How SPSS Helps in Research & Data Analysis Programs:

SPSS is a revolutionary software mainly used by research scientists which helps them process critical data in simple steps. Working on data is a complex and time-consuming process, but this software can easily handle and operate information with the help of some techniques. These techniques are used to analyze, transform, and produce a characteristic pattern between different data variables. In addition to it, the output can be obtained through graphical representation so that a user can easily understand the result.

2.4.2.3 SPSS techniques for data processing

SPSS techniques that are responsible in the process for data handling and its execution.

1. Data Transformation: This technique is used to convert the format of the data. After changing the data type, it integrates the same type of data in one place and it becomes easy to manage it. You can insert different kinds of data into SPSS and it will change its structure as per the system specification and requirement. It means that even if you change the operating system, SPSS can still work on old data.

2. Regression Analysis: It is used to understand the relationship between dependent and interdependent variables that are stored in a data file. It also explains how a change in the value of an interdependent variable can affect the dependent data. The primary need of regression analysis is to understand the type of relationship between different variables.

3. ANOVA (Analysis of variance): It is a statistical approach to compare events, groups, or processes, and find out the difference between them. It can help you understand

which method is more suitable for executing a task. By looking at the result, you can find the feasibility and effectiveness of the particular method.

4. MANOVA (Multivariate analysis of variance): This method is used to compare data of random variables whose value is unknown. MANOVA technique can also be used to analyze different types of populations and what factors can affect their choices.

5. T-tests: It is used to understand the difference between two sample types, and researchers apply this method to find out the difference in the interest of two kinds of groups. This test can also understand if the produced output is meaningless or useful.

2.4.2.4 How to do Fisher's exact test, Relative Risk, and Odds Ratio test in SPSS

Fisher's Exact Test: This test tends to be used when sample sizes are small, and you want to test whether two categorical variables (for example, gender and smoking status) are associated with each other. Steps:

1. Click on Analyze -> Descriptive Statistics -> Crosstabs
2. Drag and drop (at least) one variable into the Row(s) box, and (at least) one into the Column(s) box
3. Click on Statistics, select Chi-square, and then click on Continue
4. Click on Exact, and then select the Exact option, leaving the test time limit as it is
5. Press Continue, and then OK to run the test
6. The result will appear in the SPSS output viewer

Relative Risk, Odds Ratio Test in SPSS:

1. Click on Analyze -> Descriptive Statistics -> Crosstabs
2. Drag and drop (at least) one variable into the Row(s) box, and (at least) one into the Column(s) box
3. Click on Statistics, select Chi-square, and check the Risk box in the then click on Continue

4. Press Continue, and then OK to run the test
5. The result will appear in the SPSS output viewer

Chapter 3

3 Methodology

This section details the phases of the preparation and the realization of the designed experiment.

3.1 Experimental planning

The *goal* of this experimental evaluating is to study the effectiveness of the control flow technique with the opaque predicates to increase the difficulty of reverse-engineering the software by human attacks. The main *objective* is to estimate how the obfuscation prevents or limits the ability of the attacker to alter or perform modification and understand the source code. Our work is based on [27] with some different techniques as summarized in Table 3.1.

Table 3.1: Our work VS based work

| Our work | Based work |
|------------------------------------|--------------------------------|
| Control flow obfuscation technique | Renaming obfuscation technique |
| C# source code | Java source code |
| Developers pretend as subjects | Graduated subjects |
| Two c# application as objects | Two java application |

The *quality focus* regards how the obfuscation reduces the attacker capability to understand and modify the source code, and how the obfuscation increases the effort needed to successfully complete an attack.

The study definition is summarized in Table 3.2.

