Data Set Security in a Physically Accessible System

Author

Muhammad Abdullah Abid

NUST201464150MSEECS63114F

Supervisor

DR. SHAHZAD SALEEM

A thesis submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SECURITY (MS-IS)

DEPARTMENT OF COMPUTING (DoC)

SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE (SEECS)

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY (NUST)

ISLAMABAD, PAKISTAN

(July 2018)

Declaration

I certify that this research work titled "*Data Set Security in a Physically Accessible System"* is my own work. The work has not been presented elsewhere for assessment. The material that has been used from other sources it has been properly acknowledged/referred.

Signature of Student

Muhammad Abdullah Abid

NUST201464150MSEECS63114F

Approval

It is certified that the contents and form of the thesis entitled “Data Set Security in a Physically Accessible System” submitted by M Abdullah Abid have been found satisfactory for the requirement of the degree.

Advisor: Dr. Shahzad Saleem

Signature: ______________________
Date: ______________________

External Supervisor: Dr. Mudassar Aslam

Signature: __________________________
Date: __________________________

Committee Member 1: Mr. Ubaid Ur Rehman

Signature: __________________________
Date: __________________________

Committee Member 2: Miss Hirra Anwar

Signature: __________________________
Date: __________________________

THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS thesis written by Mr. Muhammad Abdullah Abid, (Registration No NUST201464150MSEECS63114F), of SEECS (School/College/Institute), has been vetted by undersigned, found complete in all respects as per NUST Statutes/Regulations, is free of plagiarism, errors and mistakes and is accepted as partial fulfillment for award of MS/M Phil degree. It is further certified that necessary amendments as pointed out by GEC members of the scholar have also been incorporated in the said thesis.

Signature: ________________________________

Name of Supervisor: Dr. Shahzad Saleem

Date: ___________________________________

Signature (HOD): __________________________

Date: ___________________________________

Signature (Dean/Principal): __________________

Date: ___________________________________

Copyright Statement

Certificate of Originality

I hereby declare that this submission titled “Data Set Security in a Physically Accessible System” is my own work. To the best of my knowledge, it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST SEECS or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST SEECS or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project’s design and conception or in style, presentation and linguistics which has been acknowledged. I also verified the originality of contents through plagiarism software.

Author Name: M Abdullah Abid

Signature: _________________

Acknowledgments

I am very thankful to my Allah Almighty for all his help and blessings in every stage of my life.

I am also thankful to my parents, my wife and to my siblings for supporting and encouraging me throughout my life.

I would like to give special thanks to my respected sir and supervisor Dr. Shahzad Saleem for his help throughout my thesis. Besides my supervisor, I am also very thankful to Dr. Naveed Ahmed for his guidance and support throughout the research period. GEC Committee members: Mr. Ubaid Ur Rehman and Miss Ayesha Kanwal were very supportive too. So, I am thankful to all of them for their efforts, encouragement, and support to overcome numerous obstacles I have been facing throughout my research.

In addition, I would also like to thanks my fellow Mr. Zeeshan Qaiser and Mr. Umer Hayat and all other friends for their virtuous support and cooperation.

Finally, I would like to express my gratitude to all the individuals who have rendered valuable assistance to my study.

This work is dedicated to

my adorable daughters

Eshaal and Aanabiya

Abstract

The field of software engineering nowadays is of high importance due to its financial and daily life implications. So attackers find them advantageous to analyze the executables dispatched by software creating companies. In the last two decades, reverse engineering tools are increased by many folds and also their effectiveness and diversity has expanded. Due to these competent tools, the number of attacks by the malicious users have increased. Software has two types of information namely functions and data. To be secure against attacks there are many solutions and obfuscation is one of the techniques. Most of the obfuscation techniques are developed for the obfuscation of functions. For the security purpose, cryptographic techniques and encryption keys are embedded in the software executables as data. Using reverse engineering tools attacker may extract the data embedded in the executable for financial benefit. In the past different data and key storage technologies which includes hardware-based and software-based techniques. The software-based solution includes encryption, virtualization, hiding keys within the executables etc. Whereas hardware techniques involved usage of TPM chip, creating of binary state machines etc.

In this research, a software-based technique is developed for the obfuscation of data present in the executable. An application is developed that take a programming file as an input. The application extracts the constants declared in the programming file and transforms that constants into randomized code. The application then creates a new programming file, which includes the old code with some modifications, randomized code generated against the data or key, de-obfuscator function, and some supporting functions. This programming file is the output of the obfuscation application. In the end, multiple tools were used to evaluate the effectiveness of the obfuscation process designed. The evaluation done follows the techniques of static analysis. It was found that it has become harder for an attacker to extract any useful information from the executables.

Key Words: *Key Security, Obfuscation, Cryptography, Key Hiding, Open System Security, Software Security, Secure Executables*

Table of Contents

List of Figures

List of Tables

List of Acronyms

CD	Compact Disk
DVD	Digital versatile disk
COTS	Commercial off the shelf
STB	Setup Box
SSL	Secure Socket Layer
HMAC	Hash Message authentication code
RAM	Random Access Memory
FROST	Forensics recovery of scrambled telephones
NLFSR	Non-Linear Feedback Shift Register
IDS	Intrusion Detection System
SK	Secure Key
CUT1	Code Under Test – Sample 1
CUT2	Code Under Test – Sample 2
OCO1	Obfuscated Code Obtained against Sample 1
OCO2	Obfuscated Code Obtained against Sample 2

CHAPTER 1

1 INTRODUCTION

1.1 Motivation

We are living in the era of computing and machines. Most of the utilities and entertainment functions that are available online or sent to end users are in the form of software executables. Sometimes these executables are dispatched on the optical disk drives in form of CD or DVD, can be shared online on a website. Software types range from games, documentation utilities, simulation software, personal assistant and many more.

Software distributed can be in one of the following categories i.e. Custom Software, COTS (commercial off the shelf) or open-source software. Open-source software doesn't have the issue of privacy but Custom or COTS software implements the algorithm that is designed by the organization itself or by the third party and they want end users to use the services provided by the software but keep the algorithm secret [1]. These algorithms and the data associated with the software like serials, product keys, and cryptographic keys are vital to the organization. Profitability of that organization is directly dependent on the secrecy of the software they are creating. Either they will be selling that software or using that software within the organization for their business process improvement [2]. If someone plagiarizes or modifies the software, it will dent/decrease the earning patterns of the actual owner organization by making illegal copies of that software. This can be visualized by looking at the number of tempered software and games present online without the consent of the owners.

1.2 Categorization of Systems and their Security Needs

Usually, Systems in terms of accessibility are categorized into Open Systems and Closed Systems. Now a day most of the systems can be categorized under Open Systems. Usually, these open systems contain the software executables which contains Intellectual Property in the form of data, that can be easily accessible by the attackers. Even though sometimes legal customers can be tempted to attack the systems like STB to gain financial benefits [3]. So, in short getting access to a system and applying the reverse engineering techniques on the executable is not difficult now as was in the past. Irrespective of the system designated as an open system or closed, one can extract or analyze the executable and systems hardware.

Embedded systems are the type of machines that are gaining popularity. These type of machines are used in home automation, the textile industry, automobiles, avionics, customized applications like STB and many more [4]. In these type of machines there is not separate software part associated to them but come preinstalled and usually, it cannot be altered. In such machines, all the necessary data that is categorized as Intellectual property are made part of such machines/hardware. Hence anyone with the expertise can get the access to these type of machines, they can analyze and eventually can extract all or very useful chunk of information. There is a research in which it is shown that subsystems of automobiles like the brake subsystem can be accessed using wireless devices like media players [5]. IoT [6] is one of the hot topics in the research domain. If there are N number of devices, then accessing any one of them may lead to a system having administrative controls. So looking at the important roles of the embedded systems, security is very important to these systems [6]. In the past, some solutions were discussed by some of the researchers but this issue remains alive for the researchers, due to the usage of the latest technologies by the attackers. So the software part of these embedded systems needed to be secured to give them an extra layer of security.

1.3 Cryptography and its usage in security implementations

In a number of Government or strategic organizations, where there is a need for secure and critical computations, these organizations then move to take help from the cryptographic mechanism. The list of useful cryptographic mechanisms may include Digital Signatures, HMAC, Symmetric Encryptions, Asymmetric Encryptions and many more [7]. These mechanisms are helpful in implementing a better security on a single system as well as on the data moving over the network. Usually, OpenSSL or similar libraries are used for the above-mentioned requirements. From the attacker’s point of view, it’s way harder to attack the algorithm as these algorithms are international standards and have been tested many times. Another important part of the cryptosystem is the security keys [8]. As these are randomly selected or the user can select this key, so there can be a vulnerability associated with the key. So now attacker will attack the set of keys associated with the cryptographic mechanisms and if the key is compromised, everything related to cryptosystem is compromised as well[8].

Cryptographic keys are one of the main tools that are used to secure multiple Intellectual Property based contents present in the executable. With the advanced needs of security, different cryptographic algorithms are implemented in the software especially encryption. Security of any encryption algorithm is totally dependent on the security of the key and its distribution according to Kirchhoff's Rule [8]. So, sensing the need for securing cryptographic keys, White Box Cryptography is the technique that is designed for such need [9]. This technique was created such that for any person that has the access to an open system or software executable, even then make it harder for the attacker to extract cryptographic keys present in the software executable.

1.4 Software Implementation and Security

Software implementation can be broken down into two parts, one is data and the other is functions and both must be secured. Algorithm flow is usually coded as the functions, or services assigned to objects are coded as functions [10]. Data may include definitions of a lot of different data types. Not all but some of the definitions are much important for the secrecy of the software that may include many types of cryptographic keys, serials, product keys and in the form of constants in the software executables. For any designed software product, there is a need to keep the algorithm secret as well as all of the other contents related to Intellectual Property [11]. There are multiple ways for hiding each of these integral parts of the software i.e. data and functions in the executable or systems. These include obfuscation, software encryption techniques, key hiding using virtualization, use of binary state machines for storing keys etc.

1.5 Reverse engineering helping the attackers

With the advancements done in the field of reverse engineering, there are a lot of tools available online that can reverse engineer the executable binaries for analysis. Some of the top-rated tools include *IDA Pro, Ollydbg*, gdb, Nasm and many more that are available online. Some Linux based utilities are also available for the analysis purpose, they include but not limited to *Strings, strip, Idd, nm* and other such commands. Using any of the tools, one can get information about the implemented algorithm in the form of functions and Intellectual Property present in the form of data [12] [13].

Due to the availability of multiple methods, tool, and techniques, it’s hard to describe a unanimous method of reverse engineering any executable to extract anything useful. Usually, attackers get access to the executables in the form of binaries. Attackers then converge their attacking ideas to one or more productive components of executables. Next step usually applied is to convert this given binary to assembly code, so that attacker may analyze it further. Assembly is far better to understand the logic implemented or to extract anything useful. The software/tools used for the purpose are called disassemblers. An attacker cannot get all of the information about the implementation language or exact 1 to 1 algorithm implemented but the attacker can get to know about its working. There are many disassemblers that can be used, some of the tools that can be used are IDA Pro, Hopper, x64dbg, Immunity Debugger and many more that are available online. Once an attacker gets assembly code, attacker selects some parts of the code to be analyzed in depth. As the analysis of the whole assembly code is time-consuming and it’s hard for the attacker to analyze all the assembly code. Hence attacker may select only some parts of the assembly code to analyze that looks like the code more relevant to the goals finalized. An attacker cannot be stopped to convert binary to respective assembly code, but we can make it harder for the attacker to analyze the assembly by applying obfuscation techniques. So that the time taken to analyze the binary will increase. Better the obfuscation technique, the bigger the time taken to analyze the binary [13] [14].

1.6 Attacks and their Security Counter-measures

Over the years, many techniques are designed to hide data and functions in an executable. Obfuscation is one of the many techniques that can be used. Code obfuscation [15] [13] [14] changes the physical appearance of code and keeping the functionality intact. Due to obfuscation, it becomes hard for the attacker to reverse engineer the software. It is a useful technique for protecting the Intellectual Property. There are many types of obfuscations that are designed, these types can be evaluated using the parameters like cost, stealth, potency, resilience. Other technique that can be used, is encrypting software code. it is one of the solutions that is used to hide the implementation of both data and functions [16]. But for execution, the software has to be decrypted. As a result, software in the unencrypted form is loaded into the memory (RAM). Then, an attacker can take an image of RAM or can extract the information from process memory dump to get access to decrypted information related to executable [17] [18]. This concept is used in many attacks like Cold bot attack. Cold bot attack is a type of side channel attack in which emphasis is on the weak implementation rather than on the algorithm itself. FROST [19] is one of the tools that is used in the domain of mobile forensics and uses that concept of cold bot attacks. It recovers the important set of information from RAM by taking the image of the RAM. In one of the related researches, it was shown that encryption keys can be easily extracted using the tool named FROST. Hence in the context of security, securing both data and functions in any software is important. In this research, we are only focusing on data part of the software that often includes the data contents that comes under Intellectual Property.

In the Recent past, there were multiple ways that were introduced besides the obfuscation that can hide Intellectual Property. One of the categories of techniques includes the storage of keys in hardware such as magnetic disks, CD/DVD drives, RFID cards or special onboard ROM chips specially designed for this purpose. These can be stored in the plain text or encrypted but considering encrypted storage better of the two. It also has the issues, as these will be loaded in the RAM as the plain text and that can be extracted or analyzed using known techniques. RFID has the privacy issues related to them even if it is kept at rest. So, these can work in some limited environments but these cannot be preferred generally[17].

One of the techniques that are in research is to store keys on chips present on board. That chip implements the cryptographic algorithms. Off-chip storage is considered vulnerable due to the presence of a large number of attacks possible. These chips can be volatile or non-volatile. Volatile chips are considered better than the non-volatile ones. As volatiles chips offer more flexibility than non-volatile but non-volatile saves space. Volatile chips also have the issue that they are subject to cold boot attacks [17]. As volatile memories tend to keep keys in the memory for some time even after the power is removed. While non-volatile memories keep the keys in the memories even till the time keys are removed by another process. Hence these keys can be copied by the attacker [20]. To solve the issue many solutions are presented that use NLFSR (Non-Linear Feedback Shift Register) to store keys [21]. One such implementation will be discussed in the literature review section.

1.7 Research Problem

In this thesis, we will be discussing a proposed algorithm that can be used to hide different data sets that are present in any software executables as constants. if the system is categorized as the open system i.e. attacker has the access to the system or access of software executable. It will be hard for the attacker to extract anything useful by using reverse engineering tools available.

1.8 Chapters Overview

This thesis is organized as a sequence of chapters that explain the implementation and research done for this thesis.

