Code Obfuscation Techniques for Software Protection

Author

Marrium Aziz Khan

FALL 2015-MS-15(CSE) 00000117433

MS-15 (CSE)

Supervisor

Dr. Wasi Haider Butt

Department of Computer Engineering

College of Electrical & Mechanical Engineering

National University of Science and Technology, Islamabad

January, 2019

Code Obfuscation Techniques for Software Protection

Author

Marrium Aziz Khan

FALL 2015-MS-15(CSE) 00000117433

A thesis submitted in partial fulfillment of the requirements for the degree of MS Software Engineering

Supervisor

Dr. Wasi Haider Butt

Thesis Supervisor's Signature: ___________________________________

Department of Computer Engineering

College of Electrical & Mechanical Engineering

National University of Science and Technology, Islamabad

January, 2019

Declaration

I certify that this research work titled *"Code Obfuscation techniques for Software Protection"* is my own work under the supervision of *"Dr. Wasi Haider Butt"*. 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

Marrium Aziz Khan

FALL 2015-MS-15(CSE) 00000117433

LANGUAGE CORRECTNESS CERTIFICATE

This thesis is free of typing, syntax, semantic, grammatical and spelling mistakes. Thesis is also according to the format given by the University for MS thesis work.

Signature of Student

Marrium Aziz Khan

FALL 2015-MS-15(CSE) 00000117433

Signature of Supervisor

Copyright Statement

Acknowledgements

Glory is to, Almighty Allah, The Most Merciful and The Most Beneficent, who bestowed upon me great source of knowledge and his blessings made me able to complete my task. Indeed I could have done nothing without Your priceless help and guidance. Whosoever helped me throughout the course of my thesis, whether my parents or any other individual was Your will, so indeed none be worthy of praise but You. The respectful and beloved Prophet (SAWW) who is role model for us.

I am profusely thankful to my beloved parents who raised me when I was not capable of walking and continued to support me throughout in every department of my life. I am copiously thankful to **Mr. Junaid Wasif** who is great strength of me.

I would also like to express special thanks to my supervisor **Dr. Wasi Haider Butt** who guided me throughout my thesis and kept my morale high and backed up with zest, constructive criticism and inspiring guidance. Also for his tremendous efforts to taught me Software Requirement Engineering and Web-based Software Development. I can safely say that I haven't learned any other engineering subject in such depth than the ones which he has taught.

I would also like to thank **Dr. Arsalan Shaukat** and **Dr. Muazzam A. Khattak** for being on my thesis guidance and evaluation committee. Their recommendations are very valued for improvement of the work. I appreciate their guidance throughout the whole thesis.

I would also like to pay special thanks to **Mr. Furqan Azhar** for his tremendous support and cooperation. Without his help I wouldn’t have been able to complete my thesis. I appreciate his patience and guidance throughout the whole thesis.

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

Dedicated to my exceptional parents and adored siblings whose tremendous support and cooperation led me to this wonderful accomplishment

Abstract

In the need of reinforcing the digital improvement, software application transforms into the most appealing bit of the propelled world. So its common trend to dispense mobile application in such structure that are isomorphic to the original code. Whereby using the process of reverse engineering, attackers identify the software behavior and extract the legitimate algorithms where traditional approaches like firewall, cryptography is not enough. Hence code obfuscation is one of the indispensable approaches which ratify intensified security by deterring the software code without changing the semantics and functionality. In this research, we proposed hybrid methodology to secure the mobile application by using Pro-guard, paranoid and Allatori. Detailed analysis and comparison of other technologies and their gaps are also part of this research. We provided an open source package for code obfuscation supporting android. By using Collberg's taxonomy we also improved potency, resilience, cost and stealth in mobile application by using hybrid approach of multiple algorithms. After carrying out test cases, we evaluated code optimization, apk size shrinking and statement to improve Collberg's taxonomy. We successfully obfuscated all java source files. Effective stress test has also been performed to de-obfuscate the code and unable to get back defined algorithm. Only resource files <xml> are left to be obfuscate, as main focus of this research was on managed code instead of scripting code.

Table of Contents

List of Figures

List of Tables

List of Equations

CHAPTER 1

INTRODUCTION

Chapter # 1: Introduction

The introductory chapter of thesis report presents general overview of topic. It is distributed into following sections. **Section 1.1** delivers the overview, **Section 1.2** elaborates the problem statement, **Section 1.3** provides the technical analysis and implementation overview of hybrid approach for code obfuscation techniques, whereas **Section 1.4** contains research contribution and finally **Section 1.5** contains thesis organization.

1.1. Overview:

In the necessity of strengthening the digital enhancement, software turns into the most alluring piece of the advanced world. IT industry burn through billions of dollars every year for averting from security attacks, i.e. tampering and malicious reverse engineering **[1, 2].** Furthermore mobile applications has been increasing greater ubiquity and malevolent reverse engineering is an intellectual property intrusion attack looked by applications, which points on comprehension and reusing the functionalities of a current item in the improvement of different items. Such attacks make it simpler for the attackers to extricate the exclusive calculations and information structures from applications so as to fuse them into their own projects, to reduce their effort on time and cost **[3].**

As a result of enormous application and improvement on digital advancements, the huge necessity for research on sanctuary and protection has been framed. Every organization has its own licensed revolution and it's a major purpose for using them to secure their information. There are multiple ways to secure information, i.e. Network protection, software legislation, piracy or infusion of malicious code and so on.

1.1.1. Network Protection:

The assurance to protect the devices at hardware level against vulnerabilities is network security. Many organizations define some policies and rules to verify the hardware security at network level from Man-in-the-middle attack as shown in **Figure 1.1 [4]**.

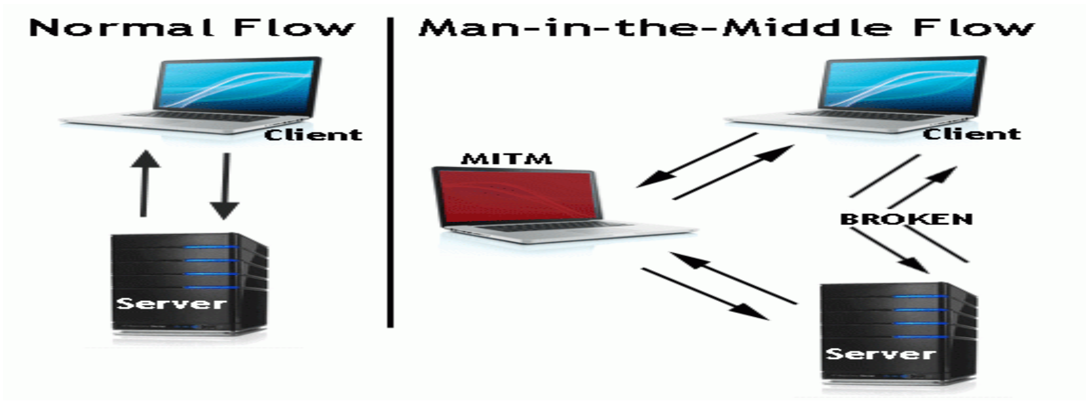


Figure1.1: Comparison of normal flow and MIMF (https://www.veracode.com/file/325)

It involves device to examine the network traffic and identify abnormal behavior from external world. It mostly secures access from external network by using many terminologies like firewall as shown in **Figure 1.2**, proxy servers (i.e. access to VPN), DMZs etc. **[5, 6]**. Some hardware security modules are also implemented cryptographic keys and encryptions **[7, 8]**. But all these security mechanisms will only secure against theft if the whole process is implemented on device. Uncertainty application is on commercial level it will always decompile the decrypted code in case if code is encrypted and executed in software by virtual machine **[9]**.

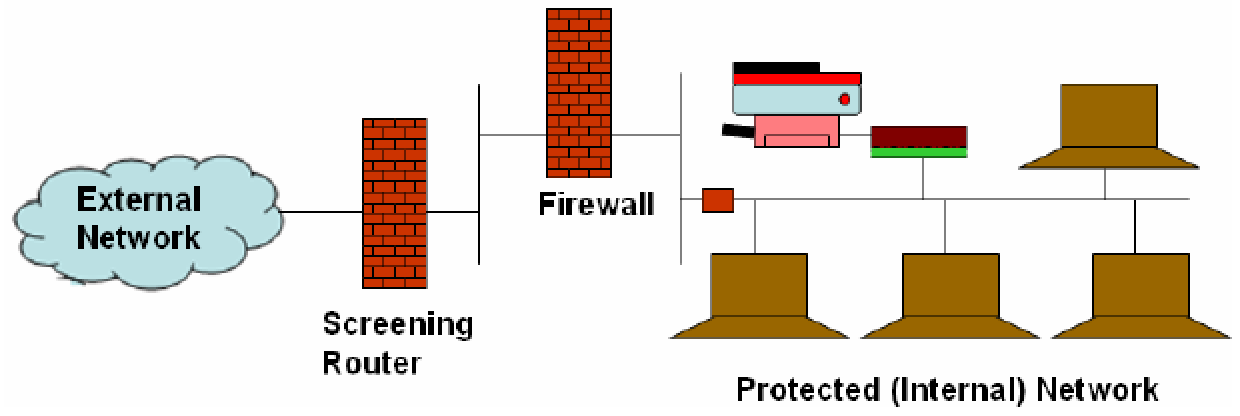


Figure1.2: Firewall Network (https://www.researchgate.net/figure/A-Layered-Firewall-Architecture_fig11_228527355)

1.1.2. Software Protection:

Encryption and firewalls are satisfactory approaches to alleviate the threads of attackers (Man-in-the-middle) who endeavor to disrupt into the targeted system **[10]**. Though, these approaches don't resist shielding targeted system when attacker is end client (Man-at-the-end) **[11, 12]**. Furthermore, whenever targeted system is business and proposed to be utilized on customer device then hardware protection isn't sufficient advance to take. So in any approach developer needs to secure the system with software mean also.

The expense of commercial applications drives a few contenders to fall back on intellectual property theft. If the application inside any item isn't secured in any capacity, contenders can essentially purchase a duplicate of the item, figure out the calculations actualized in that item and utilize these calculations inside their very own item. This term leads to software cracking and there are significant loses have been incurred for such commercial applications **[13]**.