Table 3.2: Summarization of the experiment

| | |
|----------------------------|---|
| goal | Study the effectiveness of the control flow technique with the opaque predicates to increase the difficulty of reverse-engineering the software by human attacks. |
| Quality focus | Ability of understanding the obfuscated code. Ability to perform attacks on the obfuscated code. |
| Context | Objects: two C# Application. (Login.exe and Activation.exe) Subjects: Developers whom familiar with coding and developing windows and web application. |
| Main factor | Obfuscation with (control flow with opaque predicates) Obfuscated code VS clear code. |
| Other factors | Ability, system, subjects. |
| Dependent variables | 1. Ability to perform understanding tasks. 2. Time required for understanding. 3. Ability to correctly perform a change task. 4. Time required performing change task. |

The *context* of this experiment study consists of *subjects* who are pretending as attackers and *objects* which are the applications to be attacked. The *subjects* are programmers working at Smart Vision Company for software development at Hadhramout governorate in Yemen. These developers are familiar with coding and developing web and desktop applications, the total number of subjects and the distribution of the tasks are shown in Table 3.3.

Table 3.3: total number of subjects and the distribution of the tasks

| Total subjects | Total tasks | Understanding Tasks | Modification tasks |
|----------------|-------------|---------------------|--------------------|
| 14 | 44 | 22 | 22 |

The *objects* in the experiment are two .exe files developed using C# programming language, where the control flow obfuscation technique is based on the Algorithm proposed at [28]; the two applications are outlined as follows:

The **first application** is as shown in Fig 3.1. This application simulates the software activation process. It asks the user to enter the registration number, After the user enters the number in the text box and clicks the activation button, the program checks that number if the activation number is correct the program displays a message that informs the user that the activation is successful, and vice versa in case of wrong activation number.

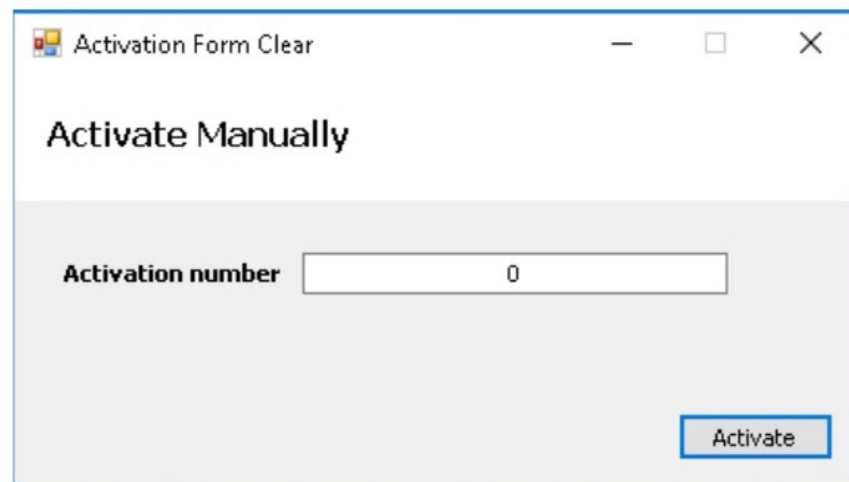


Fig. 3. 1. Activation screenshot

The **second application** is shown in Fig 3.2. This application simulates the software login process. When the user runs the application, the application checks the availability of Internet service, and then it accesses its company web site to check for any available updates. Then the application displays the login screen. The user enters the login credentials, a user name, and password and clicks on the login button after clicking by the

user. In the case of the availability of new updates, the program displays a message to the user asking to update the program and opens the link of the new version on the browser.