"Introduction" is the chapter 1, this chapter describes the basics of topics associated with the thesis. Initially, it explains the importance of software executables and their transportation to the client end. Then how keys and data are important for the software executables is explained. There are multiple ways that can be used to hide the keys some of the ways are explained and at the end, the problem is pointed out about which this thesis is done.

"Literature Review" is the chapter 2, this chapter summarizes the different research projects done in the domain. There are a large number of research projects done in this domain but we have selected 9 research projects that are most related to our research. Each research project is summarized and then it is compared to our idea.

"Research Methodology" is the chapter 3, this chapter explains different methodologies that are used for research projects now a day. Out of those techniques Mixed Approach is selected for this thesis. This chapter then explains how the mixed approach helped in completing each and every objective of this thesis.

"Proposed Algorithm" is the Chapter 4. This chapter explains the algorithm designed for the problem identified. It describes each and every step of the algorithm with the reasons for selecting those steps. This algorithm is for the software distribution end, so in the end, chapters explain how any software distribution can be helped with this and how this algorithm can be integrated with the software.

“Implementation and Results” is the Chapter 5. This chapter consists of two portions. First explains the implementation part of the algorithm. It is developed in Java SE. It also explains the main characteristics of the implementations along with the limitations of our implementation i.e. it explains what our implementation can do and what it can’t do. While the second portion is related to evaluation and results. It deals with what are different types of attacks that are possible in the domain of our research. It also explains the resistance our solution provides against the attacked outlined second phase of this chapter. In the end, the results are shared.

“Conclusions” is the chapter 6. This chapter mainly summarizes our whole effort. It highlights the background of the problem and then the solution that is proposed. It also highlights the main discoveries of the algorithm. It is also explained how this algorithm can be used in the software industry. In the end, some recommendations are listed that can be helpful in the relevant future research projects.

CHAPTER 2

2 LITERATURE REVIEW

In order to get a deep knowledge and strong understanding of the software executable security domain, numerous research papers have been studied that were presented by distinguishable authors and we have also cited numerous online resources to understand the need and the gap present in the current domain to accurately define the problem and its scope. Some of the notable citations are briefed below and rest of citations are listed in references.

2.1 A Key Hiding Based Software Encryption Protection Scheme [22]

2.1.1 Problem discussed

Software is highly important in today's life. Software is considered critical e-assets of a company residing in their inventory or being sold to clients. Most of these e-assets contain valuable information, this information may include but not limited to cryptographic keys, serial keys, product keys, and functions. Advancements in the field of reverse engineering have opened new ways to identify Intellectual Property information present in any software. Attackers may attack any executable/bytecode of any software and extract major part of useful information residing in it [15]. Usually, the method adopted by the attackers is to decompile the executable/bytecode into the source code. Obfuscation is one of the techniques that are new in research and provides a better protection against reverse engineering [23]. It makes program's control flow harder to understand by rigorous rearranging and renaming but this cannot save the executable/bytecode from being decompiled. In this research, it was proposed that a robust encryption technique can be better at hardening the executable/bytecode to understand. In this technique it was proposed that to secure that Intellectual Property item, it is better to encrypt the whole software at the vendor end before sending it to the client. Encryption output must or should be randomized, hence randomized information in the executable/bytecode will be useless for the attacker [22]. The security of this process relies on the security of key(s). For the security of key, it was proposed that Trusted Runtime Environment (TRE) must be placed on the user machine, similar to the one that exists in the context of TPM. In this algorithm Static code is kept in encrypted form, concatenated with decryption algorithm. At runtime, the hidden key recover algorithm is placed along with the software loader part. Key recover algorithm outputs the key that is used for decrypting the software before the actual software is executed.

2.1.2 Proposed Solution

Encryption process was designed to support symmetric as well as asymmetric encryption techniques. A key named SK (i.e. Secret Key) is used for the encryption. In the case of Symmetric encryption, SK will be the only key and will be present in the software. Whereas in asymmetric encryption, the encryption key will reside with the software vendor. Whereas SK will be the decryption key and it will be present in the software. To secure the SK threshold key scheme was used which can be Shamir's (n, t) scheme [22]. Then protected SK named is PSK is concatenated with the encrypted software. At last Key hiding, the algorithm is used, which mixes the key/key factors into the encrypted software. Now, this secured and protected software can be released/dispatched.

At the client end, protected software is input to the key recovery algorithm present in the runtime environment. This recovery algorithm output any t key factors out of n where ($t<n$), encrypted software and protected SK [22]. Protected SK and t key factors are passed into key protection scheme that outputs SK that is used for decryption purpose.

2.1.3 Paper Analysis/Review

The encryption tools used in the proposed algorithm [22] has already known vulnerability to identify the encryption algorithm as well as keys. In this algorithm key is processed and mixed in the ciphertext, this is a kind of key management. The algorithm proposed in [22] outputs the encrypted software. So, ciphertext-only attacks can be launched on the encrypted software. Key stored in executable can by extracted completely or partially by locating the key factors. There are a lot of redundant key bits stored in the executable, hence there is a greater probability of finding key bits. Let's suppose one can find 40% key bits then lesser number of bits have to be brute forced. Secondly the designed algorithm is a time-consuming process, software is supposed to be robust but in this case, every time software is executed it must be decrypted and for this key has to be recovered every time.

2.2 Security Challenges for Open Embedded Systems [24]

2.2.1 Embedded Systems – An Overview

Embedded Systems are usually defined as computing systems that are designed for a specific function and have a larger mechanical or electrical part. Embedded systems have found their usability in different fields of daily life that range from automobiles, factories automation, home appliances that come in smart homes, healthcare, transportation, sensors, commerce & finance etc. Traditionally embedded systems are characterized as closed systems i.e. the systems that do not communicate with the outside world and having all the necessary information that is needed for their operations. Now with the advancement in the technologies and need for embedded systems, these systems find their usage as open systems too. Embedded Systems with network connection or that can communicate with the outside environment by any means is an open system. These open embedded systems deploy standardized protocols. As these systems have the ability to communicate or share the information, so these systems allow remote access, remote maintenance, and remote control. This remote access and management make the open embedded system vulnerable.

2.2.2 Security issues associated with Embedded Systems

On possible attack, may cause to compromise any of the basic three pillars of security i.e. Confidentiality, Integrity and Availability and may even cause lose total control of the device. In this paper, two types of security challenges were discussed

- Device security
- Information Security

Device security includes the physical security as well as the logical control of the device. Physical security is related to the deployments of embedded systems in any unsafe environments, hence can be accessed by the adversaries. But bigger security issue is, adversaries can access the devices remotely exploiting the flaws in software’s implementation [25]. There are many vulnerabilities that can be present in any of the system. These vulnerabilities include programming errors, access control, weak authentications, weakness in cryptographic implementations, vulnerabilities in web-based implementations [24]. These problems usually arise because most of the developers are not good at security. Secondly, due to a large number of projects that are in pipeline at any organization, it's near impossible to test each and every execution path present in the code [26]. By using some of the following ways we can avoid these problems

- Control Flow integrity to deal with buffer overflow attacks [26] [27]
- Secure set instructions to call functions [28]
- Use Safe Programming Language [29]
- Protection of OS Kernel E.g. Trust Zone [30]
- Effective use of Software Engineering Practices [11]
- Use of IDS/IPS for detection of Malicious software [31]

Weak Authentication and access control is another problem which results in an adversary taking control of a sensor node and may modify or block any data sent by the sensor node. An attacker may also send sensitive data to an unauthorized person or application. Hence it may harm the node in the context of confidentiality, Integrity, privacy, and availability [24]. There are multiple solutions or remedies that can lessen the effect

- Implementation of the hierarchical or zone-based scheme rather than a flat one
- Encrypted exchange of authentication messages
- Strong password policy
- Dual or Multi-factor authentication

Information is also an important part of any embedded system, there are many characteristics that must be held in the process of information processing, collection, storage, and communication. These characteristics include confidentiality, integrity, privacy, and availability. There are some basic flaws in the embedded systems like cryptographic algorithms are usually not used to have better performance, some have no authentication procedure while allowing remote connection. If these are used, then keys related to them are usually easily guessable or being used for longer time durations.

DOS attacks are also possible as embedded systems can run out of battery in case of heavy computations given by attacker. Solution or remedies to these problems can be

- Lightweight cryptographic algorithms can be implemented [32]
- Some of the workloads can be shifted to gateways or some other controllers [33]
- Battery Problems can be solved by having alternative solutions for power supply [34]

2.3 Protecting Cryptographic keys on client platforms using virtualization and raw disk image [35]

2.3.1 Problem discussed

Secure key management is a serious challenge in the domain of software-based cryptosystems. Due to advancements in the reverse engineering and forensics domain, it is now easier to attack such systems. Due to changing trends, it is easier to steal cryptographic keys rather than attacking the cryptographic algorithms. Usage of different cryptographic algorithms i.e. digital signatures, encryptions are used to have secure communications, but there is a problem that systems itself remain vulnerable. Any unverified application can be executed by the system user, which can exploit the system [36]. Malware can be used to exploit such applications and bugs in it. True crypt [37]and Bit locker [38]can allow the users to encrypt the data present in the system but applications can cause the vulnerabilities in the system.

2.3.2 Proposed Solution

In this paper, there are two methods explained. In the first method, virtualization is used to secure cryptographic keys. In the second method, there is an application that securely stores and retrieve keys from the secondary storage. These methods are software based key hiding methodologies as these can be used everywhere in each scenario as hardware-based methodologies are not feasible for every class of users [35].

Virtualization provides the basic functionality of memory isolation. This property can be used for storing keys in an untrusted system. The system in the proposed solution comprised of three main components.

- Host OS
- VMM (Virtual Machine Monitor)
- Guest OS

Guest OS is the system, on which all of the services are running like FTP(File Transfer Protocol) service, email, HTTP etc. The user uses the guest OS to perform any of the functionalities that are required. VMM is installed on the host OS and have the role of managing all the guest OS present in the host OS. Whereas host OS is the main OS and has no applications installed but just only VMM and cryptographic keys and routines. Each guest OS cannot access any of the other guest OS, VMM or the host OS. So, anything stored in the host OS is saved from the applications and attacks on the guest OS. If the user executes any vulnerable applications still the attacker cannot access the host OS and cryptographic keys.

In the second method, an application is designed that writes key bits on the secondary storage and on request key bits can be collected from the memory to be used in cryptographic processes. Applications analyze the unused sectors of a file on the secondary storage. This all process doesn’t use system calls of the operating system. Now key bits are spread all over the locations. This stored information is made part of the program as key fetch block. These extra instructions are added to the program as the junk instructions. In each of the installation, there are different locations on which key bits are stored. Hence if any of system is compromised, the remaining of the systems with same implementations will remain safe [35].

2.3.3 Paper Analysis/Review

These methods are better in the context these are software based key hiding methods but are vulnerable against attacks, that include hardware-based methods to acquire cryptographic keys. i.e. tools that can extract RAM contents or HDD image. In the first method, systems must be specially designed having keys installed and cryptographic routines pre-installed. Guest OS performance is never at par with the host OS. There is performance lack in this method. In the second method, RAM image and HDD image is very useful for key extraction.

2.4 Secure Key Storage Using State Machines [39]

2.4.1 Problem discussed

This paper addresses the hardware-based storage of cryptographic keys. Cryptographic keys can be stored in hardware as well as in software but when there is the hardware-based implementation of the cryptographic algorithm then the key is preferred to be stored in some hardware-based module likely an on-chip memory [40]. On-chip memories can be divided as volatile and non-volatile. Volatile memories need a battery back-up so key remains there in the chip. This makes the hardware-based solutions costly and in some solutions, there is not enough space to accommodate the battery. whenever battery will be powered off, the key will be lost. Hence battery-backed systems are vulnerable. Volatile memories are also vulnerable to Cold Bot Attacks [17]. While Non – Volatile memories have the vulnerability, that attacker can take an image of the memory using any of the states of the art imaging/forensics tools [20]. As key remains in the memory all the time so this is attack has the higher chances to be successful. So, we face limitations in both the implementations. In this paper, state machines are used to store keys. Binary State machines are preferred in the implementation above the basic NLFSRs, there are multiple reasons explained but two important were

- Binary state machines will be smaller in size
- Binary state machines have a smaller propagation delay

2.4.2 Proposed solution

This implementation process starts with a key. Key can be of a variable length of any length till a maximum of 1M bits. Key is divided into m-tuples. This division is there to reduce the size of the implementation circuit required, this m can be of any value like 2,3,4…. Then these tuples are encoded into a new set of data (encoded ones). Tuples are analyzed to find out that which combination is repeating itself the most number of times and let call that number of repetition n. Then (log n) determine the number of additional bits be appended at the end of each tuple to identify each tuple separately, especially the repeating ones. The number of tuples that are not repeating has these bits as don’t care bits. Then these modified tuples can be transformed into the functions that can change the current state to next state. In this transformation, the table has don’t care in input as well as the output.

Hence the state machine implementation size can be further reduced. In this whole process, all the steps can be standardized can gave good accuracy except the best circuit for feedback functions as we have the standards/exact solutions for five bits but when the number of bits increases when don't have these but there are certain heuristics algorithms that can be used [39].

2.4.3 Paper Analysis/Review

This Implementation was a move ahead for storing the key on a system that is in the un-trustable environment. But the limitation remains that this is a hardware-based solution and uses FPGA's hence this solution can find its usability in a certain domain but can't be applied generically on each system. Secondly when we work in cryptographic domain using keys sometimes we have multiple keys and each have different life spans i.e. master key, session key etc. when we have implemented a set of keys and of the key is to be changed the whole process is to be executed again and this loop will continue until the keys are changed. So, this problem still persists with this implementation. Every time the key is changed the system will be changed the hardware implementations of keys hence the time delays will be associated with this. So, this process can be used but in the limited domain only.

2.5 Keeping Secret Keys Secret in Open Systems. [10]

2.5.1 Problem discussed

This paper describes the previous work done in this research work. Basically, cryptography is centered around the importance of key i.e. if the key is safe then whole cryptosystems is safe [8]. whenever software is dispatched having the key in it. These types of software trust the host system that its safe from attacks. One of the examples from real life is STB in these keys are embedded, users or attackers may attack this STB for financial benefits [3]. Nowadays open systems or closed systems being vulnerable to attacks due to recent research in forensics and reverse engineering. Hence memory access from the attackers is a serious concern for the security of keys. Some of the hardware storages may include the storage of keys in ROM, USB, RFID cards etc. but these all are vulnerable too with a list of attacks [17]. There are multiple encryption techniques for the systems have evolved. These techniques include full disk encryption, file vault. But whenever the code or data is to be used, it is loaded into the memory as plaintext. Hence any of the memory attacks can be used to get the data from the memory. There are some white box techniques that are in use, these techniques imply that keys must be scattered in the executable binary [8]. So, that if the attacker has the access to binary even then it will be hard for him to extract anything useful from the binary. There are multiple solutions that were purposed by the researchers, these include some of the hardware-based solutions and some were the software-based solutions [39]. Hardware-based solutions were better in security but have very limited applications as compared to software-based solutions. Hardware-based solutions include the State machines implementation in FPGA, while software-based implementations include the code obfuscation, Code Encryption and some of white box cryptographic techniques.