There are two general way to ensure the protection of software application, in the legal mean **[14, 15]** and technical mean. In terms of technical means, proprietary software might be vulnerable with the anticipation that legitimate provisions will be sufficient to halt attackers. Nevertheless, legislation might be restricted to explicit geographical territory and not adjusted to the globe but software can be distributed globally. Subsequently, software developer must secure commercial application in technical means as well to make it troublesome for attackers who are utilizing application on their own devices. With the mean of accomplishing security at higher level is as yet not feasible subsequent to introducing security and cryptography networks. With respect to protect software application at code level, among all other techniques **code obfuscation** is staple implementation which endorses intensified security by averting code alterations **[16]**.

1.1.2.1. Obfuscation:

Obfuscation is a standout amongst the most utilized protection method to keep the cognizance of projects against MATE attacks. There are many methodologies to reach high level of obfuscation **[17].** Some of approaches are data obfuscation and code obfuscation. In the mean of securing application all obfuscation approaches comprises of program change methodologies that intend to make programs more unreservedly to examine, while preserving their semantics. Though, data obfuscation aims to stow away the structure of variables content and its usage, whereas code obfuscation changes format and data to render the code difficult to figure out and reverse engineer **[18]**.

1.1.2.2. Code Obfuscation:

Code obfuscation is method to alleviate process of reverser engineering by altering the original code without changing its semantics and hindering to understandable by human and fetching algorithm **[19, 20, 21, 22]**. Reverse engineering is the process that targets refurbishing a high level representation to investigate its structure and behavior as shown in **Figure 1.3 [23]**. Now day's applications are being developed and installed on client devices, where process of reverse engineering cannot be fully avoided. Consequently there is a need to secure end-to-end application against reverse engineering and raise bar against attackers to a magnitude where it does not merit contributing the assets to de-obfuscate application or extracting algorithm in any term. The proprietary systems are secured in software, secrecy causes, and duplicate assurance components are the most vital examples.

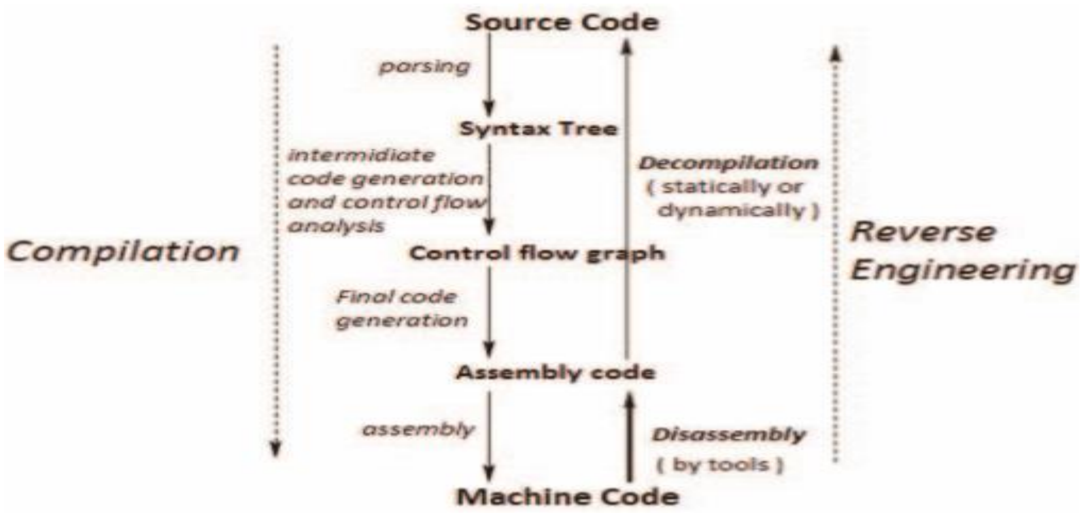


Figure1.3: Code Compilation Procedure (https://www.semanticscholar.org/paper/A-code-obfuscation-technique-to-prevent-reverse-Dalai-Das/0eac488df57ae4cc3c580a4c9ad25d097d322abb)

Some other important features like cryptographic algorithms i.e. AES that are implemented for trusted end-points by entrenching encryption and decryption and endure attacks in black-box context where attacker is not able to get awareness inside algorithms. Nonetheless this traditional end-to-end encryption is not enough to withstand attacker when it's easy to investigate the software while its execution. Hence code obfuscation is method to obscure the executions that are sensitive to client and used to mitigate the way to reverse engineering as shown in **Figure 1.4 [23].**

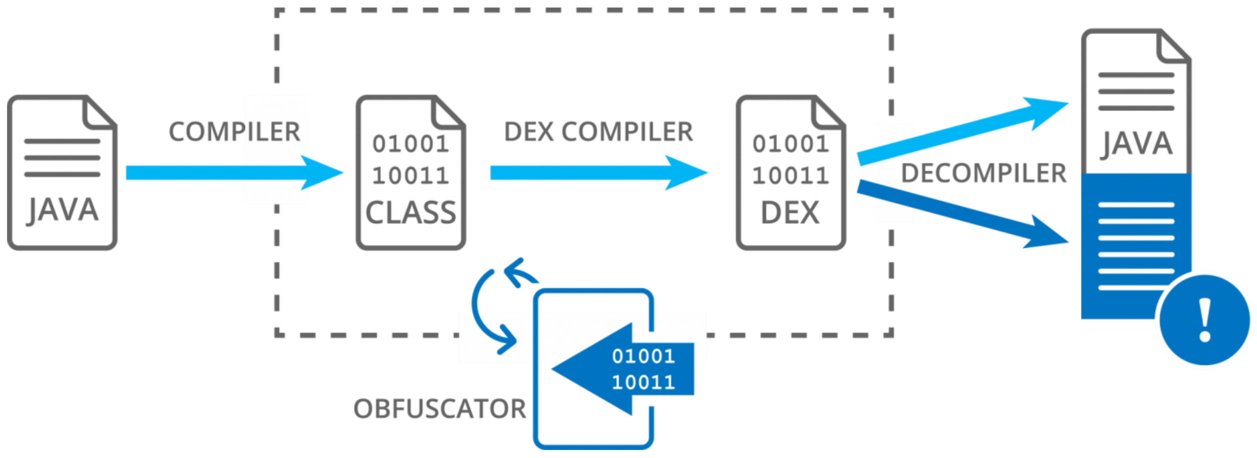


Figure1.4 : Java Compilation Flow (https://www.guardsquare.com/en/blog/decompiling-obfuscated-android-applications)

Collberg et al define the code obfuscation as transformation τ as transformation of program P and convert into obfuscated program P'. After obfuscation their behavior examined that P and P' are expecting to have same behavior but P' is difficult to understand by user after the process of reverse engineering. Code obfuscation shield against malicious alterations of application developed and software piracy as an assailant should initially know and comprehend the product before they can make indicated adjustments. The dimension of security in an application comprises of the required obstruction of the application against reverse engineering and tampering attacks.

1.2. Problem statement:

In the age of digital enrichment the mobile applications are in hand and its use is not restricted. In the establishment of digital environment, the malevolent Android applications are expanding hastily. Consequently many files and settings are being smashed by Android malware, moreover some unwanted application are also installed in result of this attack. So it is mandatory to identify such behavior. In result, many security methodologies are presented to hinder these malware. Traditional techniques like cryptography are not enough here to rescue from malicious activities. Subsequently some methodologies required to handle the Man-at-the-end attacks. Many techniques and algorithms are proposed to handle such troublesome however none of the algorithm obscures the code to grasp at maximum level of security. Moreover, some prior techniques aspect some issues regarding scalability. And none of any algorithm is open source to offer maximum obscurity.

1.3. Research Contribution:

The intact process is being conducted in the design-based research as explained in **Figure 1.5**. As the research premises, we perform brief analysis of practical problematic area and it is supposed to be very first step. Once problem is identified, then we suggest the solution for recognized problem in analogous to literature review for accomplishing experiments with numerous hypothetical frameworks for obfuscation. Hereby the literature review elaborates the background study of code obfuscation.

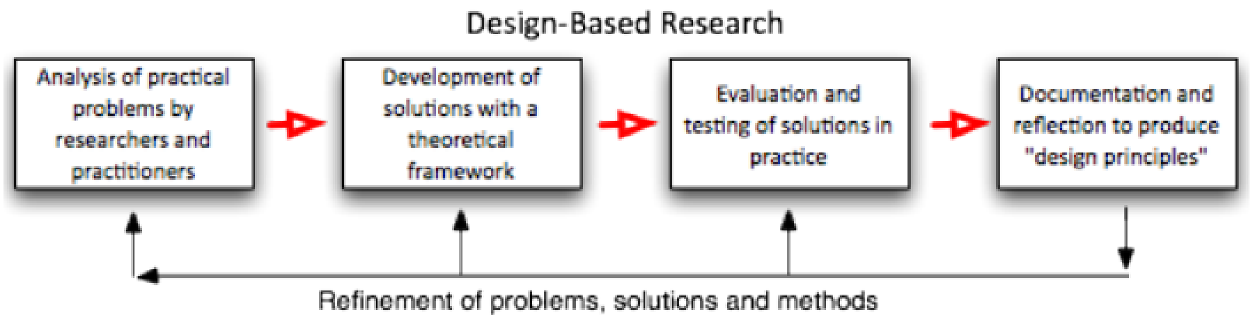

Design-Based Research
Analysis of practical problems by researchers and practitioners
Development of solutions with a theoretical framework
Evaluation and testing of solutions in practice
Documentation and reflection to produce "design principles"
Refinement of problems, solutions and methods

Figure1.5: Research Methodology Flow

The proposed approach describes hybrid obfuscation techniques for obscurity and software protection. As the part of evaluation and testing the solution, we perform the experiments and get the consequences from hybrid approach; firstly investigation is done on VOT4CS which is virtualization obfuscation tool for .NET. Depending upon its results, where we found no area of improvement restrictions on complier level and Roslyn code analyzer **[13]**, we moved to hybrid obfuscation for android. Here we used three techniques Pro-guard, Paranoid and Allatori. For the purpose of obfuscation, we enabled configuration settings to define our rules by using Pro-guard, Paranoid and Allatori. After obfuscating APK has been generated and decompiled in de-complier. Test cases have been performed to re-generate java files and analyzed the behavior of code as abstractly elaborated in **Figure 1.6**.