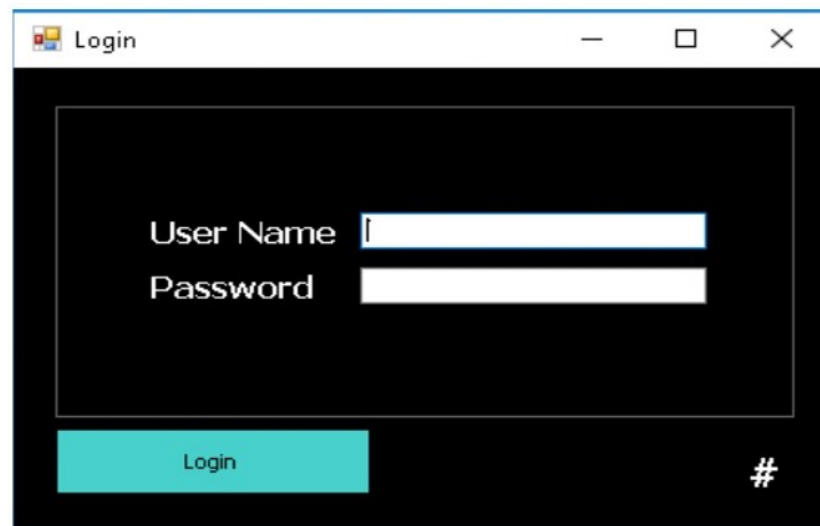


Fig. 3. 2. Login screenshot

3.2 Research questions

The research questions in this study are formulated as follows:

1. To what degree the capability of subjects to understand source code is reduced when the control flow obfuscation with opaque predicate is used?
2. To which level the control flow obfuscation increment the time required to perform an understanding task?
3. What is the amount of reduction caused by the control flow technique and opaque predicate on the ability of subjects to modify or alter the source code?
4. To which level the control flow obfuscation and opaque predicate increment the required time to modify or alter the source code?

3.3 Null hypotheses

Based on the above research questions we formulated the null hypotheses to be examined in our experiment.

1. **H0.** The control flow obfuscation does not significantly reduce the capability of subjects to perform code understanding tasks correctly.
2. **H1.** The control flow obfuscation does not significantly increment the time required to perform understanding tasks.
3. **H2.** The control flow obfuscation does not reduce the ability of subjects to perform a modification task correctly.
4. **H3.** The control flow obfuscation does not significantly increment the time required to perform modification tasks.

As we are planning to investigate the effect of control flow techniques on one side, it is noted that the above two null hypotheses are one-tailed. There are two dependent variables: understanding level and success of the modification task.

For measuring the attacker's understanding level, subjects had been requested to read the source code and execute two understanding assignments:

- What is the correct registration number that makes the program shows “activated” message dialog?
- When a user enters a correct registration number and activation message is displayed, a compare process takes place between the value of the text box and the integer variable. What is the name of that integer variable and what are the values of the increments that took place to that variable?

In order to examine the capability of the attacker to alter the source code, subjects were requested to execute two modification tasks:

- Modify the application such that the application does not check the internet availability.
- When users log in the application display a link of the updated version for the user to be downloaded, modify the link to visit the Google web site.

The correctness of the attack was evaluated by testing the changed code. Subjects have to give an answer to every task. Each correct answer is assigned one, while a wrong answer as zero.

For this experiment, the obfuscation (control flow with opaque predicates) is considered as the independent variable. In our case, there are two treatments, for example, performing understanding and also modification on the obfuscated copy and also on the original one of source code. So we introduced two copies of every application, one with clear code and the other is obfuscated with a control flow obfuscation technique.

3.4 Experiment material and procedure

Before the experiment starts, subjects were prepared and trained about obfuscation and performing understanding and modification tasks on the obfuscated and clearing source code. Subjects were provided with details and explanations on every task to be executed through the experiment. Subjects used a desktop computer with Microsoft visual studio development environment and dot net Reflector and Reflexil, which are the tools for debugging and editing source code.

The following materials were distributed to subjects:

1. A short textual description of the program (login and activation examples by C#) they have to attack, which contains run instructions.
2. Two .exe programs (login and activation examples by C#) one is clear and the other is obfuscated.
3. Explanation slides of the overall procedure of the experiment.