2.5.2 Proposed Solution

This data specific obfuscation's process is divided into the following steps [10]

- Key is divided into randomized sized chunks. Using randomization source as the input.
- Each chunk is divided into respective mathematical functions. These are equations that have basic additions and multiplications
- These functions are then converted into java syntax expressions. These include some junk expressions too.
- As the last step, code obfuscation technique is applied. This code obfuscations technique can be chosen from any of the known techniques.
- Then at last compiler created an executable that can be dispatched to the client

2.5.3 Paper Analysis/Review

In this implementation, a try is made to make data secured but there are some problems in this implementation. If there are equations are a quite simple i.e. simple addition and multiplication. Hence mapping of original and mapped data can be analyzed. Secondly, there are functions that return the value of data, that may be present in the memory and is prone to memory attacks. A random number is of 4-bit space and 12-bit space hence a number of combinations will be less hence can be mapped and analyzed. So this is an easily attackable implementation.

2.6 Methods to Protect Cryptographic keys on Safety Critical Systems. [41]

2.6.1 Problem discussed

There is an increasing need for embedded systems and due to cost-effective solutions of hardware components required for the embedded systems. But these systems have quite a large number of vulnerabilities and issues related to them but there are certain safety-critical systems that need special security attention [42]. There are attacks on its core functions i.e. memory management, process management and data management. But there are some security issues such as data confidentiality especially when we are dealing with Intellectual Property. These devices are also vulnerable to MATE attacks [43]. There are some attacks make cryptographic keys disclosure. One of the possible solutions for key protection is TPM [44] but it will increase the cost of the whole system and also require hardware architecture redesigning. One way of handling this scenario without TPM is using software-based solutions. One of the methods is data obfuscation it increases the cost of reverse engineering. There are two types of obfuscation Static and dynamic [16]. Dynamic one is the difficult one to attack than a static one. This paper proposes the strategy of learning attacker’s way of action and then applying strategy accordingly.

2.6.2 Proposed Solution

This paper describes multiple implementations that are all software based solutions. There are solutions that store keys in the processor using special instructions and these don’t require special hardware for this implementation. One of the solutions uses software encryption and decryption to hide the data and functions that are present in the software executables. Encrypted Software is different to understand hence time for reverse engineering is increased. Another solution transforms keys into randomized equations and adds that to code. If there are attackers, then it will hard for them to reverse engineering these randomized equations to data. There are solutions which have virtual machine monitor hiding keys in an open system. This paper presents multiple ways for saving cryptographic keys. First are the dead execution spots that were created against strings attacks. It can be created using call obfuscation, return obfuscation or false return obfuscation. If someone still gets information using entropy analysis, then one of the possible solutions can be splitting cryptographic keys to reducing the impact of entropy analysis. If these two strategies don't work then false instructions can be created and inserted. The number of instructions that are inserted into the software depends on the size of the cryptographic key. Then Insertion of garbage instructions is the next solution that can be applied. Later in the paper program diffing and recurrent attack can be applied on static obfuscation. Then obfuscation engine with multiple algorithms can be the solution. Then insert anti-debugging techniques can be a solution as well [41].

2.6.3 Paper Analysis/Review

In this paper, there are multiple solutions that are discussed but they have not described a specific technique that can be applied. We have to apply all 6 techniques that may increase the obfuscation creation code time and increased processing time and 5 out 6 have attacks that are well known hence its easily attackable system. This paper can be treated as a survey of multiple techniques present for the problem.

2.7 Embedded Systems Security: Threats Vulnerabilities and Attack Taxonomy [45]

2.7.1 Problem discussed

This paper is about the study of the vulnerabilities and attacks related to embedded systems. Embedded systems are the combinations of software, hardware, and the mechanical part. Due to an increase in usability, most of the domains in everyday life are using embedded systems for example cars, airplanes, hospital instruments, railway department instruments etc. [4]. Most of the systems are connected to the internet and have physical and remote access features. Due to these features security becomes an important feature. As many of the systems are safety critical systems, hence there is a need for security implementations for these embedded systems [46]. There are many proposals that were made in which taxonomy was discussed. There are many key factors related to embedded systems like deployment scale, resource limitations, physical protection and securing of all factors is a challenge [47]. While creating a taxonomy for the threats, vulnerabilities, and attacks. The first thing to learn is that how the attack is made on the embedded systems hence learning the attack patterns is important. This paper uses the above-explained methodology to create a learning-based taxonomy.

2.7.2 Proposed Solution

In this paper, the researchers have used the CVE (Common Vulnerabilities and Exposures) list of vulnerabilities. For creating the taxonomy five attributes are being used [45]

- Precondition
- Vulnerability
- Target
- Attack Method
- Effect of the attack

There are multiple types that were selected under each of the above-listed attributes. Precondition have multiple types that can be selected but the ones that were selected were [45]

- The set of devices that are constantly connected with the internet
- Set of devices that are on the network and allows local and remote access to any user.
- Set of devices that have direct access for the attackers
- Devices for which attacker can come in the physical proximity of the attacker. Like the area of the range of Wi-Fi or radio frequency range
- A miscellaneous precondition, like device, may have to run a code or to do some communication over the network.
- At last, there are some devices that have unknown preconditions.

The second attribute that was researched was a vulnerability and the types of vulnerabilities listed in the research were [45]

- Programming errors, which may include control flow attacks, input parsing errors, memory management errors.
- If the device has a web-based interface for management tool of the device. These web-based interfaces may be less regularly updated hence can have unpatched vulnerabilities
- Now a day almost every device has an authentication and access method implemented. So, any of the weak authentication methods will be a vulnerability.
- Due to security requirements, many embedded systems contain the cryptographic tools. Improper use of cryptographic tools or key mishandling lead to vulnerabilities

- There are a lot the vulnerabilities that are unknown can be identified any day, similar to the one that is known as a zero-day vulnerability.

As for the targets, it can be any of the following

- Hardware
- Firmware/OS
- Application
- Protocol of communication used can also be under attack.

As per the attacks, there can be many attacks but majorly used attacks are

- Control Hijacking attacks
- Reverse Engineering
- Malware
- Injecting Crafted Packets or Input
- Eavesdropping
- Brute-force attacks
- Normal use of the device can lead to misusage of the device due to mal configuration of the device.
- Unknown Attacks that can be used for the listed vulnerabilities

Possible effects of the above attacks can be

- DOS – Denial of Service
- Code Execution
- Integrity Violation
- Information leakage
- Illegitimate access
- Financial loss
- Degraded level of protection
- Miscellaneous- like for some effects there is not full information available like the redirected traffic online
- Unknown

Using the above-listed types, a taxonomy was created. 3862 CVE list subset was tested and executed successfully.

2.8 Cryptographic key protection against FROST for mobile devices [48]

2.8.1 Problem discussed

With the more advanced research and usage of IoT & mobile devices, there are ever increasing privacy issues that are coming to these devices [49]. There are many reverse engineering tools that are available for analysis of these devices. One of the tools is FROST [19] (forensic recovery of scrambled telephones) that was proposed by Muller et al. The main problem that came with the IoT and mobile devices is how to have convenience as well as better security in terms of privacy of information. Most of the devices use Android due to its expendability and openness [50]. Android provides the option of full disk encryption since 4.0 version. It may be problematic for some of the forensics tools when the phone is locked and investigator can’t get into the phone to get the data. one of the options possible is the cold bot attack [51] or simply the side channel attack which takes the image of the RAM to get the maximum possible information from the RAM and there are certain conditions that can increase the percentage of the information that is retrieved and this is all the logic of the FROST implementations.one of the possible solutions to lessen the effect of FROST is to use the registers of CPU for the storage of keys, but this can slow the execution of encryption and decryption process. As per the Cold bot attack, there are certain things that can increase the information extracted from the RAM [19]. RAM usually follows the remanence effect i.e. that data contents degrade with times and this effect’s rate increases when the temperature is high and lower rate when the temperature is low. As discussed FDE is the main feature of Android 4.0 and is a compulsory feature after version 5.0.

2.8.2 Implemented Solution

For the implementation Encryption key is very important hence its creation and storage is a feature to look for. There is a data structure called crypto footer that stores the key generated after a long key generation process along with the salt. In the research of changing the location of storage of key. The steps of the process are as follows [48]

- Apply for the structure of encryption algorithm

- Calculate the Storage address of command line parameters
- Returns the address to the structure
- Set key and initial vector and data
- Fast reboot to recovery mode
- Key is covered by command line parameters
- Forensics tools failed to capture key.

With the usage of the above-said algorithm, the cost in terms of time is almost zero. The time factor is increased by a factor of 1.03. Secondly, this method is limited to work only one key. If there are multiple keys then this implementation will not work

2.9 Embedded Systems Security Challenges [7]

2.9.1 Embedded Systems and their usage

Embedded Systems are the major necessity of life now days having its applications in each and every domain of daily routine from home use devices like washing machines, watches. These also include components in the cars like the MP3 modules [5]. These devices have small sizes like watches till the large size like PLC. Some of the devices are vital, time-critical applications and these devices have many issues related to security like leakage of private information i.e. confidentiality of data, Integrity of data and availability of data and services at any of these issues will be critical for the system. Next generation is of IoT and these will be next generations of Embedded systems [6]

2.9.2 Security Challenges associated with Embedded Systems

This paper discusses different security challenges related to above mentioned embedded systems. Three major physical layer security challenges are [7]

- Side Channel Attacks: These comprise a major set of attacks on the systems/devices when we deal in the physical layer. This includes attacks on related modules but not on the main application. These include getting a copy of RAM, run-time re-configuration can be exploited, Self-recovery of a device as also be exploited.
- TPM (Trusted Platform Module) is considered a better solution when we are dealing with security for individual devices. The challenge with these of modules are i.e. firstly as these are

microcontroller it can carry online a small information only and if we want to increase the capacity then performance will decrease and cost of the overall system will be increased.

- Protection of Power supply: Most of the embedded systems are depended on the battery power and if there is no power then it is considered as the availability issue and whole system or the individual node will be off and services will be cut off, this scenario is known as a denial of service attack

2.9.3 Possible Solutions to Security Challenges.

There are multiple ways to handle the above-mentioned challenges, one of them is access control. Access system methodology depends on the hardware and services provided by the embedded system. There are three types of access control that can be applied but correct implementation and selection of any of these is a challenge.

- Profile-based authentication and access control
- Access code based authentication and access control
- Predefined topology like MAC filtering.
- Policy-based access control, this is XACML based architecture that comprises of the following modules,
    - Policy Enforcement Point (PEP)
    - Policy Administration Point (PAP)
    - Policy Decision Point (PDP)
    - Policy Information Point (PIP)
    - Context Handler

In the domain of Access control, irregular access can also create issues like DOS or DDOS. These types of attacks can be launched at any OSI layer [52]. Hence are multiple ways to implement DOS attacks but correct authentication and authorization along with integrity checking can save many systems from these attacks.

For any security-based implementation, cryptography is an important tool that is used. All of the modern-day cryptographic tools require relatively higher processing power than the older ways of doing cryptography. As discussed embedded systems are small and have the lower processing power and have low storage hence there is a limitation in the cryptography domain as well. This is also a challenge for the creators/developers of the system. For this, there is a solution named lightweight cryptography [53]. There are many algorithms that came under this heading. Another issue that is related to cryptography is the key distribution [54]. There can be pre-shared key or runtime sharing of the key by the server. Both ways have their own pros and cons. The pre-shared key issue will also be solved in this research.

There are multiple challenges that are related to network communication and management but most of them are out of scope for this thesis hence only those are listed. Following are the challenges that are related

- Secure Resource management
- Reputation-based Schemes
- Anonymity and location privacy
- Secure Service discovery, composition, and delivery protocol
- Communication Security

In this paper multiple challenges are discussed related to open embedded systems, in the software part of systems contains keys and data related to systems, there are challenges related to them as discussed in paper and in this thesis, there is solution presented that can solve some of the listed challenged i.e. storage of pre-shared key in the software so that no side channel attack can be launched on that.

CHAPTER 3

3 RESEARCH METHODOLOGY

3.1 Research

Research usually represents a set of steps that are used to find an answer against any set of problems and that answer is not present in the domain as yet. First steps follow to gather knowledge and information from the domain of problem set. In this way, we are able to find out the existing research in the particular domain and existing solutions and room for new research. This above-said step is usually termed as a literature review by the researchers. In this existing solutions and contributions made in the domain of the problem can be identified. In this, all process the problem set can be refined and a clear scope can be defined as the problem according to the work done in the field. According to the major group of researchers, research can be broken down into the following steps [55]

- Defining and redefining problems
- Study existing Solutions (if there is any)
- Suggesting a new solution and formulating a hypothesis for that
- Implement/simulate the proposed solution, organize the collected data and evaluate
- Craft deductions and, draw conclusions based on the results
- Match/Test the deductions and conclusions with the above-finalized hypothesis

Research work or in short, the solutions that are under consideration need to have a logical proof. As research work doesn’t allow guesswork and anything that is without any solid ground. For this solid ground for the solution proof, a systematic and sequential approach is necessary besides the innovation and initiatives that are required for the research work. There is a need for analysis, facts, figures to explain any concept. Moreover, a research process is based on logical reasoning is more meaningful than the one which is devoid of logical analysis. Logical reasoning rules consisting of either inductive or deductive approach, govern a course of research [55]. Based on the role of usage of other researcher knowledge in the research forms some categories describing different ways of doing research. Some are listed below

- Research based on hypothesis and testing cycle
- Research based on diagnostics
- Research based on descriptive case studies
- Research based on explanations and formulae

3.2 Types of Research

There are many types of research that are in use by the researchers following is the list of these types and their brief description

- **Descriptive Research** is a survey research. As per the name suggest it is a type of academic research, that is based on data and a describe properties of the state of affairs as existent presently. This type of research is important in the compilation of observations on the data collected. It can also include the analysis of the data to draw conclusions from it.
- **Analytical Research** it is a type of research that involves analysis on the collected data and existing data and algorithms. These set of analysis can be very helpful in creating logical conclusions for the explanation of a particular phenomenon.
- **Applied Research** it is a type of research that can solve problems that are identified by industry, businessman, society, economics. This is a type of research that can find immediate remedies to the problems faced by the above set of people. This type of research may include social problems and technological upgradations and advancements.
- **Fundamental Research** it is a type of research that is planned to accrue information and awareness to build up and strengthen the existing body of knowledge. This research, in fact, offers a foundation for the applied research to build upon. This kind of research is fueled by the interest and curiosity of the scientists and researchers.