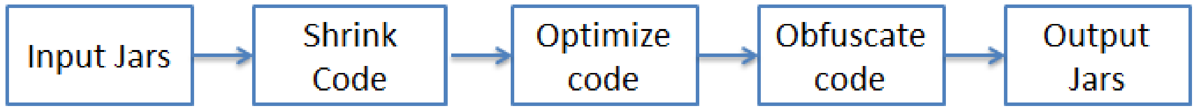


Figure1.6: Input / Output Flow of Obfuscated Code

This research is accomplished to provide open source hybrid obfuscator for Android Application in one hand. Three techniques are used to make challenging to understand the code without changing semantics and logic of program. The main aim of this research was improving obfuscation algorithm in that way no one could acquire the program in original form after de-compilation. Main contribution of this research to achieve application obfuscation, following were the steps followed in exact order as stated:

- Enable ProGuard Configuration Settings in our Android Studio Project. Android Studio provides a file that contains all of the essential configuration settings to run Android ProGuard obfuscation with application. We have used our own customized ProGuard file for better obfuscation than what default file provides us.
- To enable code optimization, we used a different default ProGuard configuration file. We modified the ‘proguard-android.txt’ value to ‘proguard-android-optimize.txt’ in our gradle file.
- For more advanced optimizations, we added certain filters in ProGuard rules file to improve code removal, simplification and merging.
- We have created a text file with random customized entries which is being used for obfuscation of variables, classes and packages. Otherwise, ProGuard uses default values to implement obfuscation.
- As ProGuard cannot obfuscate hard-coded string, we have used Paranoid library to achieve our purpose. To make Paranoid functional in your code, we have annotated all of our classes and activities with @Obfuscate which contain strings and need to be obfuscated. Every string literal and string constant defined in annotated classes or activities will be obfuscated.
- Sync and Build Project to allow Android Studio run all obfuscation, shrinking and optimization techniques
- Generate Signed APK
- Upload Signed APK on http://www.javadecompilers.com
- APK decompiled online through http://www.javadecompilers.com
- Java code extracted to verify code obfuscation, shrinking and optimization

1.4. Thesis organization:

Thesis has been prepared in multiple segments identified by distinct chapters as shown in **Figure 1.7. CHAPTER 1: INTRODUCTION** gives general overview and background study of selected topic. It explains problem statement, proposed solution, research contribution and thesis organization. **CHAPTER 2: LITERATURE REVIEW** embraces the recent knowledge including substantive findings, hypothetical and methodological contributions regarding code obfuscation. Initially it elaborated research methodology then it discussed theoretical frameworks for software protection and specifically code obfuscation excluding original experimental work. **CHAPTER 3: PROPOSED METHODOLOGY** presents overview of hybrid code obfuscation techniques, their detailed purpose of usage, limitation and causes of using hybrid approach. **CHAPTER 4: IMPLEMENTATION** describes the execution of proposed hybrid approach to obfuscate the Android application. Certain customized rules are defined to make improvements in all methodologies. Obfuscating the code with customized rules and generating APK by securing the original functionality of Application. De-compilation of APK has been done by Javadecompile. Ensuring that after de-compilation java files must not be in original form. **CHAPTER 5: RESULTS AND LIMITATIONS** gives results and comparison after applying hybrid obfuscation techniques on three different projects and analyze the end results for the sake of improvement in code obfuscation and discusses limitations on these approaches. **CHAPTER 6: CONCLUSION** concludes the research done in field of code obfuscation and mentions the future work.

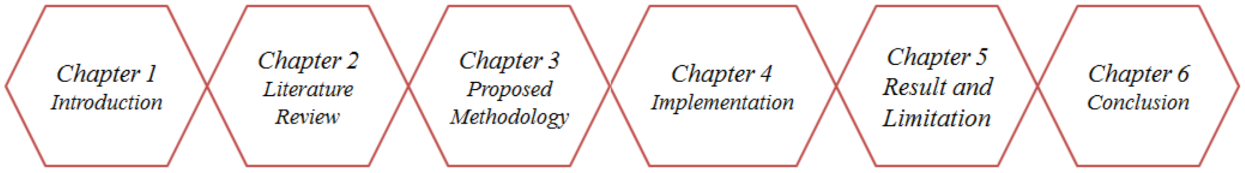


Figure1.7: Thesis Organization

CHAPTER 2

LITERATURE REVIEW

Chapter # 2: Literature Review

This chapter of thesis report will provide an overview of the current literature on the relevant area in detail which includes computer security comprises network security and software security. The main focus of this literature is on code obfuscation by protection the software applications. To attain the characterized strategy and techniques to carry out the research, the next step is to review the literature that will discuss below and shown in **Figure 2.1**:

- Relevant research of selected topic is done previously.
- Terminologies and frameworks presented and measured.
- Findings and consequences of the previous research.
- Gaps in previous research.

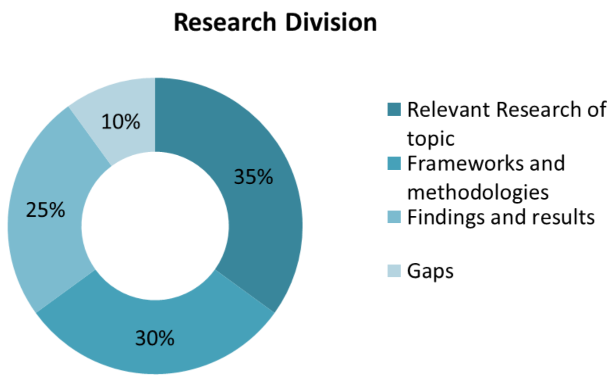


Figure2.1: Research Division

With the purpose of these highlighted points, we extensively revised the current literature by using different sources and their sequences shown in **Figure 2.2**:

- Research Engines i.e. IEEE, Elsevier, ACM, Springer.
- Academics and organizations websites.
- Git-hub.

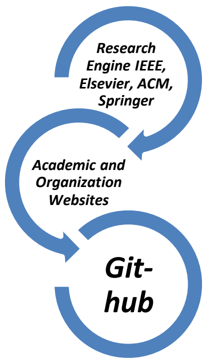

Research Engine IEEE, Elsevier, ACM, Springer
Academic and Organization Websites
Git-hub

Figure2.2: Research Sequence

Literature review is segmented into following parts. **Section 2.1** elaborates research methodology nominated for this specific research. **Section 2.2** contains prior study on selected topic and results of these studies. **Section 2.3** discusses the procedure of research conducted whereas **Section 2.4** will discuss relevant study.

2.1. Research Methodology:

Research methodology is measure of the study that distinguishes the procedure through which research will directed. It is the approach which is used to identify the road map of research, select the right methodology and analyze the information about selected topic. As we selected designed based research method to cover code obfuscation techniques for protection at software level, which allows us recursive method to choose hybrid techniques, perform critical analysis on selected techniques and algorithms after that assemble concentrated results to improve existing terminologies. Our main purpose is to improve educational practices through iterative experimentation that covers analysis, design, development and implementation with recursive criticism and collaboration of practitioners. Our basic approach is shown in **Figure 2.3** below:

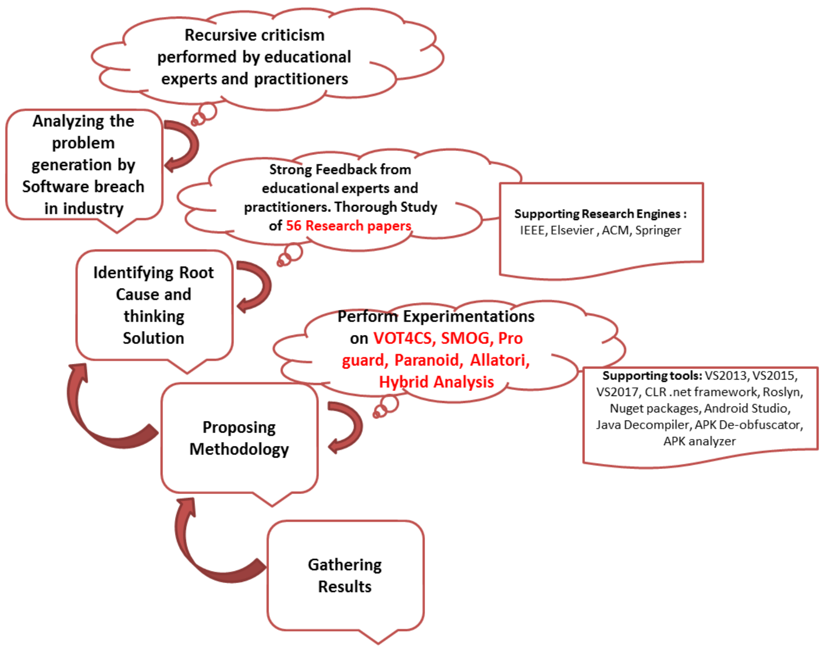

Recursive criticism performed by educational experts and practitioners
Analyzing the problem generation by Software breach in industry
Strong Feedback from educational experts and practitioners. Thorough Study of 56 Research papers
Supporting Research Engines : IEEE, Elsevier , ACM, Springer
Identifying Root Cause and thinking Solution
Perform Experimentations on VOT4CS, SMOG, Pro guard, Paranoid, Allatori, Hybrid Analysis
Supporting tools: VS2013, VS2015, VS2017, CLR .net framework, Roslyn, Nuget packages, Android Studio, Java Decompiler, APK De-obfuscator, APK analyzer
Proposing Methodology
Gathering Results

Figure2.3: Research Approach

2.2. Prior Study:

In the period of hasty alteration in software application and innovation, digitalization has now contacted essentially all aspects of our lives where it conveys substantial social assistances and financial benefits subsequently. In many organizations, IT-based systems are executed and exceedingly networked. In the interim, there are some enormous fundamental changes happened in industry where daily life is relying on digitalization. Progressively use of digital applications provides opportunities and many risks in the same time. So security of user is extremely crucial factor when the aim of attacker is not only to exploit vulnerabilities but also crack application algorithm by using techniques of reverse engineering and re-use it in many other ways **[24, 25, 26, 27]**. According to the definition of W. Stallings about computer Security given in NIST Computer Security Handbook *"Computer security is protection afforded to automated information system in order to attain the applicable objectives of preserving the integrity, availability and confidentiality of information system resources"* **[28]**. According to study of W. Stallings increasing trend in usage of internet and applications, the chance of attacks increases which leads to raise the requisite of execution for complex protocols and secure applications. Any application can be secure in three ways i.e. database security, software security and network security as shown in **Figure 2.4.**