The experiment has been done according to the following procedure:-

- 1) Read the program description.
- 2) Run the program to be familiar with it.
- 3) For performing each task :
 - i. Ask the supervisor of the sheet that describes the task to be executed.
 - ii. Note and write down the time to start.
 - iii. Execute the task.
 - iv. For the understanding tasks, an answer must be provided.
 - v. Note the end time and submit the sheet to the supervisor.
- 4) Finally, submit the maintained or the attacked program to the supervisor.

3.5 Analysis method

To analyze the obtained results, statistical tests have been used. To investigate the effect of the obfuscation technique used on the correctness of the understanding and the modification tasks, we have to utilize tests that apply to categorical data where performing the task may result in a correct answer or wrong one. For this purpose, two kinds of measurements were used. Firstly, measure the significant difference; secondly, calculate the effect size between the different treatments of the experiment.

3.5.1 Measurement of the significant difference

In statistical hypothesis testing, a significant difference is used to justify whether the null hypothesis should be rejected or confirmed.

To analyze the significant difference between the two different treatments (clear versus obfuscation), we use Fisher's exact test [29] which is a statistical significance test used to examine the significance of the association between two kinds of classifications or differences in categorical data. In statistical difference, many alternative tests could be used, but the Fisher test is used especially in cases of small samples.

To analyze the significant difference between the amounts of time spent by subjects for the case of modification or understanding tasks we use The Mann-Whitney U test [30]. This test is mostly used to compare differences between two independent groups when the dependent variable is either ordinal or continuous, but not normally distributed.

A Mann-Whitney is a one-tailed test used because it is more powerful than a two-tailed test, as we aren't considering the effect in the opposite direction. We compare the p-value to the significance level (denoted as α or alpha) of 0.05. When $P\text{-value} \leq \alpha$: The

difference between the medians is statistically significant, while if $P\text{-value} > \alpha$: The difference between the medians is not statistically significant.

3.5.2 The measure of the effect size

In statistics, the size of an effect is a simple way of quantifying the difference between the two groups. It has many advantages over the use of tests of statistical significance alone. It focuses on the size of the difference rather than focusing on sample size [31].

This method is more suitable because it is used to provide information about the magnitude of a difference since we want to measure the extent to which the presence of obfuscation decreases the capability of the reverse engineering to successfully complete a source code attack.

Relative risk test is used in the statistical analysis of the data of experimental studies, to estimate the strength of the association between treatments or risk factors. It is often used in cases where the study involves comparing the likelihood, or chance, of an event occurring between two groups. The Relative Risk uses the probability of occurrence of an event in one group compared to the probability of an event occurring in the other group. It requires the examination of two dichotomous variables, where one variable measures the event (occurred vs. not occurred) and the other variable measures the groups (group 1 vs. group 2). Values of RR can be interpreted as follows:

$$\begin{cases} RR = 1, \text{ means that the exposure does not have effect on the outcome.} \\ RR < 1, \text{ means that the risk of the outcome is decreased by the exposure.} \\ RR > 1, \text{ means that the risk of the outcome is increased by the exposure.} \end{cases}$$

Chapter 4

4 Results

This section presents the results of this experimental study that was performed at a smart vision company for software development at Hadhramout governorate in Yemen with 14 subjects who are familiar with web and desktop application development. The experiment aims to assess the control flow obfuscation technique.

4.1 Task results

Table 4.1 reports the correct and wrong tasks for understanding and modification tasks of the experiment.

TABLE 4.1. CORRECT AND WRONG TASKS

| Treatment | Understanding | | Modification | | Total | |
|-----------|---------------|----------------|--------------|----------------|--------------|----------------|
| | <i>Wrong</i> | <i>Correct</i> | <i>Wrong</i> | <i>Correct</i> | <i>Wrong</i> | <i>Correct</i> |
| Clear | 1 | 10 | 2 | 9 | 3 | 19 |
| Obfuscate | 7 | 4 | 5 | 6 | 12 | 10 |

4.2 Significant difference results

We examined the availability of a significant difference between the obfuscated and clear code using the Fisher exact test, for the understanding tasks the significant difference is available where P-value is equal to 0.0237 so this is evidence to reject the first null hypothesis H0.