- **Quantitative Research** it is a type of research that focuses on establishing cause and effect relationships to test a certain hypothesis. This research includes the use of statistical functions that are applied to the collected data to validate propositions. The actual focus of this research tests an issue that is already defined and the scope is defined.
- **Qualitative Research** it is a type of research that tries to explaining and understand the social phenomenon, qualitative data is used. Some of the ways and means or procedures employed in this type of research include but not limited to aptitude tests, opinion polls/tests, interviews, literature reviews, program evaluation etc. [55].
- **Conceptual Research** it is a type of research that tries to develop new ideas and concepts based on existing theories. In this methodology, an idea is identified and investigated. The study is completed on the pros and cons of the idea and then reinterpreting it in a modified and improved definition. So, this research aims at clarification and creating new idea and concepts to explain and improve existing perceptions.
- **Empirical Research** it is a type of research that is divided into two phases. First, one is the collection of facts and creating and drawing conclusions based on facts. In the second phase verification of the conclusions that were created in the first phase. These create new notions based on those conclusions, by observations and experimentation. This type of research involves research, implementation, and practice. The main aim of this research involves creating a hypothesis which is then evaluated and tested to be true or false [55].

There are two main concepts related to research. These concepts are Research Methods and Research Methodology. The former is the collection of tasks, procedures, processes, etc. which a researcher conducts and employs during the duration of his research work. On the other hand, research methodology stipulates guidelines on how to carry out research, outlining the rules of the game to steer and regulate a systematic and structured research. Research methodology is a general term which covers research methods as well. Researchers need to understand and pick research methods which are most apt and applicable in given scenarios. The basic premise of each research method and its applicability in the actual situation must be understood by the potential researchers. Research methodology bears broader scope than research methods. It constitutes research methods and techniques as well as the logic behind using certain methods for particular scenarios.

3.3 Thesis Research Methodology

Thesis research methodology is based on the defined research objectives that are created to add to the domain of problem that is identified by proposing a suitable solution. In this thesis research cryptographic is the domain. A problem is the storage of cryptographic keys and Intellectual property present in the software. There is a number of attacks using the technology of reverse engineering tools. These attacks are usually launched due to implementation issues of the software. In research on the above problem, method of attacks and implementation of software are important to look into to get to a proper solution. There are multiple reverse engineering tools that open executable and a lot of information can be extracted from that special keys, product keys, serial keys etc. As for the cryptography, all the security lies in the key so securing key will be important. In this thesis research, our research matches two of the research types listed above descriptive research and applied research. As we have to study about the already methods of obfuscation designed by other researchers, flaws if any and to solve the problems that still exist in the domain. Applied Research is as obfuscation is already implemented but it can be applied in every scenario hence it can be seen as a problem faced by business and industry [55]. Hence new methods will be designed and will be verified using the data analysis against the older methods developed hence analytical and empirical research will be involved as well. Then, during the second part of our endeavor, we follow the edicts of "Empirical Research" in which we exhaustively implement the hypothesized solutions, collect data and analyze results. Based on those results we refined and narrowed the approach to implementation of the solution while recording the improvements.

3.4 Research Objectives

Following are the research objectives for this thesis research

- Study different attacks on the executables and ways of data extraction using Reverse Engineering tools and techniques

- Design a randomized transformation which may be applied to the Dataset and outputs the result in the form of:
    - y = f(x)
- Design an application which applies this particular transformation on any given program
- Design an attack model to verify the security level, enhanced by the algorithm

3.5 Research Approach

The first research objective is achieved using the descriptive research as it needs the survey of the domain so that all the existing solutions can be studied and collected. This involved collecting all the implementations made in the domain related to the problem statement. In this thesis, we are searching for obfuscation. In this thesis, a total of nine techniques were studied. Then data collected is analyzed using analyzing research. Then a hypothesis is generated a design is generated related to the hypothesis. Now it comes to the next stage in which implementation and testing will be done. So, this phase comprises of

- A new implementation conforms to the requirements of its proposed design.
- New implementation succeeds or fails to address the weaknesses and vulnerabilities of existing setup which it was supposed to improve.
- The implemented module introduces new weaknesses or vulnerabilities of its own.

3.6 Literature Review

The first of the research objective is the survey or the literature review. This includes the study of different implementations made related to obfuscation techniques. All different techniques and their vulnerabilities were studied. The literature review also includes the security challenges of the systems that are physically assessable. At last literature review includes the all the attacks that can be launch on such systems. Attacks patterns will allow us to create better solutions, that can mitigate such attacks in the future.

3.7 Problem Identification

Using the descriptive research and empirical research first phase, it was possible to identify the problem statement for the research. Problem statement for this thesis research is

- Key and another important dataset are mostly embedded in the executable.
- With the advancements in reverse engineering techniques, attacks on the devices and their executable cause the leakage of information, leading to the creation of illegal software copies.
- There must be a platform independent transformation technique which transforms the data set in such a way that it becomes much harder for the attacker to extract any information

3.8 Evaluation Process

We understand that our research effort has been iterative in nature where evaluation of one module leads to the other. While we observed that our developed applications conformed to the requirements of design and also, they did not introduce any handicaps of their own, yet we also observed that they needed improvements.

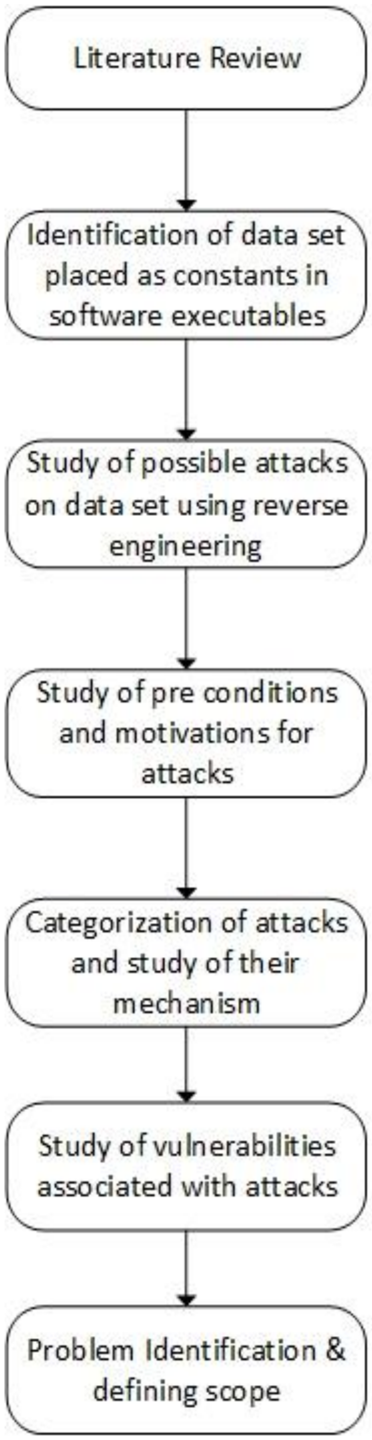

Literature Review
Identification of data set placed as constants in software executables
Study of possible attacks on data set using reverse engineering
Study of pre conditions and motivations for attacks
Categorization of attacks and study of their mechanism
Study of vulnerabilities associated with attacks
Problem Identification & defining scope

Figure 1: Analytical and Conceptual Phase of Research Work

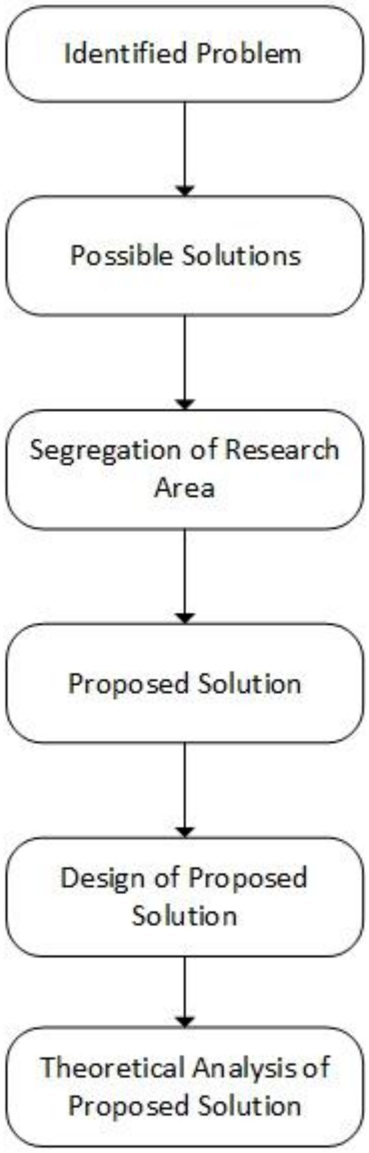

Identified Problem
Possible Solutions
Segregation of Research Area
Proposed Solution
Design of Proposed Solution
Theoretical Analysis of Proposed Solution

Figure 2: Problem Identification and Solution Design Phase

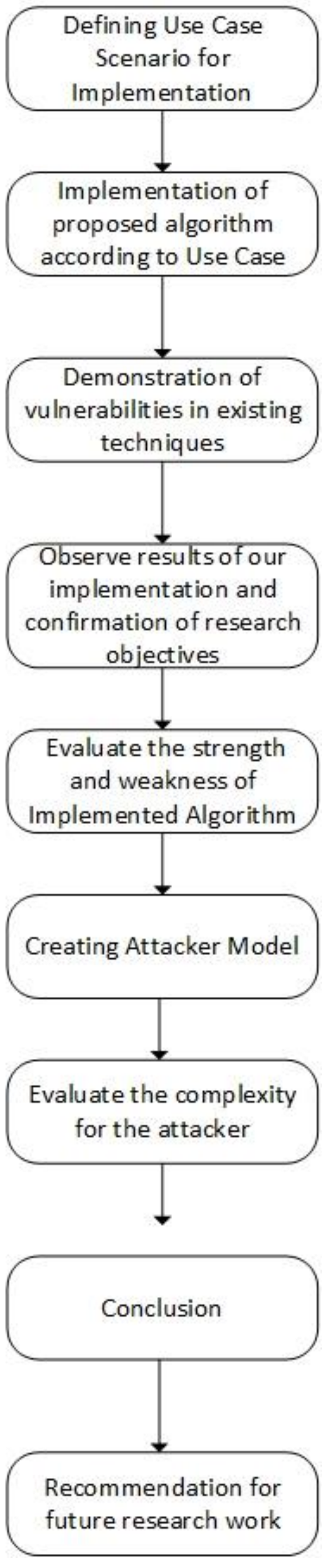

Defining Use Case Scenario for Implementation
Implementation of proposed algorithm according to Use Case
Demonstration of vulnerabilities in existing techniques
Observe results of our implementation and confirmation of research objectives
Evaluate the strength and weakness of Implemented Algorithm
Creating Attacker Model
Evaluate the complexity for the attacker
Conclusion
Recommendation for future research work

Figure 3: Implementation and Evaluation Phase of Research Work

CHAPTER 4

4 PROPOSED SOLUTION

In this thesis, we are proposing a data obfuscation technique. Each executable has some data present in it. This data can be cryptographic keys and some other intellectual property items that may include product keys, serial numbers etc. These data are often present in the form of primitive data types or some other data structure that can hold data i.e. array, string. The task in this thesis is to transform the data discussed above into compile-able programming instructions, such that these

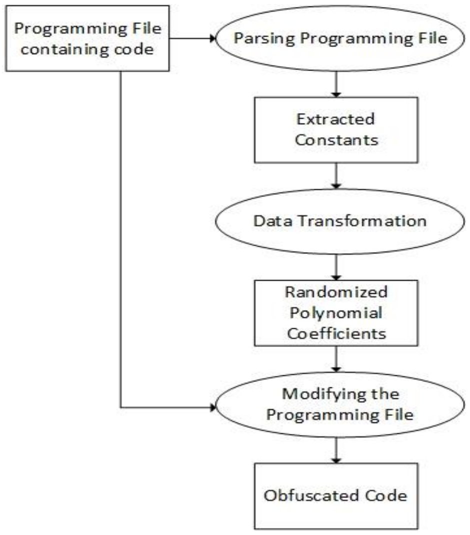

Programming File containing code
Parsing Programming File
Extracted Constants
Data Transformation
Randomized Polynomial Coefficients
Modifying the Programming File
Obfuscated Code

Figure 4: A Complete process in terms of phases, inputs, and outputs

programming instructions can be used in place of the data present. For this purpose, an algorithm is designed to obfuscate the data present and there is a process that reverses a module the transformation for the runtime usage. The whole process of obfuscation is divided into three modules as shown in

Figure 4 with the input and output of each phase of the process. In this process, each module has two or more smaller tasks/steps involved. These three modules are following

- Programming file parsing
- Data Transformation
- Modifying programming file

4.1 Module 1: Programming file parsing

The main role of this module to read/parse the programming file, to identify the data present in that file. This programming file can be in C, C++, Java or C# .NET. There will be a program that can be implemented using the standards of RegEx or simple conditional Statements. The program scans the code present in the files. It reads the code line by line and looks for the data variables present in the files. Some of the data variables hold static data or constants. These types of variables with constants are identified and separated. Cryptographic keys and data that comes under intellectual property are stored as constants in code and hence they are comparatively easier to trace in the executable. So, as the output we have the constants declared as the integer, String, Array or any data type. Now, this data will be processed in the next module. In short, this module comprises of the following steps

- Take any programming file as the input. This file can be .c, .cpp, .java etc.
- Parse that file using conditional statements or RegEx to extracts the constants present in the code. There can be single constant or as many constants as needed hence theoretically there is no limit on this.

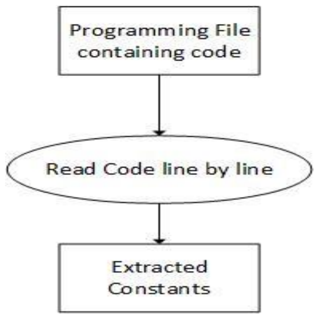


Figure 5: Phase 1 - Parsing the Programming File

4.2 Module 2: Data Transformation

The main responsibility of this module is to convert every constant to a separately randomized polynomial, i.e. same string when passed through the same data transformation process generates different polynomial every time, with the likelihood of generating the same polynomial is very low.