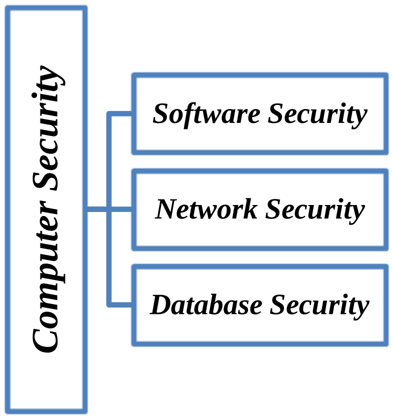


Figure2.4: COMPUTER SECURITY

Digital application has been increasing greater prominence and malevolent reverse engineering is a protected innovation infringement attack looked by android applications, which aims on comprehension and reusing the functionalities of a current application in the improvement of different application. Such attacks make it simpler for the contenders to separate the restrictive calculations and information structures from Java applications so as to fuse them into their own projects, to chop down their improvement time and cost **[1, 2, 12]**. De-compilation is the way toward producing source codes from transitional byte codes and the semi compiled nature of Java class files make it increasingly manageable to reverse engineering and re-engineering attacks by using de-compilation. Such instances of licensed innovation burglaries are hard to recognize and seek after legitimately. The American Society of Industrial Security (ASIS) has expressed that the force of protected innovation burglaries are raising and the majority of the organizations spend under 5% of their financial plans for security. ASIS has determined misfortunes because of robbery as 45 billion dollars in 1999 with the most recent figure as 150 billion dollars for every year. Software obfuscation is a well-known methodology where the program is changed into a obfuscated program utilizing an 'obfuscator' so that the usefulness and the info/yield conduct is protected in the obfuscated program though it is substantially more hard to figure out the obfuscated program **[13, 23]**. However, obscurity is a progressively practical technique for counteracting reverse engineering; there are 'deobfucators' accessible to crush a portion of the less modern obfuscation methodologies. The well-known change systems utilized for obfuscation are (i) lexical obfuscation which makes the structure of the changed program hard to fathom (ii) data flow obfuscation that clouds the significant data and their data structures. (iii) Code flow obfuscation that changes to darken the stream of execution **[16, 13]**. The obscurity can be performed on the source code, the middle of the road code or the machine executable code. According to Collberg, the effectiveness of obfuscation is measured into three four degrees; potency, resilience, cost and stealth. Hence potency alludes to the difficulty of the obfuscation to be comprehended by people, whereas resilience alludes the difficulty of the obfuscation to be circumvent naturally by a de-confusion procedure, cost refers to the computational overhead included by the obfuscation, stealth suggests to the perceptibility of the obfuscated code. In the light of literature and studied techniques we identified reverse engineering attacks are customary dispersed and effortlessly de-compliable formats like java byte code files.

An extensive research has been conducted to extract research from different scientific databases for computer security as discussed above. The research papers are categorized in **Table 2.1** that shows references aim to accomplish selected research and prioritize the topic on the basis of gaps and study conducted.

Classification on Security	**References**
Databases Security	[29],[30],[31],[32],[33],[34],[35]
Network Security	[36],[37],[38],[39],[40],[41],[42],[43],[4],[5],[6],[7],[8],[9]
Software Security	[1],[2],[3],[11],[12],[13],[17],[18],[20],[24],[25],[26]

Table2.1: CLASSIFICATION OF SECURITY

According to extensive research on computer security, a great deal of investigation is performed to ensure application protection. A total of 33 papers are used to categorize for identification of the prior study. In the light of literature review, the first barrier to resist the attacker is at network level (man-in-the-middle) and fraction of studies is high on this level. As per studies **[41]**, network is deliberately designed for the communication of two systems, geographically on same or different locations. Once when system is connected, they can freely transform information with any system. This sort of transformation is useful for individual as well as corporate. On the one side where usage network is giving huge benefits to users, on the other side it is the way to exchange most critical and confidential information where attacker can easily spasm on the communication channel and stole the required data and maltreatment the user's information in anyway. Attack on any system could be on data, application structure and algorithm or any outside virus that damages the internal network. In the challenging era it is obligatory to prevent from the attacks, so we need to implement security mechanisms. Subsequently first level of security is at network level (man-in-the-middle attacks) **[4]** where it repels entering any malware which is not identical or harmful. Some of the network level methods are Firewall **[5, 6, 42, 45, 46]**, IDs, Cryptography **[7, 8, 47]**, biometric **[38]**, encryption and DMZ layers **[49, 39, 40, 43]**. Defense-in-depth is not an artifact but security architecture that alerts network to be self-protective.

Database security is likewise imperative however it ruptures at extreme end and all organizations have prodigious reliance on database systems as crucial "data management technology" for handling large data ranging from daily task to critical operations. Hence extensive reliance on database systems offers opportunity to security breaches effecting crucial information. Where contemporary outsourced data management and cloud computing also raises bar for the need of database systems, hence security of data is more critical as ever. Traditional perimeter-oriented mechanisms for securing application are not sufficient for fine-grained security required for particular and sheltered information distribution among various clients and applications. Some of techniques to secure application in terms of data are access control mechanism, authentication, integrity controls and auditing **[29, 30, 31, 32, 33, 34, 35]**.

Even if computer system is protected by network and database end still it is not secure if application is executed on end-user mobile/ device. In our case, research is conducted to handle Man-at-the-end attacks **[10]** that's caters at software level. So principle center is around software protection on the grounds that in period of advanced application improvement for the most part applications are introduced and utilized on customer devices /mobiles. Man-At-The-End attacks are particularly common where attacker has physical access to the targeted area. These types of attacks are challenging to evaluate because attacker has limitless and authorized access to the targeted system where as all obstruction like network security are implemented on limited set of rules and they are not sufficient to address Man-at-the-end attacks because attack is not on organizational devices. A significant part of the surviving concentration to manage MATE assaults is technical. With the mean of achieving the software protection, mechanisms against Man-at-the-end are accredited as digital asset protection. Consequently network security is in adequate for such security. So to handle such attack many techniques are implemented at software level, but code obfuscation is one of the most adopted solutions and is aimed to hinder the understanding of code. Code understanding can't be totally obstructed. With spending a lot of time and cost on getting knowledge of obfuscated code attacker might not be able to extract exact algorithm and back track. Regardless of largely approved solution, its evaluation has been tended to in a roundabout way either by utilizing inside measurements or taking the perspective of code analysis, e.g., thinking about the related computational multifaceted nature. As per literature and practitioner practices, certain factors has been extracted listed in **Table2.2**

Characteristics	**Network Security**	**Database Security**	**Software Security**
Nature	Static	Dynamic	Dynamic
Development cost	High	Low	Low
Maintenance cost	High	Low	Low
Resistance	High	Low	Low
Learnability	Low	High	High
Deploy-ability	Hardware-dependent	Easy	Easy
Exploitable	High	Low	Low

Table 2.2: CHARACTERISTICS OF SECURITY

2.3. Procedure:

This research is conducted by reviewing total **60** research papers extracted from different search engines including IEEE, Elsevier, ACM, Springer. The selected papers was narrowed by focusing on concerned research specifically code obfuscation, its techniques and experimentations. Initially research is made on wider area of computer security by selecting **33** papers on the criteria of relevancy and methodology proposed; in the result of that research we identified three main security classifications as shown in **Figure 2.4.** Computer security is divided in network security scrutinized on 14 research papers, database security analyzed on 7 research papers and software security examined on 12 research papers as shown in **Figure 12**.

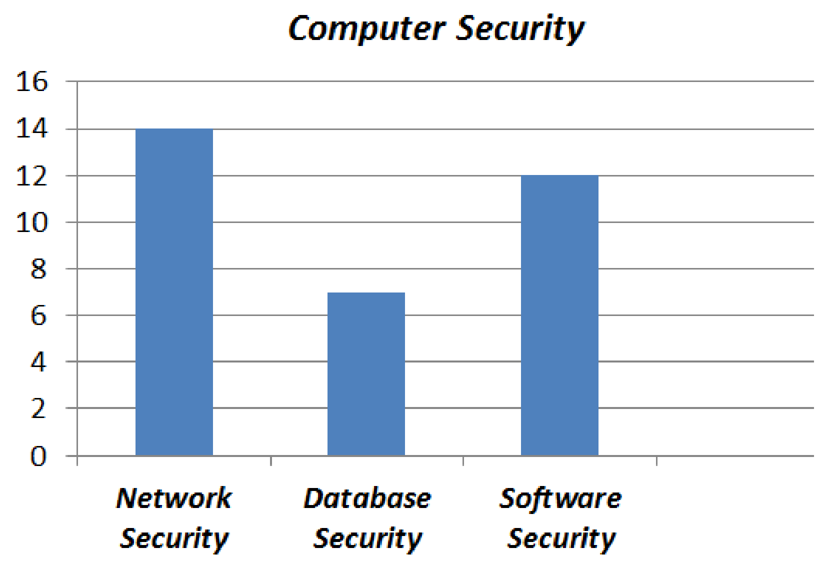


Figure2.5: Security Graph

2.3.1. Gap identified in classified terms:

After scrutinizing and reviewing selected research papers and collecting feedback from practitioners and students, it is examined that in the era of digital technology mobile applications are most commonly used as shown in **Table 2.3**

Audience	Usage over internet	Percentage
Students	Web Applications	35%
	Digital Apps	65%
Practitioners	Web Applications	45%
	Digital Apps	55%

Table2.3: AUDIENCE FEEDBACK

As per market requirement and paradigm shifted towards digital world, mobile applications are in more demands. After compiling the general feedback and prior research made in computer security, result is concluded that network security is already done vast to resist the attackers from external attacks and presented a lot of techniques, protocols and frameworks but they are not sufficient to secure mobile applications because it is always being installed on end-user device. So in this case, if mobile application is not protected inside by any mean, competitor can buy a copy and perform reverse engineering calculations; hence attacker damages the proprietary software.