For the modification tasks no significant difference, where the P-value is equal to 0.3615, so we accept the third null hypothesis H2. Also for the sum of overall tasks for understanding and modification a difference is available because the P-value is equal to 0.0097.

Fig. 4.1 shows boxplots of the average time spent by subjects to complete the tasks. Time is computed, for each subject, as the average time over understanding and modification tasks. The Mann-Whitney test showed a significant difference for the time of understanding tasks where P-value equal to 0.001, also for modification tasks time a significant difference was found where P-value equal to 0.0031, so these results suggest a rejection for the null hypothesis H1 and H3.

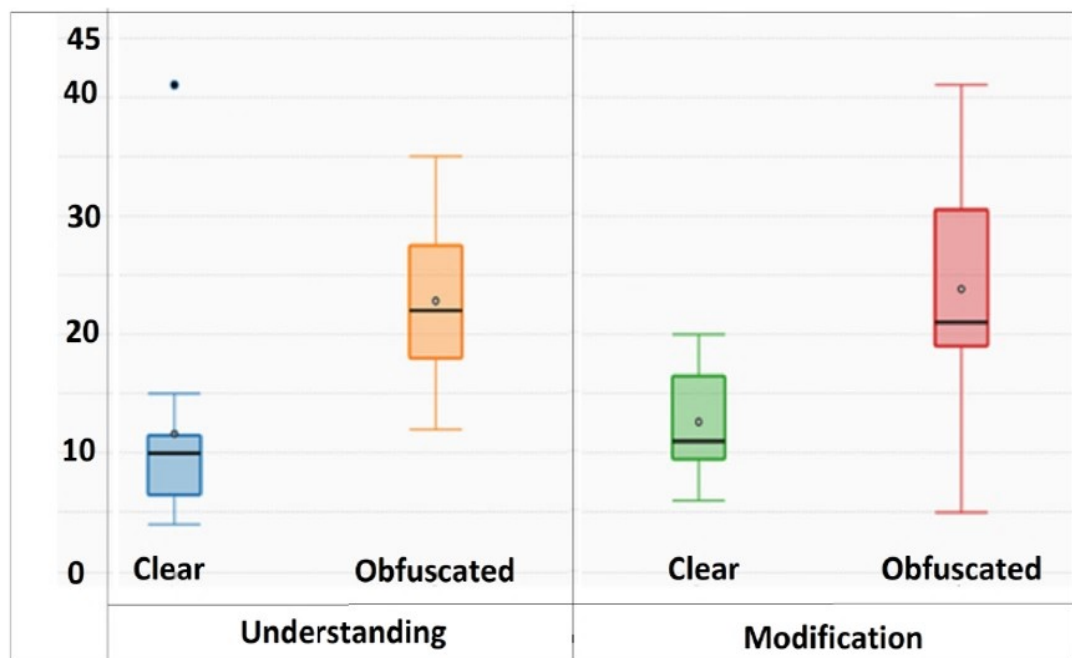


Fig 4.1 Boxplots of the average time for tasks execution

4.3 Effect size results

After calculating the availability of difference between the two treatments, we are interested in that is estimating the strength of the association or the magnitude of the difference between the clear and the obfuscated source code. For that reason, the relative risk effect size test has been used.

The results of relative risk for understanding tasks are $RR= 7$ which means the presence of obfuscation on source code increases seven times the difficulties for the attacker to successfully complete the understanding task.

For the modification task, the $RR=2.5$ which means the obfuscation on source code increases more than double times the difficulties for the attacker to successfully complete the attack,

The overall tasks for both understanding and modification the $RR= 4$.

4.4 Compare the obtained results with previous studies

When comparing our results with previous study, we find that the results are close to some extent with an inverse and very logical difference according to the nature of the obfuscation technique used in each study.