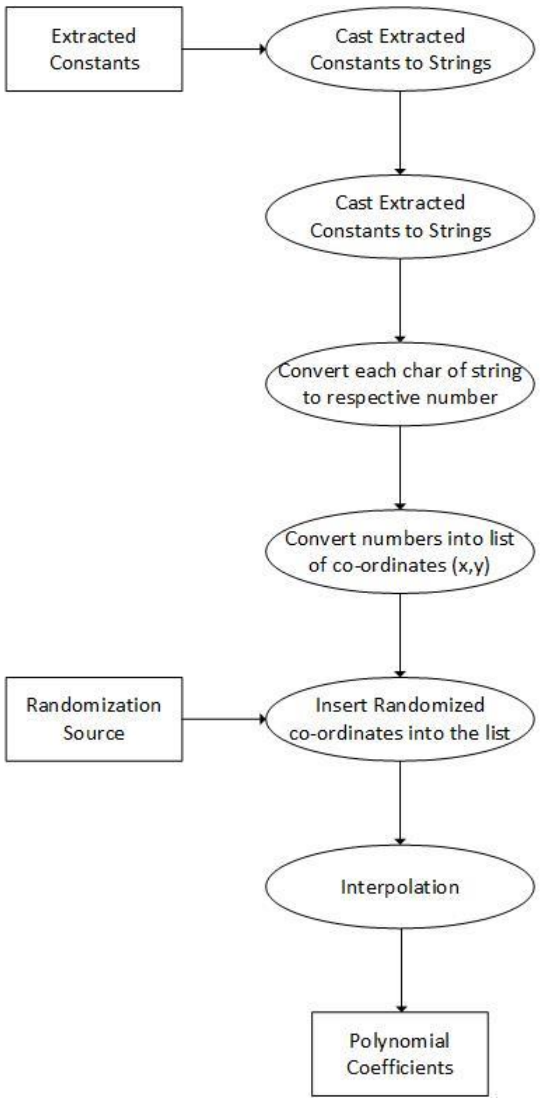

Extracted Constants
Cast Extracted Constants to Strings
Cast Extracted Constants to Strings
Convert each char of string to respective number
Convert numbers into list of co-ordinates (x,y)
Randomization Source
Insert Randomized co-ordinates into the list
Interpolation
Polynomial Coefficients

Figure 6: Phase 2 - Data Transformation Process

To implement the above-said responsibility for different tasks were designed. Following are the list of tasks that were assigned to this module

- Convert each constant to String and then convert each char present in the string to a respective number.
- Generate multiple random numbers and place them randomly in the array of points formed above.
- Now we have a list of numbers, convert these numbers into points format like (x, y).
- Create a trace of the randomized data and insert it with the data inside the array.
- Using the implementation of any standard polynomial theory. Convert the above created randomized data into a polynomial. Effectively we have the all the coefficients associated with the polynomial.

When we have the constants as the input from the module 1, we have to process them to a randomized chunk of data and then to a polynomial. So, the first step is to convert each and every constant into a string. This is done to make all the constants into one form. As constants can be an integer, char, string or any other data structure hence converting to a single data structure that is a string so that one processing logic can be applied to each constant. If all of the different data structure is kept in same then different processing logic will be required and they may not be equally effective. Now there is a string against each constant. In terms of length, there is no minimum or maximum limit associated with the size of the constants. These constants can be made of any number of alphanumeric characters. As the string is a combination of chars, we can convert these string chars into respective numbers. There are n number of ways that can use to convert these string to list of number, the following is the one that is easier to apply

- Use the standard ASCII table for the conversion against each character. Now the output has a list of numbers have numbers equal to the number of chars present in the string i.e. a = 97, B = 66, 2 = 50 and so on.
- Use the personalized conversion table for the conversion against each char. Still, now the output has a list of numbers have numbers equal to the number of chars present in the string a = 0, b = 1, A = 26, etc.

- Use of combinations of chars to form numbers using ASCII table or personalized tables. In this case, the quantity of number will be at least half of the number of characters present. i.e. AB = 6566, Aa = 6597, Z9= 9057 etc.

Now we have the list of numbers present and we want to form polynomial we must convert that to (x, y) pair form. But with this way every time the polynomial will be the same against the serial key. Hence there is a need for inserting randomized data in the list of numbers got from step 1 of module 2. There must be at least 3 random numbers to be inserted and at maximum, there can be random numbers equal to the number of numbers present in the list. These random numbers are the sequence of numbers generated by a TRNG function. After inserting the random number in random locations now. The list is now transformed into (randomized data + actual data). As the inserted garbage at a random location hence there will be a lesser chance of attack that can separate actual data from the garbage randomized data inserted.

Next step is to convert this updated list of numbers having some garbage as random numbers into (x, y) pairs so that it can be converted into the polynomial. In this transformation, there can be multiple ways but the way used is

- Any number present in the list is called y and its position in the list as x. The positions in the list start from 1.

We need a reversal process at the at the program execution time. The process reverses the impact of randomization and obfuscation. Random data is needed to be deleted with 100% accuracy so that original and exact key/intellectual property thing is calculated back. To accurately implement this property, there must a trace of random numbers that are added into the list of randomized (x, y) pairs. This trace includes a hint of multiple attributes such as possible locations, the total number of characters in actual string etc. This trace can be inserted at any location in the randomized list but for simplicity of implementation zeroth location can be used. Importance of trace to be stored and its location is explained in chapter 5 with an example.

In the last step, now when we have a list of randomized (x, y) pairs having actual data, random garbage data and trace of randomized garbage data. This list is to be converted into the polynomial. This conversion is done using the process of polynomial interpolation and we get the coefficients of the polynomial. Interpolation is defined as a process of creating or defining new data points from the existing data points. In this case, we have a list of randomized data points and we input that into the interpolation process. So, polynomial interpolation is the interpolation of the given set of data points to output the lowest degree of the polynomial that passes from the each of the given data point. Usual output is in the form of coefficients. There are multiple ways of doing the polynomial interpolation but two methods that were used in this research were

- Newton series
- Lagrange polynomial

Newton series uses the concept of Newton forward difference equations. This outputs the coefficients in the decimal points and it has the problem of storage and accuracy. So, if this method is used there will be an upper limit on the constant size that can be used as a key in software executable. But the Lagrange polynomial method uses the modulo operator, hence don’t have the limitation related to the size of a constant that is used as a key.

4.3 Module 3: Modifying the Programming file

The main responsibility of this module is to modify the code that is present in the programming file. Till now only data is extracted from the file that is the constants but no change is yet made in the programming file. Now the process of modifying the code consists of the following steps.

- Convert the coefficients of polynomial obtained as the output of module 2 into appropriate code.
- Remove the data/constants from the programming file and replace that with functions.
- Pass the code through any of the standard obfuscation techniques and then compile and dispatch the executable

In the first step, we transformed the coefficients of a polynomial into the appropriate set of instructions in any of the programming language using the respective coding syntax. There will be a function for each constant’s polynomial. If there are N number of polynomial coefficients sets then there will be N respective functions and these functions will be named against the name of the data variable of that polynomial. So that they can be identified by the implementation logic. This function will have a calling of reversal process implementation which removes the randomization hence returns the string back. For example, for a data variable name “secret_key” is converted to polynomial giving coefficients a1, a2, a3, a4, a5 then function will look like

```
Public String secret_key_fun ()
{
        int [] arr; //to contain list of numbers extracted from the code
        For (quantity of numbers required in list)
        {
                ///code of polynomial having coefficients a1, a2, a3, a4, a5
                //number is concatenated at the end of arr
        }
        arr = remove_random(arr);
        return num2string(arr);
}
```

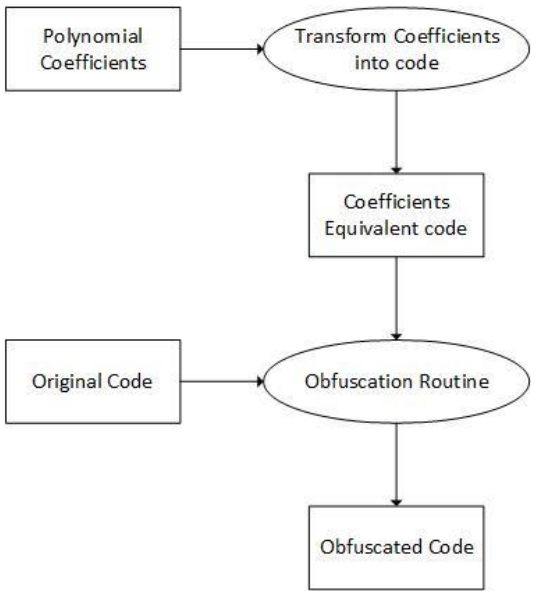


Figure 7: Phase 3 - Modifying the programming file

In this code *remove_random ()* is the reverse process algorithm that will remove the randomized data from the actual data taking help from the traces stored in module 2 of the algorithm. The whole process of this *remove_random ()* will be explained later in this chapter. *Num2string ()* is also the part of the reverse process, hence will be explained in that section. Now, this function will be used in space of the data variables, for example, *if (secret_key ==entered_key)* will be modified as *if (secret_key_fun () ==entered_key)*, All the code will be modified on the similar pattern. All the constants in the code are replaced with the corresponding functions after the modification. The whole process is designed as to transform data part of the code to respective randomized functions. This process doesn't alter any part of the code. So, in the end, we can apply any state of art code obfuscator on the newly created modified code. After this code obfuscation, the output is the actual output of the process. In the end, we will compile the code and dispatch the executable to the client, who can execute that on an untrusted system.

4.4 Inverse Transformation

The inverse transformation is executed on the runtime; it doesn't need to be executed separately before the execution of the actual program. Secondly looking at the whole transformation process module 1 and module 3 are the modules that don't need any reversals as these are the modules that deals will the code obfuscation and data extraction from the code hence no reversal needed. We will only require inverse transformation for the process of module 2. In short module 2 transforms any constants present to randomized polynomial and saving randomization traces at a particular location. So, the trace is important in the inverse transformation process. Inverse transformation process comprises of the following steps

- As the polynomial can also be expressed as y = f(x) and we can extract all the possible values of numbers list by just providing different values for x.
- First, we to extract the trace, location of the trace in the randomized list of numbers will be already known to the software. The location of the trace is by default zero, it can also be changed. For the extraction of the trace, we input location i.e. x= 0 to f(x) and then we can calculate y. This calculated value of y is the trace. Using trace, we calculate the values of attributes stored in it.

- One of the attributes stored in the trace is the size of the randomized list. Execute the *“for loop”* for the number of iterations, according to the value got from the trace.
- Another important attribute stored in the trace is randomization information. Using this value, remove all the randomization present in the list obtained in the last step.
- Transform back number into characters and concatenate them as a string and return this string.

CHAPTER 5

5 IMPLEMENTATION AND RESULTS

This technique was developed with the aim of making it harder for the attacker to extract any useful information, related to data present in the executables, by the use of reverse engineering tools. Our technique will make it harder to extract any data information by static analysis of the executables.

5.1 Use Case Scenario

The algorithm designed in this research process is a generalized process of obfuscating data present in any code. Each programming language has its own programming syntax, so to implement an application that has the ability of obfuscation any programming language is out of scope from the objectives finalized for this research thesis. So, there is a need to finalized a scenario for the implementation of POC. For the implementation, our designed POC can analyze an input code implemented in C++ language. Hence produce a resultant obfuscated code also in C++. In the algorithm as explained in chapter 4, we work on the constants defined in the CUT (Code under test). There can be any number of constants and any types of constants but in our use case scenario we are only considering String constants and we are only analyzing single String constant. So, we are working on C++ input code and will analyze and extract 1 String constant and obfuscate.

5.2 Choice of Implementation Language and platform

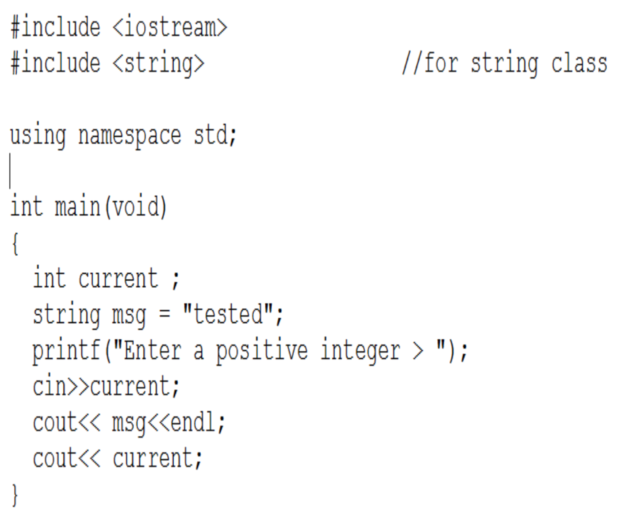
Java was used as a coding medium to implement the research modules. It covered all three phases of the research work: Programming file analysis, Data Transformation and Programming file modification. In-depth details of implementing these phases in Java are explained in the 5.3 heading.

Java is a high-level language and it affords the programmer much room to encode diverse requirements. In addition to that, it offers a modular approach where the different modules can interact with ease. Reusability, troubleshooting and easy alteration of the code were also the added benefits. Common libraries of interpolation, cryptology available free over the Internet were also helpful during the course of the thesis.

For the implementation phase, NetBeans v8.2 is used as an IDE for the implementation of all the modules. It's a versatile encoding environment which offers a comfortable coding experience. Moreover, its diagnostic platform comes handy during trial runs of the code. It helped to test more than one module of the research work simultaneously. Built-in syntax checking of the IDE and corrective suggestions free the programmer from typos and laborious search for applicable code functions

5.3 Proof of Concept

As discussed in the previous sections, Java-based implementation is done. Our implementation is done in three modules, with each of them working separately. Files created and required for the success of the process implementation are kept in H:/ directory of the system. For the first module, it read the CUT (Code under test) from the H:/ directory. There were multiple codes that were tested with the implemented modules. Two of them are shown as screenshots in Figure 8 and Figure 9. We have kept simple codes so that demonstration can be done easily. We will be referring to Figure 8 as CUT1 and Figure 9 as CUT2. In CUT1 there are two strings present and that is "tested" and "Enter a positive integer" while in CUT2 we have three strings constants i.e. tested, done and Enter a positive integer.