2.3.2. The Selection Criteria:

The selection criteria are utilized to recognize the correct way to deal with concentrate look into papers. On the premise where featured papers were not meeting the selection criteria, at that point they were avoided from research. Selection criteria covers three points discussed as follow:

- The literature depended on a hunt in the descriptor for 'computer security'. Research has been conducted by using search engines i.e. IEEE, Elsevier, ACM, Springer to collect the literature on computer security. Afterwards feedback has been collected to identify the pain area of end-user. Then a result has been compiled on the basis of feedback and research results.
- Right off the bat we chose base paper to comprehend the idea of code obfuscation, which is "Manufacturing cheap, resilient, and stealthy opaque constructs" by Collberg (1998). On the basis on main concepts, the selected papers published from 2000 to 2018. The explanation behind choosing this timeframe is that, main this topic didn't find so vast research moreover numerous journals and conferences have distributed inquires about identified with context-aware computing since 2000.
- This paper covers journal, conference, survey and one book chapter. The reason behind selecting maximum journals and conferences is that both industrial experts and academicians use these much of the time to acquire information and spread their examination discoveries. Consequently, these demonstrate the most abnormal amount of research

Sr. no	Scientific database	Selected Research	No. of researches
1.	IEEE	[1],[3],[4],[5],[6],[9],[10],[14],[15],[16],[17],[18],[19],[24],[26],[27],[44],[45],[52],[56]	20
2.	ACM	[7],[12],[13],[21],[22],[46],[48],[49],[50],[54],[55]	11
3.	Springer	[11],[23],[29],[43]	4
4.	Elsevier	[20], [31],[32],[33],[34],[35],[36],[38],[39],[40],[41],[46]	12
5.	Others	[2],[8],[25],[28],[30],[37],[42],[47],[53]	9

Table2.4: DETAILS OF RESEARCH WITH SCIENTIFIC DATABASE

2.3.3. Data Sources to Extract Research Papers:

The papers were scrutinized according to process as shown in **Figure 2.6.** The data source of this research were optional source involves journal and conferences on software security from 2000 to 2018. The papers are selected as follow: initially base paper was identified to understand the concept of code obfuscation. Then we listed search engines that will be using authentic research, afterwards papers were searched on that selected scientific databases by using titles i.e. computer securities, network protection, hardware security, software protection, code obfuscation. Almost approx. 450 papers were identified. That was shrinking down on the basis of abstract / full title description. Next, the papers were cautiously checked on to choose those that considered the concerned topic as the core part, 60 remained on the grounds that the rest of the papers did not meet the second determination criteria.

2.3.4. Method of classification of computer security:

For the purpose of this research, the classification created here comprises of following three categories i.e. network security, software security and database security. As already discussed in **Figure 2.4** and **Table 2.1** each category is further divided into sub categories.

2.3.4.1. Classification Method:

The classification is based on literature review and feedback collected including the trending topics in computer applications. It is distinguished as per titles, publication years and language used for implementation.

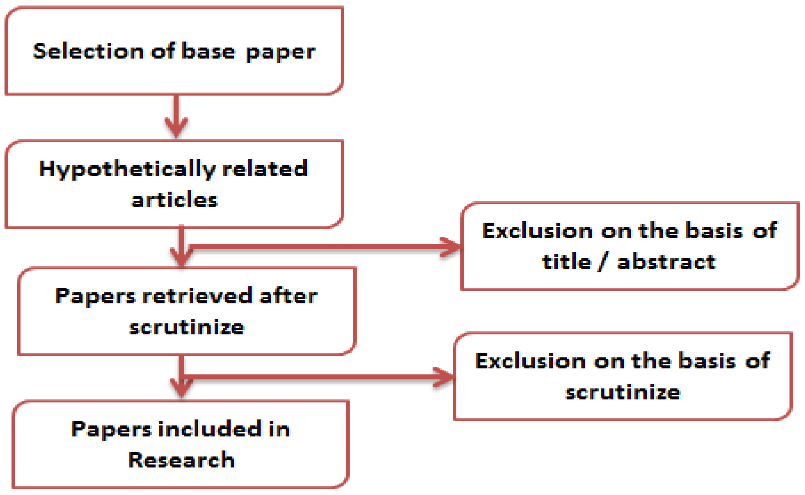

Selection of base paper
Hypothetically related articles
Exclusion on the basis of title / abstract
Papers retrieved after scrutinize
Exclusion on the basis of scrutinize
Papers included in Research

Figure2.6: Search Process

2.3.4.2. General Classification by title / Abstract:

Once base paper was selected, next step was to perform brief analysis to search for titles on scientific database. Where papers were identified and excluded if titles were not relevant. After performing this exercise, papers were excluded on the base of abstract review, where content analysis was performed

2.3.4.3. Classification by publication year:

As our research is design-based and according to topic demand, we selected papers from 2000 to 2018 because fundamental this theme didn't discover so tremendous research in addition various research have conveyed asks about related to setting mindful figuring since 2000 as shown in **Figure 2.7**.

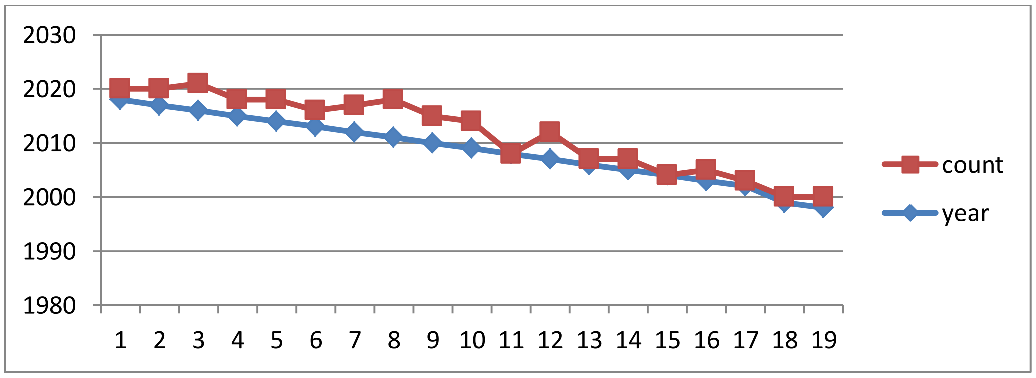


Figure2.7: Classification of Publication Year

2.3.4.4. Classification on the basis of feedback collected:

After reviewing literature, we conducted little survey to capture feedback from industries and students to identify the pain area of this era regarding trends in market. We selected 25 students and 20 practitioners to fill the survey. After collecting feedback from academics as well as industry we identify there is more need of mobile applications as shown in **Figure 2.8**. Where we identified when we are running in the era of digital technology, then only network security is not sufficient to protect applications.

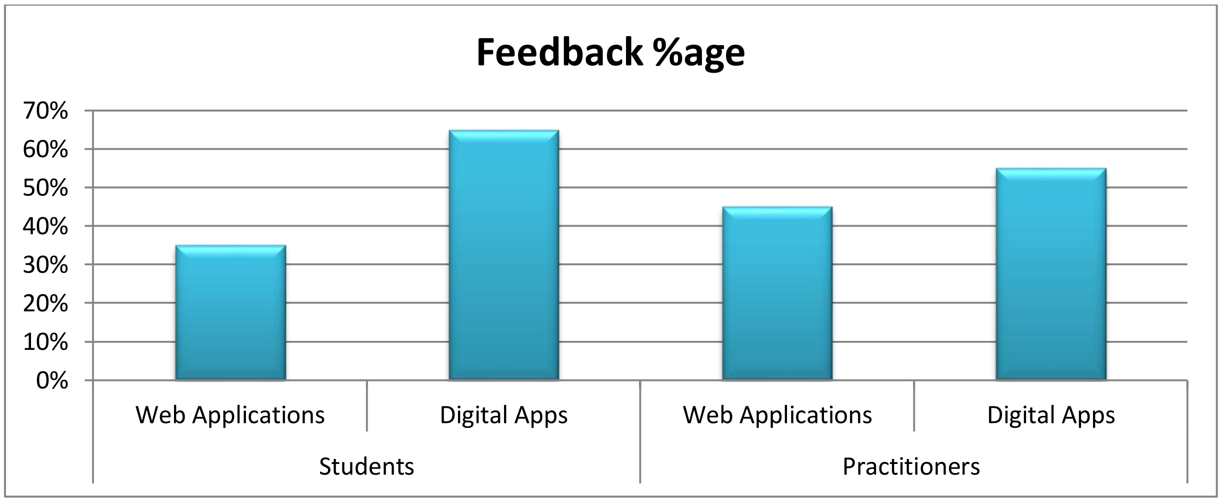


Figure2.8: Graph of Audience Feedback

2.3.4.5. Classification on the basis of attacks on software security:

After identifying the intensified problem statement, we tried to identify attacks on software security because end-user is executing the application on its own device. As already elaborated, there are many techniques to protect software from reverse engineering but we need to highlight attacks from which our mobile application is being attacked as shown in **Figure 2.9 and Table 2.5**.

Sr. no	Attacks	Percentage	Reference
1.	Buffer over flow	29%	[57]
2.	Code injection	18%	[58]
3.	JIT Spraying	11%	[59]
4.	Slicing	7%	[59]
5.	piracy	7%	[60]
6.	Reverse Engineering	28%	[23] Our Research Focus