The following table 4.1 shows a comparison between the results of our study and a similar previous study.

Table 4.2: Comparison with previous study.

| The results of the study at [38] | Our Results |
|---|--|
| Obfuscation Technique: Renaming | Obfuscation Technique: Control Flow |
| <u>Availability of Significant Difference:</u> <ul style="list-style-type: none"> • Statistical Test: Fisher Exact test. • For Understanding Tasks: no significant difference was found (p-value=0.33). • For modification Tasks: a significant difference was found (p-value=0.009). | <u>Availability of Significant Difference:</u> <ul style="list-style-type: none"> • Statistical Test: Fisher Exact test. • For Understanding Tasks: a significant difference was found (p-value= 0.023). • For modification Tasks: no significant difference was found (p-value= 0.36). |
| <u>Availability of Magnitude Difference:</u> <ul style="list-style-type: none"> • Statistical Test: Odds Ratio. • For Understanding Tasks: the odds ratio is 2.3, subjects having obfuscated code have less than half of subject having clear code to successfully complete the tasks. • For modification Tasks: the odds ratio is (7.1) Obfuscation reduces seven times the odds of completing the attack. | <u>Availability of Magnitude Difference:</u> <ul style="list-style-type: none"> • Statistical Test: Relative risk. • For Understanding Tasks: the RR= 7, means that the obfuscation increases seven times the difficulties for the attacker to complete the understanding attack. • For modification Tasks: the RR=2.5, means that the obfuscation increases more than double times the difficulties for the attacker to complete the Modification attack. |
| <u>Availability of Time Difference:</u> <ul style="list-style-type: none"> • Statistical Test: Mann-Whitney. • For Understanding Tasks: a significant difference was found (p-value=0.002). • For modification Tasks: no significant difference was found (p-value=0.19). | <u>Availability of Time Difference:</u> <ul style="list-style-type: none"> • Statistical Test: Mann-Whitney. • For Understanding Tasks: a significant difference was found (p-value= 0.001). • For modification Tasks: a significant difference was found (p-value=0.003). |

Chapter 5

5 Conclusion and Future Work

In this chapter, the conclusions of the study are presented, and an outline of future work ideas based on this thesis is also given.

5.1 Conclusion

Code obfuscation is a useful technique to protect software against reverse engineering attacks; Most of the obfuscation techniques are not based on well-defined measurements to clarify their effectiveness. In this thesis, we presented an experiment aimed at assessing control flow code obfuscation technique with the opaque predicates, in terms of the capability to understand and modify obfuscated codes.

The experiment involved 14 programmers. As a result of the statistical analysis used in this thesis, it is concluded that the presence of obfuscation on source code increases seven times the difficulties for the attacker to successfully complete the understanding task. Also the control flow obfuscation significantly decreases the capability of the attackers to correctly perform the understanding tasks while no significant difference for modification. Also the presence of obfuscation on source code increases the amount of time needs for subjects to understand and modify the source code.

The results of this study will help the software developers to choose the most appropriate and best obfuscation technique in order to protect their applications. This study will help researchers to find out - through the experimental approach - how obfuscation techniques are powerful and efficient in software protection.

5.2 Future Work

Based on the research methodology presented in this thesis, the following research topics can be pursued:

- Performing other experiments with different settings and contexts and involving larger sets of subjects and different other source code obfuscation techniques.
- Applying more accurate statistical tests for analyzing the results.
- Assessment of the combined effect of different obfuscation techniques and performing a comparative assessment of their effects.