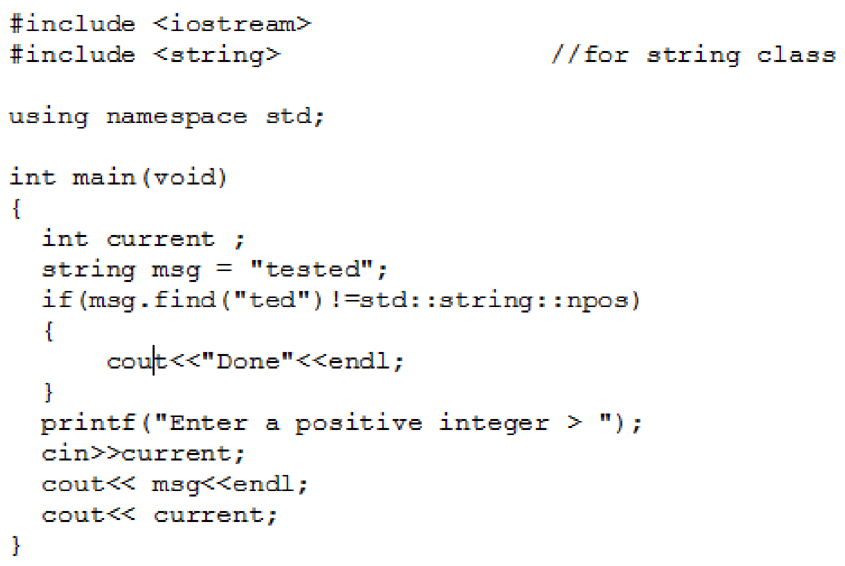


```
#include <iostream>
#include <string>                //for string class

using namespace std;

int main(void)
{
  int current ;
  string msg = "tested";
  printf("Enter a positive integer > ");
  cin>>current;
  cout<< msg<<endl;
  cout<< current;
}
```

Figure 8: Code Under Test - Sample 1



```
#include <iostream>
#include <string>                    //for string class

using namespace std;

int main(void)
{
  int current ;
  string msg = "tested";
  if(msg.find("ted")!=std::string::npos)
  {
      cout<<"Done"<<endl;
  }
  printf("Enter a positive integer > ");
  cin>>current;
  cout<< msg<<endl;
  cout<< current;
}
```

Figure 9: Code Under Test - Sample 2

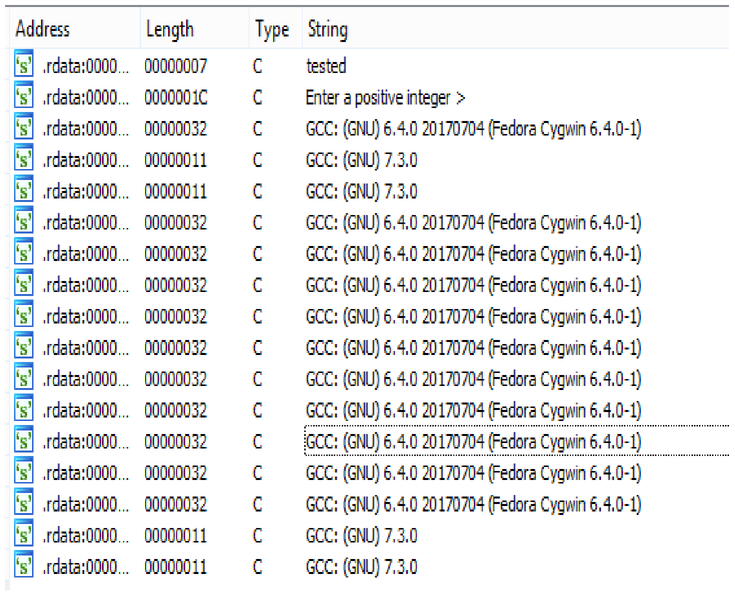
We have passed this two code along with the other code samples through our obfuscation process and output is tested and evaluated. During the obfuscation the required functions and codes are entered by the application itself, hence no modifications are to be made. Which includes the polynomial function and de-obfuscation function. So, that it can be dynamically reversed at the client end.

5.4 Results and Discussions

For the evaluation of the process implemented we selected multiple utilities that can use. The aim of the research thesis was to design and implement an obfuscation process for data present in the executable. Obfuscation is done to defy the attacker from the static analysis of the system. So, during the evaluation process comprised of tools for static analysis of the executables. In this topic, we will be discussing the results got against the static analysis of the executables. Results are collected against the multiple tools and utilities and are evaluated.

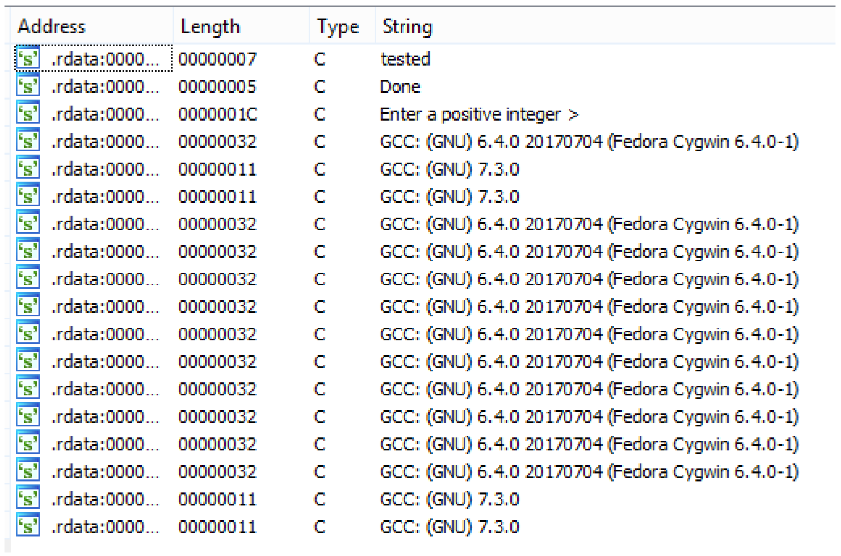
5.4.1 Before Obfuscation Results

In the evaluation process, the multiple tools were used for the process of collecting results and comparison of results between different tool against the same utilities. In this heading, results from IDA Pro are presented. Figure 10 is the output of the string utility executed on the executable of code labeled CUT1 and it clearly shows the string constant "tested" and "Enter any positive Integer" in it. That reflects the situation that if this string is an important key then it could be extracted by simple analysis. Same string is also visible in the HEX Editor Utility of IDA Pro in Figure 12. Whereas, when CUT2 was passed through the same process as expected we have all constants i.e. tested, enter any positive integer and as well as done. These two string constants can be seen in IDA Pro String utility output in Figure 11 and the same thing is visible in the HEX Editor utility of IDA Pro in Figure 13. So both of the Codes under test shows the same results as expected i.e. showing the string constants present in it.



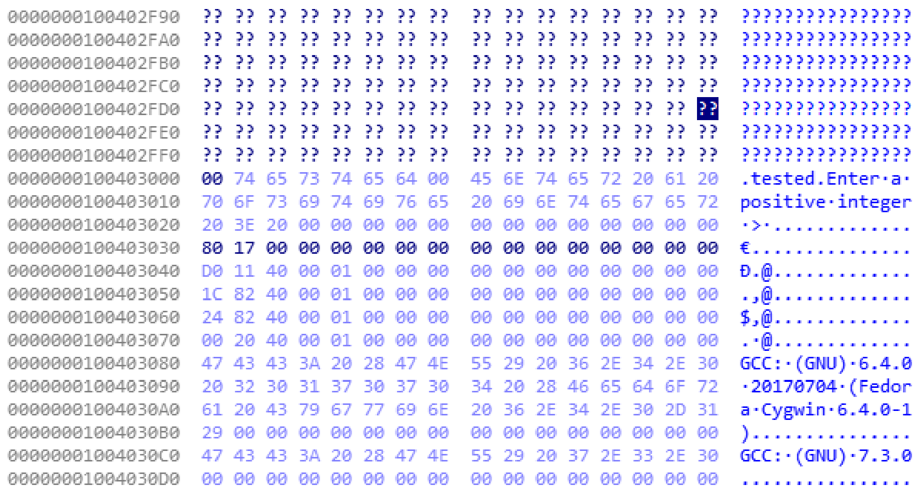
Address	Length	Type	String
.rdata:0000...	00000007	C	tested
.rdata:0000...	0000001C	C	Enter a positive integer >
.rdata:0000...	00000032	C	GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...	00000011	C	GCC: (GNU) 7.3.0
.rdata:0000...	00000011	C	GCC: (GNU) 7.3.0
.rdata:0000...	00000032	C	GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...	00000032	C	GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...	00000032	C	GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...	00000032	C	GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...	00000032	C	GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...	00000032	C	GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...	00000032	C	GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...	00000032	C	GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...	00000032	C	GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...	00000032	C	GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...	00000011	C	GCC: (GNU) 7.3.0
.rdata:0000...	00000011	C	GCC: (GNU) 7.3.0

Figure 10: IDA Pro String Utility Output - CUT1 - Before Obfuscation



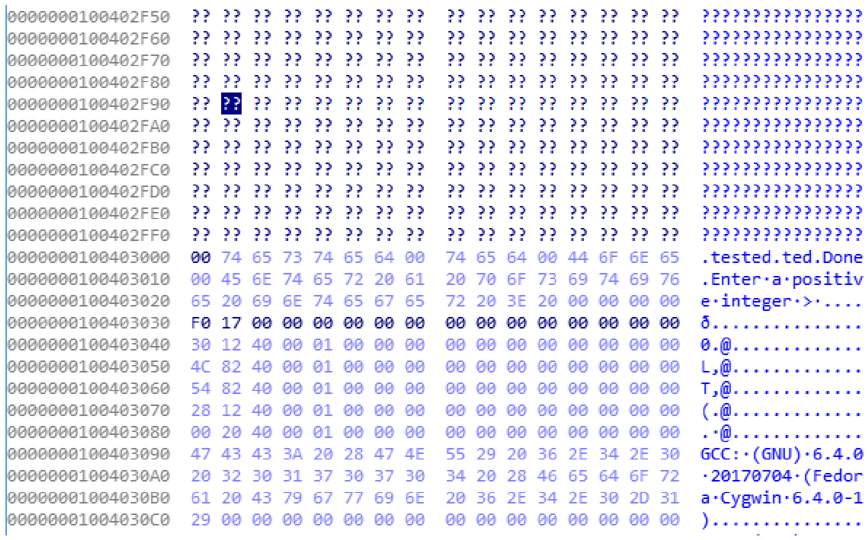
Address	Length	Type	String
.rdata:0000...	00000007	C	tested
.rdata:0000...	00000005	C	Done
.rdata:0000...	0000001C	C	Enter a positive integer >
.rdata:0000...	00000032	C	GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...	00000011	C	GCC: (GNU) 7.3.0
.rdata:0000...	00000011	C	GCC: (GNU) 7.3.0
.rdata:0000...	00000032	C	GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...	00000032	C	GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...	00000032	C	GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...	00000032	C	GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...	00000032	C	GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...	00000032	C	GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...	00000032	C	GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...	00000032	C	GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...	00000032	C	GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...	00000032	C	GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...	00000011	C	GCC: (GNU) 7.3.0
.rdata:0000...	00000011	C	GCC: (GNU) 7.3.0

Figure 11: IDA Pro String Utility Output - CUT2 - Before Obfuscation



```
0000000100402F90  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100402FA0  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100402FB0  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100402FC0  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100402FD0  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100402FE0  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100402FF0  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100403000  00 74 65 73 74 65 64 00  45 6E 74 65 72 20 61 20  .tested.Enter·a·
0000000100403010  70 6F 73 69 74 69 76 65  20 69 6E 74 65 67 65 72  positive·integer
0000000100403020  20 3E 20 00 00 00 00 00  00 00 00 00 00 00 00 00  ·>·.............
0000000100403030  80 17 00 00 00 00 00 00  00 00 00 00 00 00 00 00  €...............
0000000100403040  D0 11 40 00 01 00 00 00  00 00 00 00 00 00 00 00  Ð.@.............
0000000100403050  1C 82 40 00 01 00 00 00  00 00 00 00 00 00 00 00  .,@.............
0000000100403060  24 82 40 00 01 00 00 00  00 00 00 00 00 00 00 00  $,@.............
0000000100403070  00 20 40 00 01 00 00 00  00 00 00 00 00 00 00 00  .·@.............
0000000100403080  47 43 43 3A 20 28 47 4E  55 29 20 36 2E 34 2E 30  GCC:·(GNU)·6.4.0
0000000100403090  20 32 30 31 37 30 37 30  34 20 28 46 65 64 6F 72  ·20170704·(Fedor
00000001004030A0  61 20 43 79 67 77 69 6E  20 36 2E 34 2E 30 2D 31  a·Cygwin·6.4.0-1
00000001004030B0  29 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  )...............
00000001004030C0  47 43 43 3A 20 28 47 4E  55 29 20 37 2E 33 2E 30  GCC:·(GNU)·7.3.0
00000001004030D0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

Figure 12: IDA Pro Hex Editor Output - CUT1 - Before Obfuscation

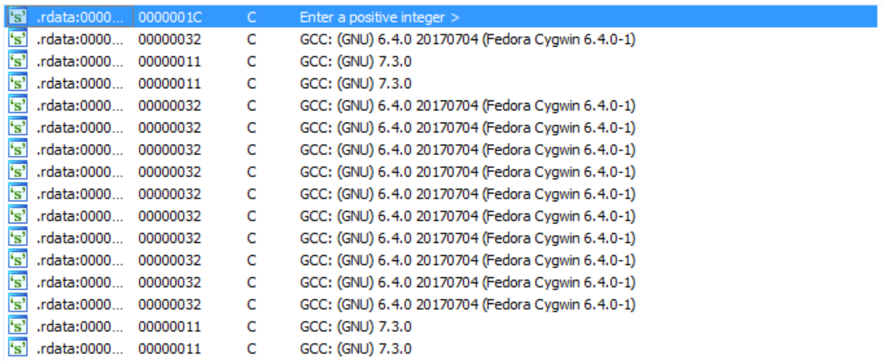