Table2.5: CLASSIFICATION OF ATTACKS

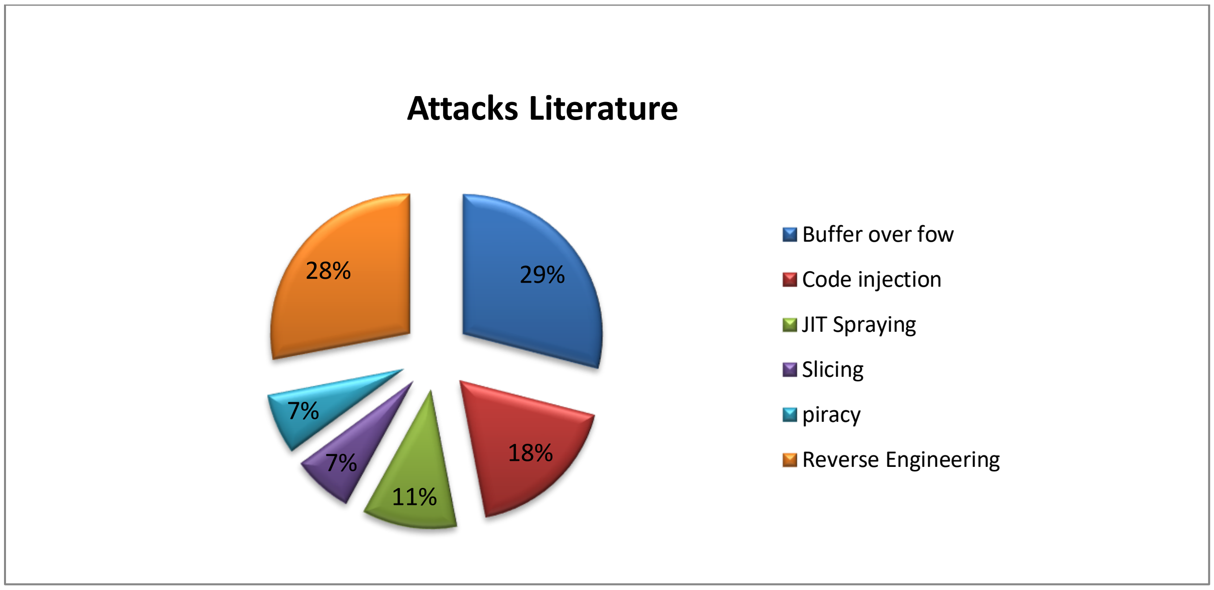
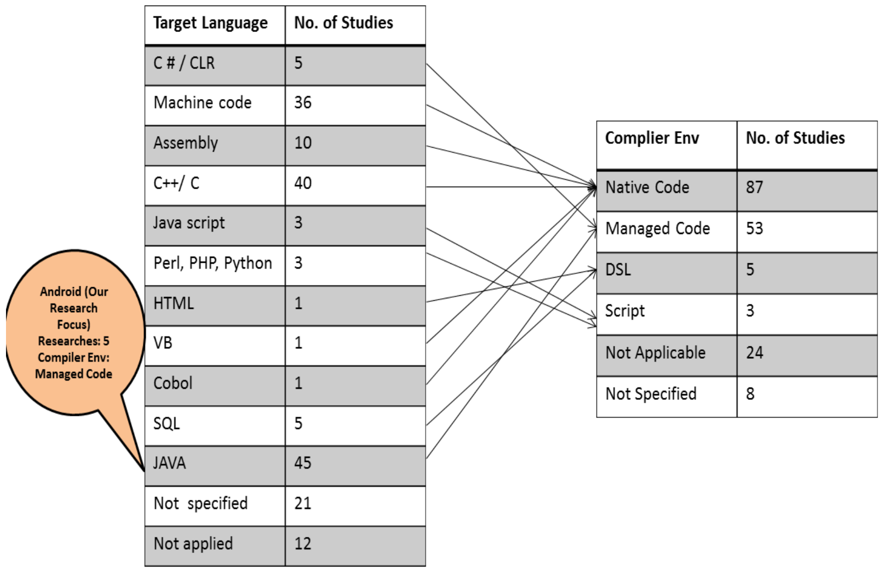


Figure2.9: Attack Literature

2.3.4.6. Classification on the basis of Environment:

After mitigating the attacks on software, we analyzed the language platform and intermediate compiler from different scientific databases to identify the gaps and flaws left in studies as per literature in software security. We collected a lot of literature from prior studies. By focusing on specific mobile applications and supporting platform i.e. android we found less literature in selected area. Detailed analysis is shown in **Figure 2.10**



Target Language	No. of Studies
C # / CLR	5
Machine code	36
Assembly	10
C++/ C	40
Java script	3
Perl, PHP, Python	3
HTML	1
VB	1
Cobol	1
SQL	5
JAVA	45
Not specified	21
Not applied	12

Complier Env	No. of Studies
Native Code	87
Managed Code	53
DSL	5
Script	3
Not Applicable	24
Not Specified	8

Android (Our Research Focus) Researches: 5 Compiler Env: Managed Code

Figure2.10: Classification of environment

CHAPTER 3

PROPOSED METHODLOGY

Chapter # 3: Proposed Methodology

By combining all efforts done for this planned research, this chapter is designed to elaborate proposed methodology. **Section 3.1** will discuss the tools and frameworks used for code obfuscation, whereas **Section 3.2** will cover proposed method of this research

3.1. Designated Obfuscators:

As part of this research, Pro-guard, Allatori and Paranoid is being compiled in single library and provided all obfuscation facilities in single open source package.

- Class, package, method name obfuscation
- Debug info obfuscation
- Incremental obfuscation
- Optimization of code
- Shrinking of unused resources
- String Obfuscation
- Flow Obfuscation
- Prevention of java files after de-obfuscation
- Dead code insertion

3.1.1. ProGuard:

ProGuard also offers defense to counter reverse engineering by obfuscating the names of classes, fields and methods. It is a default tool in development IDE's including Android Studio to obfuscate, shrink, optimize, and preverify your code. Classes, methods and variables can also be better obfuscated by specifying a text file in ProGuard rules file which contains list of custom entries. Optimization can be made more advanced by changing code filters and optimization passes given in ProGuard rules file. ProGuard does not obfuscate resources and strings.

Availability: Open-source

3.1.2. Paranoid:

Paranoid is an open-source third party string obfuscator library specifically for Android Applications. Every string literal and string constant defined in annotated classes or activities will be obfuscated. Paranoid initializes an array of unique characters from all obfuscated strings and an array of indexes in the character array per each obfuscated string. When a project is compiled all hardcoded strings are swapped with function calls with a parameter which returns that specific string. The outcome is when the code is decompiled it is not possible to simply search for strings.

Availability: Open-source

3.1.3. Allatori:

Allatori Obfuscator has all the features we will possibly require for protecting our mobile application, and is frequently enhanced and improved to meet the challenge of functioning as a cutting edge Java obfuscator system. Following features make Allatori one of the best obfuscators for Android after ProGuard:

- Name Obfuscation
- Incremental and Flow Obfuscation
- Debug Info Obfuscation
- String Encryption
- Optimizing
- Watermarking

Availability: Its trail version is available with limited features.

3.1.4. DashO:

DashO is a Java and Android Obfuscator. It provides large-scale app hardening and aggressive defense, greatly decreasing the danger of logical property theft, data theft, piracy, and tampering. Its layered obfuscation, encryption, watermarking, auto-expiry, anti-debug, anti-tampering, anti-hooking, anti-rooted device solution offers guard for applications all around the globe.

Availability: Trial Version Unpaid

3.1.5. Arxan:

Arxan provides mobile application code protection for apps written in Java and Kotlin to protect against successful decompiling and tampering. Arxan can provide defense for keys and delicate data via White-Box Cryptography for all Android-based devices. Arxan mobile application protection also comprises of risk analytics to aid customers recognize whether apps are functioning securely vs. running in a hazardous and unsafe environment.

Availability: Paid version

3.1.6. Cloakware:

Cloakware Software Protection is a collection of innovative cyber security technologies, libraries and tools that enable the users to customize the protection of their critical digital assets such as keys, code and data. It works on an extensive range of platforms including PC's and mobile devices whether Android and iOS.

Availability: Not available anymore

3.2. Proposed Solution:

With the support of extensive prior study and practitioner's feedback, we proposed hybrid open source code obfuscation technique. We created one package by using three libraries pro-guard, paranoid and allatori by enhancing and providing features in one library. By using such algorithms we defined customized set of rules. And by using our base paper, we used Collberg's points of efficiency in regards to check the level of code obfuscation. We also performed code optimization and code shrinking to reduce APK size and efficiency on **HIGH.**

Paranoid initializes an array of unique characters from all obfuscated strings and an array of indexes in the character array per each obfuscated string. When a project is compiled all hardcoded strings are swapped with function calls with a parameter which returns that specific string. **ProGuard** uses a different mechanism for hardening code to achieve name obfuscation. It utilized dictionaries to designate what to rename a class, method and package. ProGuard has a default dictionary which contains the letters a-z. It also provides flexibility to use a customized dictionary with user-defined entries. Control flow obfuscation is accomplished by inserting irrelevant and dead code, reordering statements and flattening logical structure of code. Initially, we split up the body of the method to simple blocks, and then we combine all these blocks, which were formerly at different nesting levels, adjacent to each other. **Allatori** gives one name to as many components as possible for name obfuscation. As a result, there is a high probability that one name will be reused for naming the variables, classes and methods. Allatori changes the typical java loops i.e. conditional and branching statements and where it's possible, the series of instructions are transformed so that after decompilation the original java code is impossible to find. After analyzing a class, Allatori finds all the hard-coded strings and encrypt them and are kept in constant pool of the affected class. The combination of distinct methods used in Allatori secures the application to the maximum, often resulting into failure of the decompilation process.

3.3. Set of Rules:

3.3.1. Obfuscation:

1. ’minifyEnabled’ set to true in Gradle settings file
2. Using ‘useuniqueclassmembernames’ property to give the identical obfuscated names to class members that have the same original names, and different obfuscated names to class members that have different names
3. Referring to use customized text file with random entries for better obfuscation of variables, methods, classes and packages names i.e. obfuscationdictionary, classobfuscationdictionary, packageobfuscationdictionary with parameter of custom file being used.
4. Repackaging all renamed packages in one package by name as entered after flattenpackagehierarchy property
5. Repackaging all renamed classes in one package by name as entered after repackageclasses property

3.3.2. Shrinking:

1. ‘shrinkResources’ and ’minifyEnabled’ set to true in Gradle settings file

3.3.3. Optimization:

1. ’minifyEnabled’ set to true in Gradle settings file
2. Using a different default ProGuard configuration file by modifiying the ‘proguard-android.txt’ value to ‘proguard-android-optimize.txt’ in Gradle settings
3. Adding filter ‘optimizationpasses’ with numeric value which indicates number of optimization passes to be executed. Default number of passes is only one. We have set 5 for best optimization.

4. Adding different parameters to ‘optimizations’ for achieving enhanced optimization like code/removal, code/simplification.

3.3.4. Obfuscated Chunks:

There are many obfuscated methods i.e. lexical, control flow, data flow obfuscation. These levels of obfuscation are being attained by performing many experiments.

Techniques	Results
Lexical Obfuscation	- Achieved from pro-guard for class, package, method, statements, variables obfuscation - Achieved by paranoid for String obfuscation
Control flow Obfuscation	- Achieved by using Allatori
Data flow Obfuscation	- Achieved by pro-guard

Table3.1: OBFUSCATED CHUNKS

3.4. Collberg’s Taxonomy on hybrid approach:

Collberg discussed efficient analysis of code obfuscation by giving four taxonomies i.e. Resilience, Cost, Potency, Stealth. Because of the regular code impression of the interpreter, virtualization obfuscation can be effortlessly distinguished and in this manner its stealth is generally low contrasted with most obfuscation methods. Accordingly, we don't trust this measure requires further examination for the specific execution displayed in this research work.

Rest of the three taxonomies and their analysis is below:

3.5. Analysis Metrics:

Although obfuscation implementations can make de-obfuscation attacks difficult and more complex, it is unavoidable to make any code totally unreadable and attack free. Identical obfuscation implementation was performed for all test sample projects but the result varied due to different code flows, structures and practices applied.