References

- [1] Flemming Nielson, Hanne R Nielson, and Chris Hankin. Principles of program analysis. Springer, 2015.
- [2] David Aucsmith. Tamper resistant software: An implementation. In Proceedings of the First International Workshop on Information Hiding, pages 317–333, London, UK, 1996. Springer-Verlag.
- [3] Christian Collberg and Jasvir Nagra. Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection. Addison-Wesley, 2009.
- [4] H. Xu, Y. Zhou, Y. Kang, M. R. Lyu , On Secure and Usable Program Obfuscation: A Survey, Dept. of Computer Science, The Chinese University of Hong Kong ,2017.
- [5] M. Kapoor , S. A. Sebastian, , P. Shah, T. Parekhji , A Review on Code Obfuscation, World Conference on Futuristic Trends in Research and Innovation, 2016.
- [6] C. K. Behera , D. L. Bhaskari , Different Obfuscation Techniques for Code Protection , ScienceDirect , 4thInternational Conference on Eco-friendly Computing and Communication Systems , 2015.
- [7] B. Barak , O. Goldreich , R. Impagliazzo , S. Rudich , A. Sahai , S. Vadhan , K. Yang , On the (Im) possibility of Obfuscating Programs , Journal of the ACM, 2012
- [8] S. A. Sebastian, S. Malgaonkar, P. Shah, M. Kapoor, and T. Parekhji, “A study & review on code obfuscation,” 2016 World Conference on Futuristic Trends in Research and Innovation for Social Welfare (Startup Conclave), 2016.
- [9] Z. Liang, W. Li, J. Guo, D. Qi, and J. Zeng, “A parameterized flattening control flow based obfuscation algorithm with opaque predicate for reduplicate obfuscation,” International Conference on Progress in Informatics and Computing, 2017.

- [10] V. Balachandran, DJJ Tan, Control flow obfuscation for android applications, Computers & Security, 2016.
- [11] D Xu, J Ming, D Wu, Generalized dynamic opaque predicates: A new control flow obfuscation method, International Conference on Information Security, 2016.
- [12] Y Peng, G Su, B Tian, M Sun, Q Li, Control flow obfuscation based protection method for Android applications, China Communications, 2017.
- [13] Y Wang, S Wang, P Wang, D Wu , Turing obfuscation , College of Information Sciences and Technology, The Pennsylvania State University, 2019 .
- [14] L Zhang, H Meng, VLL , Progressive control flow obfuscation for android applications Thing, TENCON IEEE conference ,2018.
- [15] Y Wang, Y Shen, K Cheng, Y Yang , Poster: Obfuscating Program Control Flow with Intel SGX m 2018 IEEE/ACM 40th ,2018.
- [16] K. Z. Ming , Control flow obfuscation via CPS transformation, School of Information Technology, Singapore , 2019.
- [17] C. Collberg, C. Thomborson, and D. Low, “A taxonomy of obfuscating transformations. Technical report,” Department of Computer Science, The University of Auckland, New Zealand, 1997.
- [18] M. Ceccato, A. Capiluppi, P. Falcarin, and C. Boldyreff, “A large study on the effect of code obfuscation on the quality of java code,” Empirical Software Engineering, pages 1–39, 2014.
- [19] R. Mohsen, “Quantitative Measures for Code Obfuscation Security Imperial,” Imperial College London, Department of Computing, 2016.