```
0000000100402F50  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100402F60  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100402F70  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100402F80  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100402F90  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100402FA0  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100402FB0  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100402FC0  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100402FD0  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100402FE0  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100402FF0  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100403000  00 74 65 73 74 65 64 00  74 65 64 00 44 6F 6E 65  .tested.ted.Done
0000000100403010  00 45 6E 74 65 72 20 61  20 70 6F 73 69 74 69 76  .Enter·a·positiv
0000000100403020  65 20 69 6E 74 65 67 65  72 20 3E 20 00 00 00 00  e·integer·>·....
0000000100403030  F0 17 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ð...............
0000000100403040  30 12 40 00 01 00 00 00  00 00 00 00 00 00 00 00  0.@.............
0000000100403050  4C 82 40 00 01 00 00 00  00 00 00 00 00 00 00 00  L,@.............
0000000100403060  54 82 40 00 01 00 00 00  00 00 00 00 00 00 00 00  T,@.............
0000000100403070  28 12 40 00 01 00 00 00  00 00 00 00 00 00 00 00  (.@.............
0000000100403080  00 20 40 00 01 00 00 00  00 00 00 00 00 00 00 00  .·@.............
0000000100403090  47 43 43 3A 20 28 47 4E  55 29 20 36 2E 34 2E 30  GCC:·(GNU)·6.4.0
00000001004030A0  20 32 30 31 37 30 37 30  34 20 28 46 65 64 6F 72  ·20170704·(Fedor
00000001004030B0  61 20 43 79 67 77 69 6E  20 36 2E 34 2E 30 2D 31  a·Cygwin·6.4.0-1
00000001004030C0  29 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  )...............
```

Figure 13: IDA Pro Hex Editor Output - CUT2 - Before Obfuscation

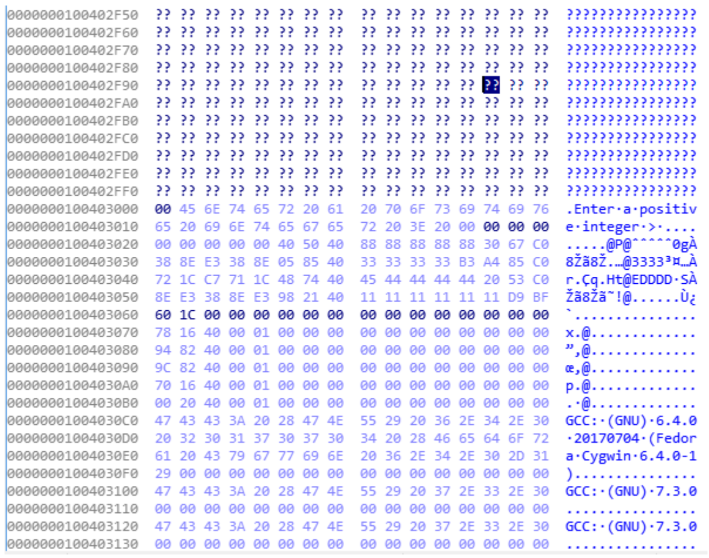
5.4.2 After Obfuscation Results

Once the results are collected before the obfuscation process, both of the code under test CUT1 and CUT2 were passed through the obfuscation process separately to get two obfuscated codes. Let's name the obfuscated code of CUT1 as OCO1 Obfuscated code obtained 1 and obfuscated code of CUT2 as OCO2 Obfuscated code obtained 2. During the obfuscation the required functions and codes are entered by the application itself, hence no modifications are to be made. So, we executed the codes OCO1 and OCO2 got from the obfuscation process and executables formed from these codes were analyzed. First, we analyzed the OCO1, in Figure 14 you can see the string “tested” is not visible and is obfuscated. The string is not shown by the IDA Pro String Utility. The same thing can be verified by the IDA Pro Hex Editor output for the OCO1, String “tested” is not visible.



```
.rdata:0000...  0000001C  C  Enter a positive integer >
.rdata:0000...  00000032  C  GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...  00000011  C  GCC: (GNU) 7.3.0
.rdata:0000...  00000011  C  GCC: (GNU) 7.3.0
.rdata:0000...  00000032  C  GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...  00000032  C  GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...  00000032  C  GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...  00000032  C  GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...  00000032  C  GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...  00000032  C  GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...  00000032  C  GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...  00000032  C  GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...  00000032  C  GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...  00000032  C  GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...  00000011  C  GCC: (GNU) 7.3.0
.rdata:0000...  00000011  C  GCC: (GNU) 7.3.0
```

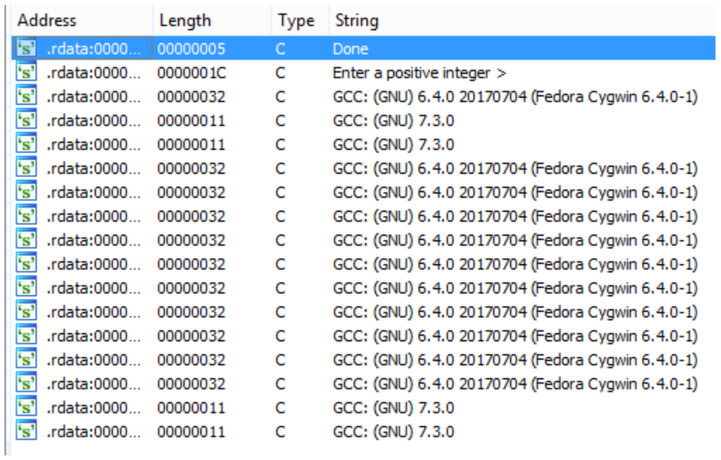
Figure 14: IDA Pro String Utility Output - OCO1 – After Obfuscation



```
0000000100402F50  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100402F60  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100402F70  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100402F80  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100402F90  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100402FA0  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100402FB0  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100402FC0  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100402FD0  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100402FE0  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100402FF0  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100403000  00 45 6E 74 65 72 20 61  20 70 6F 73 69 74 69 76  .Enter·a·positiv
0000000100403010  65 20 69 6E 74 65 67 65  72 20 3E 20 00 00 00 00  e·integer·>·....
0000000100403020  00 00 00 00 00 40 50 40  88 88 88 88 88 30 67 C0  .....@P@ˆˆˆˆˆ0gÀ
0000000100403030  38 8E E3 38 8E 05 85 40  33 33 33 33 B3 A4 85 C0  8Žã8Ž.…@3333³¤…À
0000000100403040  72 1C C7 71 1C 48 74 40  45 44 44 44 44 20 53 C0  r.Çq.Ht@EDDDD·SÀ
0000000100403050  8E E3 38 8E E3 98 21 40  11 11 11 11 11 11 D9 BF  Žã8Žã˜!@......Ù¿
0000000100403060  60 1C 00 00 00 00 00 00  00 00 00 00 00 00 00 00  `...............
0000000100403070  78 16 40 00 01 00 00 00  00 00 00 00 00 00 00 00  x.@.............
0000000100403080  94 82 40 00 01 00 00 00  00 00 00 00 00 00 00 00  ”‚@.............
0000000100403090  9C 82 40 00 01 00 00 00  00 00 00 00 00 00 00 00  œ‚@.............
00000001004030A0  70 16 40 00 01 00 00 00  00 00 00 00 00 00 00 00  p.@.............
00000001004030B0  00 20 40 00 01 00 00 00  00 00 00 00 00 00 00 00  .·@.............
00000001004030C0  47 43 43 3A 20 28 47 4E  55 29 20 36 2E 34 2E 30  GCC:·(GNU)·6.4.0
00000001004030D0  20 32 30 31 37 30 37 30  34 20 28 46 65 64 6F 72  ·20170704·(Fedor
00000001004030E0  61 20 43 79 67 77 69 6E  20 36 2E 34 2E 30 2D 31  a·Cygwin·6.4.0-1
00000001004030F0  29 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  )...............
0000000100403100  47 43 43 3A 20 28 47 4E  55 29 20 37 2E 33 2E 30  GCC:·(GNU)·7.3.0
0000000100403110  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0000000100403120  47 43 43 3A 20 28 47 4E  55 29 20 37 2E 33 2E 30  GCC:·(GNU)·7.3.0
0000000100403130  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

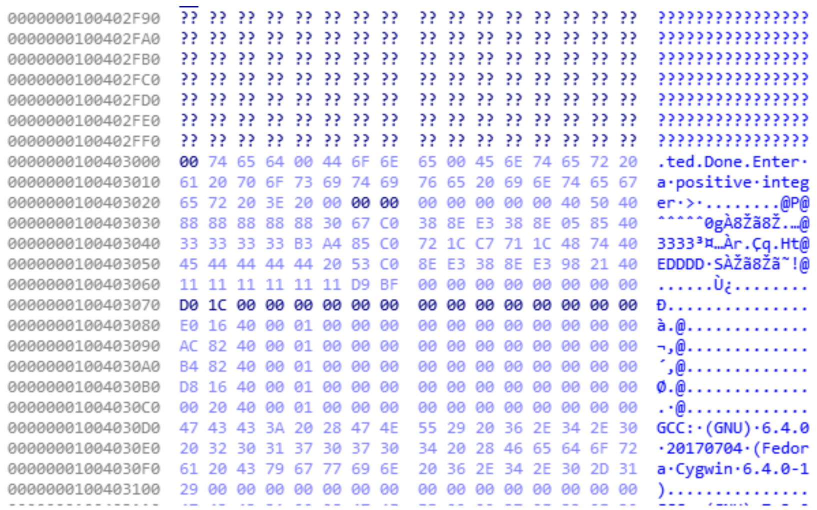
Figure 15 IDA Pro Hex Editor Output - OCO1 - After Obfuscation

Similarly, when we analyzed the OCO2 using both String utility and Hex Editor, we can see in Figure 16 and Figure 17 that string to be obfuscated i.e. tested cannot be seen in either of the images and hence is successfully obfuscated remaining string constants are present and are visible in both Figure 16 and Figure 17.



Address	Length	Type	String
.rdata:0000...	00000005	C	Done
.rdata:0000...	0000001C	C	Enter a positive integer >
.rdata:0000...	00000032	C	GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...	00000011	C	GCC: (GNU) 7.3.0
.rdata:0000...	00000011	C	GCC: (GNU) 7.3.0
.rdata:0000...	00000032	C	GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...	00000032	C	GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...	00000032	C	GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...	00000032	C	GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...	00000032	C	GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...	00000032	C	GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...	00000032	C	GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...	00000032	C	GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...	00000032	C	GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...	00000032	C	GCC: (GNU) 6.4.0 20170704 (Fedora Cygwin 6.4.0-1)
.rdata:0000...	00000011	C	GCC: (GNU) 7.3.0
.rdata:0000...	00000011	C	GCC: (GNU) 7.3.0

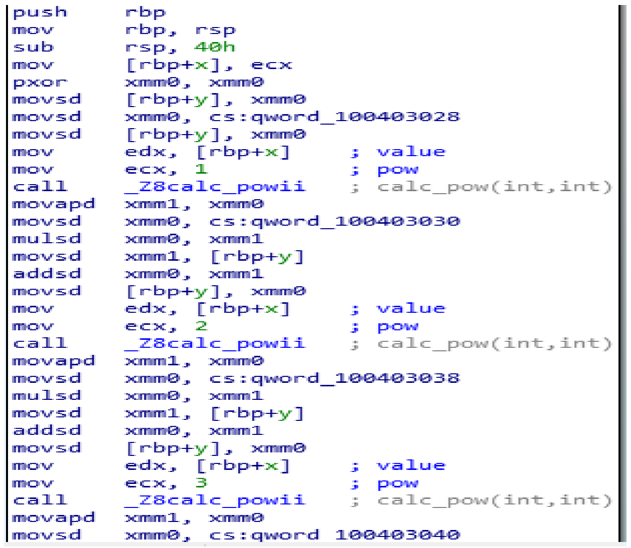
Figure 16: IDA Pro Hex Editor Output - OCO2 - After Obfuscation



```
0000000100402F90  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100402FA0  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100402FB0  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100402FC0  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100402FD0  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100402FE0  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100402FF0  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
0000000100403000  00 74 65 64 00 44 6F 6E  65 00 45 6E 74 65 72 20  .ted.Done.Enter·
0000000100403010  61 20 70 6F 73 69 74 69  76 65 20 69 6E 74 65 67  a·positive·integ
0000000100403020  65 72 20 3E 20 00 00 00  00 00 00 00 00 40 50 40  er·>·........@P@
0000000100403030  88 88 88 88 88 30 67 C0  38 8E E3 38 8E 05 85 40  ^^^^^0gÀ8Žã8Ž.…@
0000000100403040  33 33 33 33 B3 A4 85 C0  72 1C C7 71 1C 48 74 40  3333³¤…Àr.Çq.Ht@
0000000100403050  45 44 44 44 44 20 53 C0  8E E3 38 8E E3 98 21 40  EDDDD·SÀŽã8Žã˜!@
0000000100403060  11 11 11 11 11 11 D9 BF  00 00 00 00 00 00 00 00  ......Ù¿........
0000000100403070  D0 1C 00 00 00 00 00 00  00 00 00 00 00 00 00 00  Ð...............
0000000100403080  E0 16 40 00 01 00 00 00  00 00 00 00 00 00 00 00  à.@.............
0000000100403090  AC 82 40 00 01 00 00 00  00 00 00 00 00 00 00 00  ¬‚@.............
00000001004030A0  B4 82 40 00 01 00 00 00  00 00 00 00 00 00 00 00  ´‚@.............
00000001004030B0  D8 16 40 00 01 00 00 00  00 00 00 00 00 00 00 00  Ø.@.............
00000001004030C0  00 20 40 00 01 00 00 00  00 00 00 00 00 00 00 00  .·@.............
00000001004030D0  47 43 43 3A 20 28 47 4E  55 29 20 36 2E 34 2E 30  GCC:·(GNU)·6.4.0
00000001004030E0  20 32 30 31 37 30 37 30  34 20 28 46 65 64 6F 72  ·20170704·(Fedor
00000001004030F0  61 20 43 79 67 77 69 6E  20 36 2E 34 2E 30 2D 31  a·Cygwin·6.4.0-1
0000000100403100  29 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  )...............
```

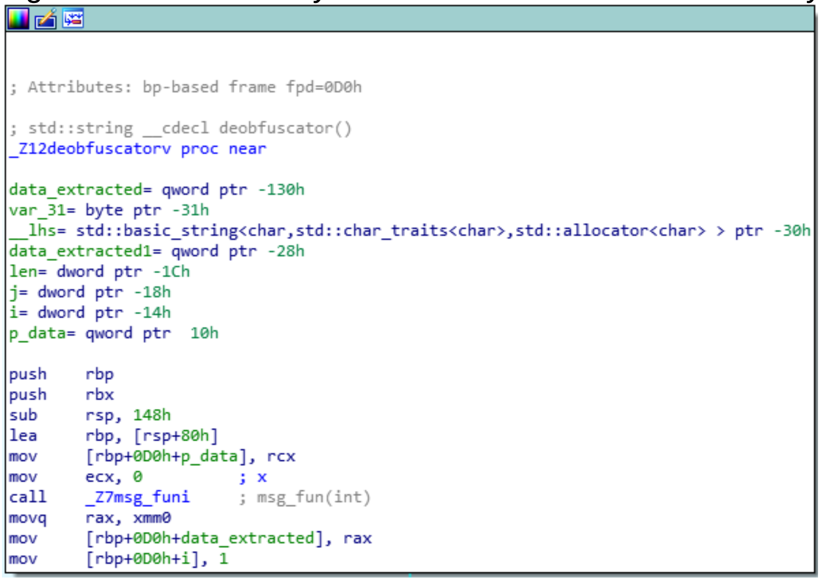
Figure 17: IDA Pro Hex Editor - OCO2 - After Obfuscation