To assess the quality of any obfuscator, the overall performance will be denoted as $\boldsymbol{S}_{quality}$. $\boldsymbol{S}_{quality}$ consists of three major metrics i.e. Potency as $\boldsymbol{S}_{pot}$, Resilience as $\boldsymbol{S}_{res}$, and Cost as $\boldsymbol{S}_{cost}$. Following are the metrics with the help of which we can assess the effectiveness of our obfuscation techniques applied on mobile application code:

3.5.1. Potency

Potency is the extent to which an attacker can read, understand and interpret any obfuscated code. Since measuring potency of any obfuscated program is difficult, the potency can be drilled down to make the analysis easier into following four complexity parts: variable potency, nesting potency, control flow potency, and code length. Nesting complexity and code length have been given more weightage than the others by experts. Low weightage factors result in modest increased time for cracking the code. A medium weight adds more increased time to decompile the code. A high weight factor not only adds overhead and decompiling time but also transforms the methodology of code structure. Potency equation is as follows:

$$S_{pot} = 0.4\,(S_{nesting}) + 0.2\,(S_{variable}) + 0.2\,(S_{control}) + 0.4\,(S_{length})$$

EQUATION3.1: POTENCY MEASURE

Potency has positive impact on the quality of any obfuscated program. Potency is measured on the scale from 0 to 1, where 1 means extremely potent and 0 means no potency in obfuscated code. Based on our extensive survey conducted from experts of Information Security field to assess the level of potency our obfuscation achieves for all tested sample projects, following was the result concluded:

Sr. #	Test	S_{pot} before obfuscation	S_{pot} after obfuscation
1	Project 1	0	0.75
2	Project 2	0	0.68
3	Project 3	0	0.65

Table 3.2: SAMPLE TEST PROJECTS FOR POTENCY

3.5.2. Resilience

Resilience is the extent to which automated attacks can manipulate and reverse engineer any obfuscated code with the intention of extracting and exploiting useful information. These attacks can be termed as efforts to convert the code back to its original source code form. Resilience also has positive impact on the quality of any obfuscated program. No specific formula has been defined to evaluate Resilience, so we give a float value from 0 to 1.

Sr. #	Test	S_{res} before obfuscation	S_{res} after obfuscation
1	Project 1	0	0.6
2	Project 2	0	0.4
3	Project 3	0	0.5

Table 3.3: SAMPLE TEST PROJECTS FOR RESILIENCE

3.5.3. Cost

The cost measures delay and additional resources utilized by an obfuscated program during runtime. Cost is split into three types: memory, storage and runtime. Memory is the heap and non-heap memory consumption for classes, arrays, methods and variables used by an obfuscation program compared to that of original program. Storage implies the file size of obfuscated code whereas runtime is the amount of time an obfuscated program takes to run and also to generate APK file. Cost has a negative impact on the quality of obfuscated code. Experts believed that memory and runtime are far more crucial than size of an obfuscated program. Below is the cost equation:

$$S_{cost} = 0.4\,(S_{memory}) + 0.45\,(S_{runtime}) + 0.15\,(S_{storage})$$

EQUATION3.2: COST MEASURE

Negative cost shows a positive impact on overall quality of obfuscation applied is shown in **Table 3.4**.

Sr. #	Test	S_{cost} before obfuscation	S_{cost} after obfuscation
1	Project 1	0	- 0.08
2	Project 2	0	- 0.05
3	Project 3	0	- 0.01

Table3.4: SAMPLE TEST PROJECTS FOR COST

$$S_{quality} = 0.4\,(S_{pot}) + 0.6\,(S_{res}) - S_{cost}$$

EQUATION3.3: COLLBERG'S TAXONOMY

Both potency and resilience have positive impact on quality while cost has negative impact on quality of an obfuscator. Resilience has been given more weightage than potency by experts as cognitive aptitude of computer programs is far low as compared to that of humans. If above statement gives zero result value, it means that no obfuscation has been implemented in code and is entirely readable and understandable. Therefore, greater the result value is, better will be the overall quality of an obfuscator as shown in **Table 3.5**.

Sr. #	Test	$S_{quality}$ before obfuscation	$S_{quality}$ after obfuscation	Overall $S_{quality}$
1	Project 1	0	0.4 (0.75) + 0.6 (0.6) – (–0.08) = 0.3 + 0.36 + 0.08	0.74
2	Project 2	0	0.4 (0.68) + 0.6 (0.4) – (–0.05) = 0.27 + 0.24 + 0.05	0.56
3	Project 3	0	0.4 (0.65) + 0.6 (0.5) – (–0.01) = 0.26 + 0.3 + 0.01	0.57

Table3.5: Sample Test Projects with Overall Quality Score Before And After Obfuscation

CHAPTER 4

IMPLEMENTATION

Chapter # 4 Implementation

4.1. Pro-Guard

Pro-Guard is one of the most popular and trustworthy java obfuscators. It makes Java and Android applications up to 90% smaller and up to 20% faster. Pro-Guard also offers defense to counter reverse engineering by obfuscating the names of classes, fields and methods. Pro-Guard can be used free of cost to secure mobile applications. It is a default tool in development IDE's including Android Studio to obfuscate, shrink, optimize, and pre-verify your code. Below Figure 4.1 shows Pro-Guard techniques steps.

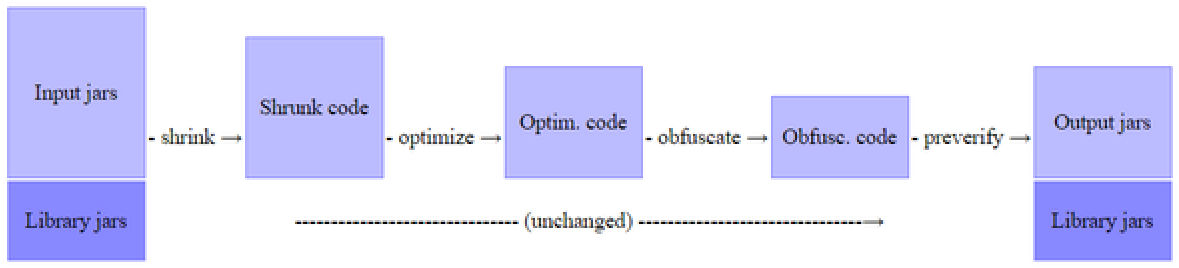
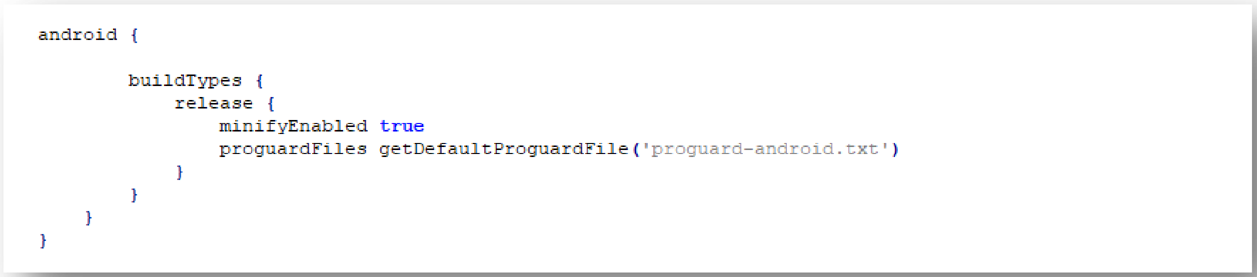


Figure4.1: Pro-Guard Techniques Steps

Android Studio provides a file that contains all of the essential configuration settings to run Android Pro-Guard obfuscation with application. To enable Pro-Guard work with default settings in Android Studio, following steps were taken:

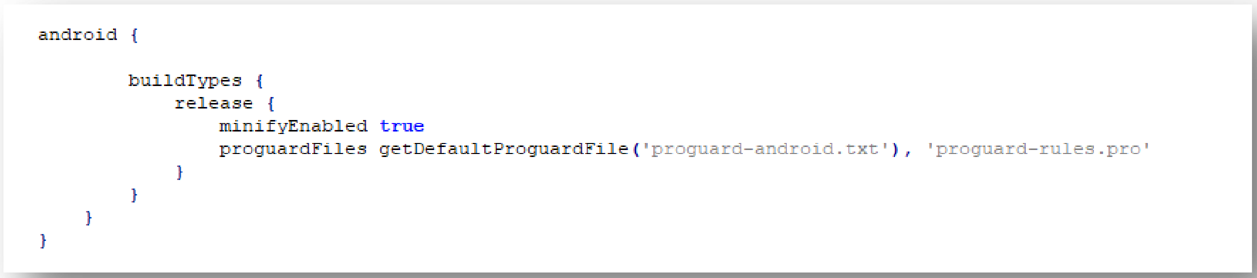
1. Set Minify enabled as true and default Pro-Guard file settings in App level gradle file as shown in below Figure 4.2.



```
android {

        buildTypes {
            release {
                minifyEnabled true
                proguardFiles getDefaultProguardFile('proguard-android.txt')
            }
        }
    }
}
```

Figure4.2: Enabling Default Pro-Guard Settings

We have used our own customized ProGuard file for better obfuscation than what default file provides us. For that, following change has to be done in above code as illustrated in Figure4.3.



```
android {

        buildTypes {
            release {
                minifyEnabled true
                proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
            }
        }
    }
}
```

Figure4.3: Enabling Customized Pro-Guard Settings

4.1.1. Shrinking Code:

There are many noticeable advantages of making app size smaller, such as improving user retention and contentment, faster downloads and no. of installations, and reaching users on lower-end devices, particularly in evolving markets. Many developers are more concerned about the obfuscation part of ProGuard, but the major edge it offers is the elimination of all unused code and resources that are otherwise dispatched with APK file. Below image shows the contribution of contents in making up the whole app size. From the looks of this Figure, resources (res) and code (.dex) contribute massively to the APK size.