- [20] A. Pawlowski, M. Contag, and T. HolzHorst, "Probfuscation: An Obfuscation Approach using Probabilistic Control Flows," Gortz Institute (HGI), Ruhr-University Bochum, Germany, 2016.
- [21] An Empirical Evaluation of Software Obfuscation Techniques Applied to Android APKs 28 Effects of Code Obfuscation on Android App Similarity Analysis
- [22] J. Park, H. Kim, Y. Jeong, S.Cho, S. Han, M. Park, Effects of Code Obfuscation on Android App Similarity Analysis, Dankook University, Yongin, Korea,2019.
- [23] M. Ceccato, M. D. Penta, J. Nagra, P. Falcarin, F. Ricca, M. Torchiano, and P. Tonella, "The effectiveness of source code obfuscation: An experimental assessment," 2009 IEEE 17th International Conference on Program Comprehension, 2009.
- [24] A. Viticchie, L. Regano, M. Torchiano, C. Basile, M. Ceccato, P. Tonella, and R. Tiella, "Assessment of Source Code Obfuscation Techniques," 2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation, 2016.
- [25] M. Ott and L. Held, "Bayesian Calibration of p-Values from Fishers Exact Test," International Statistical Review, vol. 87, no. 2, pp. 285–305, Apr, 2018.
- [26] D. Werkke, N. Huaman, Y. Acer, B. Reaves, P. Traynor, S. Fabel, "A Large Scale Investigation Of Obfuscation Use In Google Play," Annual Computer Security Applications Conference, 2018.
- [27] M. Ceccato, M. Di Penta, J. Nagra, P. Falcarin, F. Ricca, M. Torchiano, P. Tonella , "Assessment of Source Code Obfuscation Techniques", 2017
- [28] Z. Liang, W. Li, J. Guo, D. Qi, and J. Zeng, "A parameterized flattening control flow based obfuscation algorithm with opaque predicate for reduplicate obfuscation," International Conference on Progress in Informatics and Computing (PIC), 2017.

- [29] M. Ott and L. Held, “Bayesian Calibration of p-Values from Fishers Exact Test,” *International Statistical Review*, vol. 87, no. 2, pp. 285–305, Apr, 2018.
- [30] T. W. Macfarlan, J. M. Yates, “Mann-Witeny U-test, ” *Introduction to Nonparametric Statistics for the Biological Sciences Using R* (pp.103-132), 2016.
- [31] A. Bakker, J. Cai, L. English, G. Kaiser, V. Mesa, and W. V. Dooren, “Beyond small, medium, or large: points of consideration when interpreting effect sizes,” *Educational Studies in Mathematics*, vol. 102, no. 1, pp. 1–8, 2019.
- [32] Chikofsky, E. J.; Cross, J. H. (January 1990). Reverse engineering and design recovery: A taxonomy. *IEEE Software*.
- [33] Warden, R. (1992). *Software Reuse and Reverse Engineering in Practice*. London, England: Chapman & Hall. pp. 283–305.
- [34] Shahbaz, Muzammil (2012). *Reverse Engineering and Testing of Black-Box Software Components: by Grammatical Inference techniques*. LAP LAMBERT Academic Publishing.
- [35] Schwarze, K., Schwarze, K., & Schwarze, K. *Reverse Engineering Tutorial: How to Reverse Engineer Any Software*. Retrieved from <https://www.udemy.com/blog/reverse-engineering-tutorial> . June 6, 2014.
- [36] Tdykstra, microsoft. Ildasm.exe (IL Disassembler). Retrieved from <https://docs.microsoft.com/en-us/dotnet/framework/tools/ildasm-exe-il-disassembler>.
- [37] Kexugit. Ten Must-Have .NET Tools Every Developer Should Download Now. Retrieved from <http://msdn.microsoft.com/en-us/magazine/cc300497.aspx>
- [38] M. Ceccato, M. Di Penta, J. Nagra, *Towards Experimental Evaluation of Code Obfuscation Techniques*, 2017.

List of Publications

- 1- Mohammd. H. BinShamlan, Mohammed. A. Bamatraf and A. A. Zain, "The Impact of Control Flow Obfuscation Technique on Software Protection Against Human Attacks," 2019 First International Conference of Intelligent Computing and Engineering (ICOICE), doi: 10.1109/ICOICE48418.2019.9035187, URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9035187&isnumber=9035128> .
- 2- Mohammd. H. BinShamlan, Mohammed. A. Bamatraf and A. A. Zain, “(2021) Experimental Evaluation of the Obfuscation Techniques Against Reverse Engineering”, Advances on Smart and Soft Computing. Advances in Intelligent Systems and Computing, . Springer, , URL: https://link.springer.com/chapter/10.1007%2F978-981-15-6048-4_33