It was of greater concern that code generated might not leak any information about the polynomial coefficients, that may lead to data visibility to the attacker. Hence in Figure 18, you can see the disassembly of the polynomial function code generated by the application and nothing useful can be extracted from this disassembly. Similar is the case for the disassembly for de-obfuscator function as shown in Figure 19. Nothing useful can be extracted from that disassembly either.



```
push    rbp
mov     rbp, rsp
sub     rsp, 40h
mov     [rbp+x], ecx
pxor    xmm0, xmm0
movsd   [rbp+y], xmm0
movsd   xmm0, cs:qword_100403028
movsd   [rbp+y], xmm0
mov     edx, [rbp+x]      ; value
mov     ecx, 1            ; pow
call    _Z8calc_powii     ; calc_pow(int,int)
movapd  xmm1, xmm0
movsd   xmm0, cs:qword_100403030
mulsd   xmm0, xmm1
movsd   xmm1, [rbp+y]
addsd   xmm0, xmm1
movsd   [rbp+y], xmm0
mov     edx, [rbp+x]      ; value
mov     ecx, 2            ; pow
call    _Z8calc_powii     ; calc_pow(int,int)
movapd  xmm1, xmm0
movsd   xmm0, cs:qword_100403038
mulsd   xmm0, xmm1
movsd   xmm1, [rbp+y]
addsd   xmm0, xmm1
movsd   [rbp+y], xmm0
mov     edx, [rbp+x]      ; value
mov     ecx, 3            ; pow
call    _Z8calc_powii     ; calc_pow(int,int)
movapd  xmm1, xmm0
movsd   xmm0, cs:qword_100403040
```

Figure 18: Data -> Polynomial -> Function -> Disassembly



```
; Attributes: bp-based frame fpd=0D0h

; std::string __cdecl deobfuscator()
_Z12deobfuscatorv proc near

data_extracted= qword ptr -130h
var_31= byte ptr -31h
__lhs= std::basic_string<char,std::char_traits<char>,std::allocator<char> > ptr -30h
data_extracted1= qword ptr -28h
len= dword ptr -1Ch
j= dword ptr -18h
i= dword ptr -14h
p_data= qword ptr  10h

push    rbp
push    rbx
sub     rsp, 148h
lea     rbp, [rsp+80h]
mov     [rbp+0D0h+p_data], rcx
mov     ecx, 0          ; x
call    _Z7msg_funi     ; msg_fun(int)
movq    rax, xmm0
mov     [rbp+0D0h+data_extracted], rax
mov     [rbp+0D0h+i], 1
```

Figure 19: De-obfuscator Function Disassembly

The last thing that was evaluated is when the same string is obfuscated multiple times from the obfuscation application what is the similarity factor between the coefficients of the polynomial. So the string obfuscated “tested” is passed to obfuscation application 5 times and results are shown in Table 1. As we can see that none of the five runs generate the different polynomial coefficients hence a different code. So for a company which sends the same software to different clients can obfuscate from our application and send different copies to everyone.

Table 1: Randomization Similarity Checking Table

Coefficients Location	1st Run	2nd Run	3rd Run	4th Run	5th Run
X^0	63.0	66.0	64.0	66.0	63.0
X^1	-1202.263095	336.190476	1232.126190	337.357142	-1197.596428
X^2	3005.8697420	-582.891666	-2669.111111	-585.7222222	2997.075297
X^3	-2665.160416	429.298611	2218.231944	431.866666	-2660.372916
X^4	1152.638715	-162.583333	-899.923611	-163.722222	1152.290798
X^5	-271.1625	33.361111	189.815277	33.625000	-271.6
X^6	35.4246527	-3.5249999	-19.965277	-3.555555	35.566319
X^7	-2.413988	0.1498015	0.826587	0.151190	-2.430654
X^8	0.0668898	--	--	--	0.0675843

CHAPTER 6

6 CONCLUSIONS

The software is one of the major commercial products that exist in the world today. Due to a large variety of reverse engineering tools and financial benefits, the number of attacks on software are exponentially increased. There are many solutions present for the functions obfuscation and security but a lesser number of solutions exist for data portion of the software, In the obfuscation process we designed and formulated a way to disperse the data present in the executable. So, by using the analysis tools, it will be much harder for the attacker to extract anything useful present as data constants from the executable.

6.1 Future Directions

In this research thesis, we developed a generic data obfuscation that can be widely applied but for the POC we have implemented it with some limitations i.e. one string constant and C++ as input and output language. This concept can be extended to the next level. Here are some of the recommendations

- The POC implemented is for one string constant and can be extended to multiple strings in a single code. Then it can also be enhanced for different data types or data structures.
- We faced the limitation due to significant figures precision that we can obfuscate only 6-8 characters' maximum, this limitation can be removed. This implementation can be enhanced to the obfuscation of any length of the polynomial.
- Our Algorithm can be enhanced to different language-specific derivatives.

7 REFERENCES

[1] T. Fraser, L. Badger, and M. Feldman, "Hardening COTS software with generic software wrappers," in *IEEE Symposium on Security and Privacy*, Oakland, USA, 1999.

[2] T. C. Sander T., " On Software Protection via Function Hiding," in *Information Hiding. IH 1998. Lecture Notes in Computer Science*, Berlin, Heidelberg, 1998.

[3] T. H. a. C. Wenz, "DRM under attack: weakness in existing systems," *Digital Rights Management,* pp. 206-223, 2003.

[4] P. Marwedel, "Embedded system design: Embedded systems foundations of cyber-physical systems.," in *Springer Science & Business Media,* 2010.

[5] K. C. A. R. F. P. S. K. T. S. M. D. K. B. A. D. Koscher, "Experimental security analysis of a modern automobile," *Symposium on Security and Privacy,* pp. 447-462, 2010.

[6] S. C. M. M. R. a. N. J. Alam, "Interoperability of security-enabled Internet of Things," *Wireless Personal Communications,* vol. 61, p. 567–586, 2011.

[7] G. H. K. R. A. P. Konstantinos Fysarakis, "Embedded Systems Security Challenges," 2014.

[8] B. Wyseur, "White Box Cryptography (Ph.D. Thesis)," Katholieke Universiteit, Heverlee, Belgium, 2009.

[9] P. E. H. J. P. C. V. S. Chow, "White-Box Cryptography and an AES Implementation," in *Selected Areas in Cryptography*, 2003.

[10] N. A. A. G. A. Irfan Azhar, "Keeping Secret Keys Secret in Open systems," in *International Conference on Open Source systems and technologies*, Lahore, Pakistan, 2014.

[11] A. R. a. C. S. Ravi, "Tamper Resistance mechanisms for secure embedded systems," in a *17th International conference on VLSI design*, 2004.

[12] J. H. J. a. j. C Wang, "Software Temper Resistance: Obstructing static analysis of programs," 2000.

[13] S. D. Cullen Linn, "Obfuscation of executable code to improve resistance to static disassembly," *Proceedings of the 10th ACM conference on Computer and communications security,* pp. 290-299, 2003.

[14] Y. M. S. a. A. M. T Ogiso, "Software Obfuscation on a theoretical basis and its implementation," in *IEEE Trans. Fundamentals*, 2003.

[15] J. a. Y. W. Chan, "Advanced Obfuscation Techniques for Java bytecode," *Jornal of Systems and software,* vol. 71, pp. 1-10, 2001.

[16] J. N. a. C. Collberg, Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection, Pearson Education, 2009.

[17] S. D. S. N. H. J. A. Halderman, "Lest we remember: Cold Bot attacks on encryption keys," *17th USENIX Security Symposium,* vol. 52, no. 5, pp. 388-397, 2008.

[18] M. h. a. S. Taylor, "Memory Encryption: A survey of existing techniques," *ACM Computing Surveys,* vol. 46, no. 4, p. 53, 2014.

[19] T. S. Müller, "Frost," in *International Conference on Applied Cryptography and Network Security*, NY, 2013.

[20] S. Skorobogatov, "Semi-invasive attacks - a new approach to hardware security analysis," University of Cambridge, April 2005.

[21] C. J. Jansen, "Investigations on Non-linear Streamcipher Systems: Construction and Evaluation Methods," Technical University of Delft, 1989.

[22] Q. W. W. T. A.-F. S. Jian Jun Hu, "A key Hiding based Software Encryption Protection Scheme," in *IEEE 13th International Conference on Communication Technology*, Jinan, China, 2011.

[23] C. C. a. C. Thompson, "Watermarking, Tamper-proofing and Obfuscation - Tools for Software protection," Computer Science Department, University of Auckland, Auckland, 2000.

[24] M. F. Z. Long Zheng Cai, "Security challenges for open embedded systems," in *IEEE, International Conference on Engineering Technology and Technopreneurship (ICE2T),* Kuala Lumpur, Malaysia, 2017.

[25] D. M. a. L. B. Z. Papp, "Embedded Systems Security: Threats, Vulnerabilities and Attack Taxonomy," in *13th Annual Conference on Privacy, Security and Trust*, Izmir, Turkey, 2015.

[26] D. F. a. C. Castelluccia, "Defending Embedded Systems against control flow attacks," in *1st ACM Workshop on the secure execution of untrusted code*, Newyork, 2009.

[27] N. S. a. E. McCluskey, "Control-flow Checking Using Watch Dog Assists and Extended-Precision Checksum," *IEEE Trans. Comput.,* vol. 39, pp. 554-559, 1990.

[28] W. a. Y. S.Das, "A Fine-grained control flow integrity approach against runtime memory attacks for embedded systems," in *IEEE Trans. VLSI Systems*, 2016.

[29] S. P. a. T. Wolf, "Embedded Systems Security - An Overview," *Des. Autom. Embed. Syst,* vol. 12, pp. 173-183, 2008.

[30] ARM, "TrustZone," 2016. [Online]. Available: www.arm.com/products/processors/technologies/trustzone/.

[31] H. L. a. E. B. M. Rahmatian, "Hardware-Assisted detection of malicious software in embedded systems," *IEEE Embedded System Letter,* vol. 4, pp. 94-97, 2012.

[32] H. Gebotys, "Low Energy Security optimization in Embedded Cryptographic Systems," in *International Conference on Hardware/Software Co-Design and System Synthesis,* Washington DC, 2004.

[33] B. P. a. F. M. Feng, "Embedded System for sensor communication and security," *IET Information Security,* vol. 2, pp. 111-121, 2012.

[34] M. A. a. N. M.M.kermani, "Emerging Frontiers in Embedded Security," in *26th International*

conference on VLSI design, 2013.

[35] J. L. a. R. Sujit Sanjeev, "Protecting Cryptographic keys on client platforms using virtualization and raw disk image access," in *IEEE International Conference on privacy security risk and trust*, 2011.

[36] T. J. a. E.J.Weyuker, "The Distribution of Faults in a large industrial software system," in *ACM, SIGSOFT International Symposium on Software testing and analysis*, 2002.

[37] TrueCrypt, [Online]. Available: www.truecrypt.com.

[38] Microsoft, [Online]. Available: www.windows.microsoft.com/en-US/windows7/products/features/bitlocker.

[39] S. S. E. D. Nan Li, "Secure Key Storage Using State Machines," in *IEEE 43rd International Symposium on Multiple-Valued Logic*, 2013.

[40] H. C. v. Tilborg, Encyclopedia of cryptography and security, NJ USA: Springer-Verlag New York, 2005.

[41] D. B. RAFAEL COSTA, "Methods to Protect Cryptographic Keys on Safety-Critical Systems," *WSEAS TRANSACTIONS on INFORMATION SCIENCE and APPLICATIONS,* vol. 12, 2015.

[42] B. N. B. C. a. Z. H. H. Dong, "Automatic train control system development and simulation for high-speed railways," *IEEE Circuits and Systems Magazine,* vol. 10, pp. 6-18, 2010.

[43] M. S. N. B. A. G. E. A. M. S. S. F. A. H. A. Akhunzada, " Man-At-The-End attacks: Analysis, taxonomy, human aspects, motivation and future directions," *J. Netw. Comput. Appl.,* vol. 48, pp. 44-57, 2015.

[44] S. L. Kinney, Trusted Platform Module Basics: Using TPM in Embedded Systems, Newnes, 2006.

[45] Z. M. L. B. Dorottya Papp, "Embedded Systems Security: Threats, Vulnerabilities, and Attack Taxonomy," in *Thirteenth Annual Conference on Privacy, Security, and Trust (PST),* 2015.

[46] R. Langner, "Stuxnet: Dissecting a cyber warfare weapon," *Security and Privacy,* vol. 9, no. 3, pp. 49-51, 2011.

[47] D. N. Serpanos and A. G. Voyiatzis, "Security challenges in embedded systems," *ACM Transactions on Embedded Computing Systems (TECS),* vol. 12, no. 1, p. 66, 2013.

[48] X. Z.-a. T. X. Z. L. Z. Zheng, "Cryptographic key protection against FROST for mobile devices," *Cluster Computing,* vol. 20, no. 3, p. 2393–2402, 2017.

[49] B. Gupta, Handbook of Research on Modern Cryptographic Solutions forComputer andCyber Security, Hershey: IGI Global, 2016.

[50] M. S. C. J. Y. S. M.-W. L. K. Z. C. D. Xu, "Toward engineering a secure Android ecosystem: a survey of existing techniques," in *ACM Computing Survey*, 2016.

[51] R. B. C. S. Carbone, "An in-depth analysis of the cold boot attack," Defence Research and Development Canada, Canada, 2011.

[52] G. K. G. B. R. R. a. R. S. Carl, "Denial-of-service attack detection techniques," *IEEE Internet Computing,* vol. 10, no. 1, pp. 82-89, 2006.

[53] M. R. a. S. K. M. S. Doomun, "Analytical comparison of cryptographic techniques for resource-

constrained wireless security," *International Journal of Network Security*, vol. 9, no. 1, pp. 82-94, 2009.

[54] C. L. M. N. R. a. P. A. Kuo, "Message-in-a-bottle: User-friendly and secure key deployment," in *5th International Conference on Embedded Networked Sensor Systems*, Sydney, Australia, 2007.

[55] C. Kothari, Research methodology: methods and techniques, New Age International, 2009.