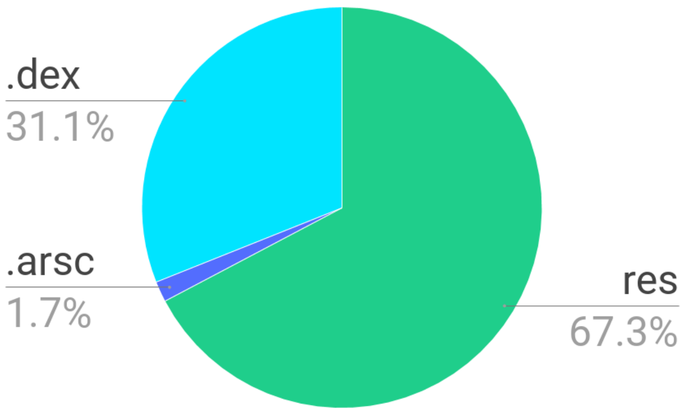
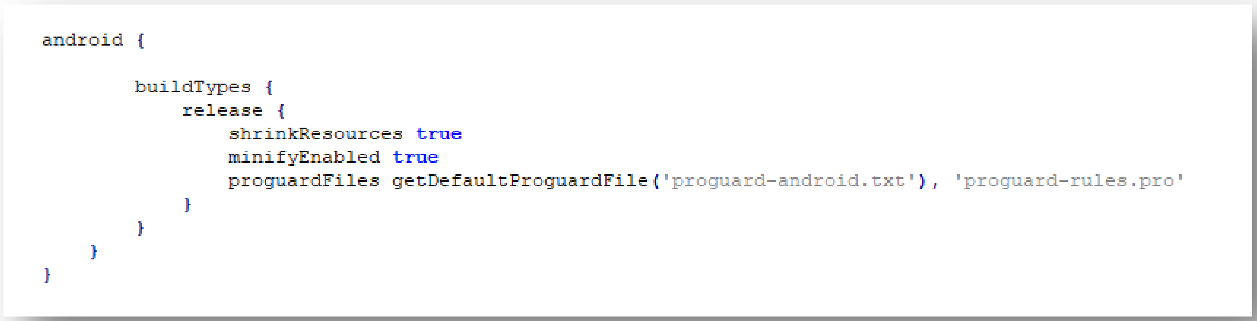


Figure 4.4: APK Size Ratio

Setting ‘minifyEnabled’ true shrinks code by removing unused chunks of code and references including code libraries. Whereas, setting ‘shrinkResources’ true on the other hand removes unused resource files from resources folder that are not references in the code. For ‘shrinkResources’ to work effectively, ‘minifyEnabled’ must be set to true as well as indicated in below Figure. Our APK size was decreased from 7.13 MB to 4.34 MB after setting both properties as true.

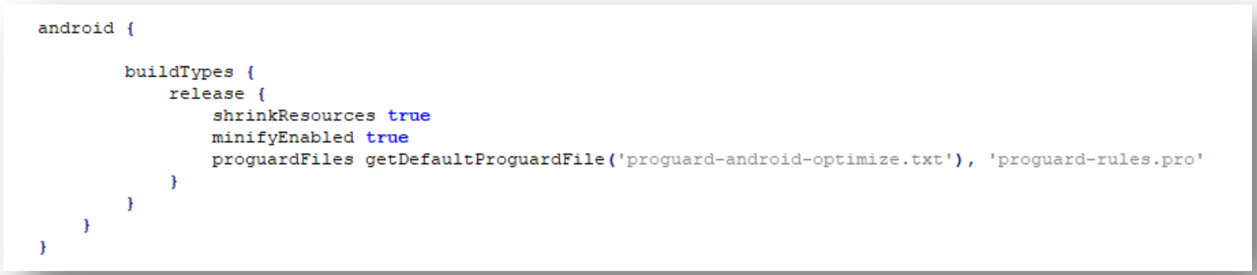


```
android {

        buildTypes {
            release {
                shrinkResources true
                minifyEnabled true
                proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
            }
        }
    }
}
```

Figure 4.5: Shrink APK Size

4.1.2. Optimizing Code:

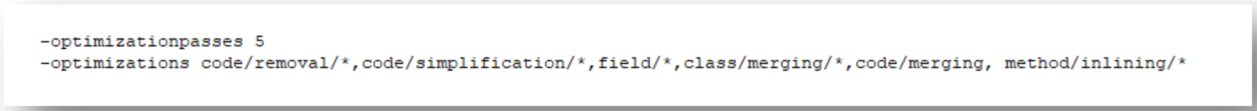
To unlock code optimization, we used a different default Pro-Guard configuration file. We modified the ‘proguard-android.txt’ value to ‘proguard-android-optimize.txt’ in build.gradle file as given in below Figure 4.6.



```
android {

        buildTypes {
            release {
                shrinkResources true
                minifyEnabled true
                proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
            }
        }
    }
}
```

Figure4.6: Optimization Settings

This will possibly enable app to run faster and decrease code size due to optimizations such as method in lining, class merging and code simplification and removal. For more advanced optimizations, we added certain filters in Pro-Guard rules file which are given in below figure:



```
-optimizationpasses 5
-optimizations code/removal/*,code/simplification/*,field/*,class/merging/*,code/merging, method/inlining/*
```

Figure4.7: APK Size Ratio

First filter 'optimizationpasses' indicate number of optimization passes to be executed. Default number of passes is only one. Numerous passes on code result in improved optimizations definitely. When maximum optimization is done, no more passes are performed. In **second filter**, we have included several parameters for enhanced optimization.

code/removal - It removes unused variables, dead code based on control flow analysis and data flow analysis and also removes exceptions with empty try blocks.

code/simplification – It performs peephole optimizations for variable loading and storing, arithmetic instructions, casting operations, field loading and storing, branch instructions, constant strings. It also simplifies code based on control flow analysis and data flow analysis.

class/merging – It merges classes vertically and horizontally in the class hierarchy

code/merging – It merges identical blocks of code by modifying branch targets

method/in-lining – It does inline short methods, inline methods that are only called once, and simplifies tail recursion calls wherever feasible.

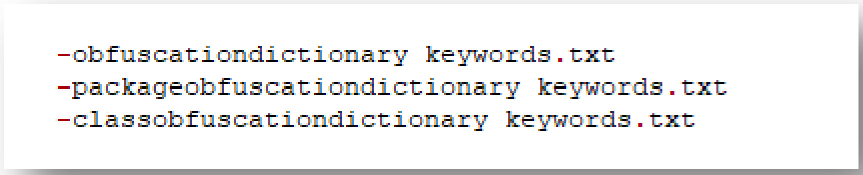
field – It removes write-only fields, marks fields as private and propagates the values of fields across methods.

4.1.3. Obfuscating Code:

4.1.3.1. Obfuscation of variables, methods, classes, packages:

Variable and method names are obfuscated with default set of characters and strings like a, b, xy etc. But they can also be better obfuscated by specifying a text file in Pro-Guard rules file which contains list of custom entries. Name obfuscation is hardly improved by doing this therefore, to overcome it, it is recommended to provide entries which can be confusing like using java keywords i.e. class, string, public, if etc. Entries from the list are chosen entirely random. Punctuation characters, empty lines, white spaces and comments after hash sign are totally ignored.

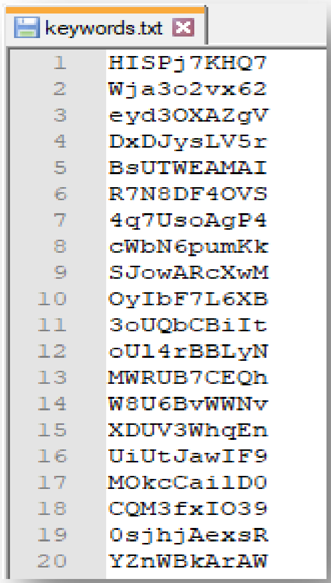
Above technique is only limited to renaming of methods and fields. To obfuscate class names, we have to specify text file separately in Pro-Guard rules file. Similarly, to obfuscate package names, we must provide text file explicitly in Pro-Guard rules file.



```
-obfuscationdictionary keywords.txt
-packageobfuscationdictionary keywords.txt
-classobfuscationdictionary keywords.txt
```

Figure 4.8: Properties for Obfuscation Dictionaries

Here, keywords.txt is our file with customized entries which is being used for obfuscation of variables, classes and packages. Different files can also be used and assigned to variable, classes and packages but we have used same file for this purpose. Below figure only shows a small part of keywords text file due to space limitation.

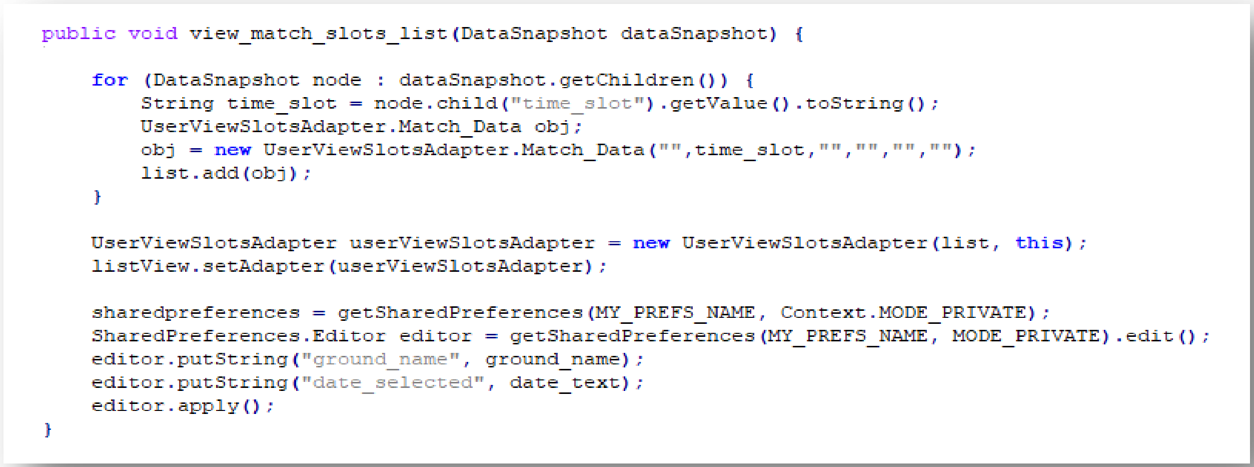

keywords.txt

```
HISPj7KHQ7
Wja3o2vx62
eyd3OXAZgV
DxDJysLV5r
BsUTWEAMAI
R7N8DF4OVS
4q7UsoAgP4
cWbN6pumKk
SJowARcXwM
OyIbF7L6XB
3oUQbCBiIt
oUl4rBBLyN
MWRUB7CEQh
W8U6BvWWNv
XDUV3WhqEn
UiUtJawIF9
MOkcCailD0
CQM3fxIO39
OsjhjAexsR
YZnWBkArAW
```

Figure 4.9: Custom entries to replace variable and class names Dictionaries

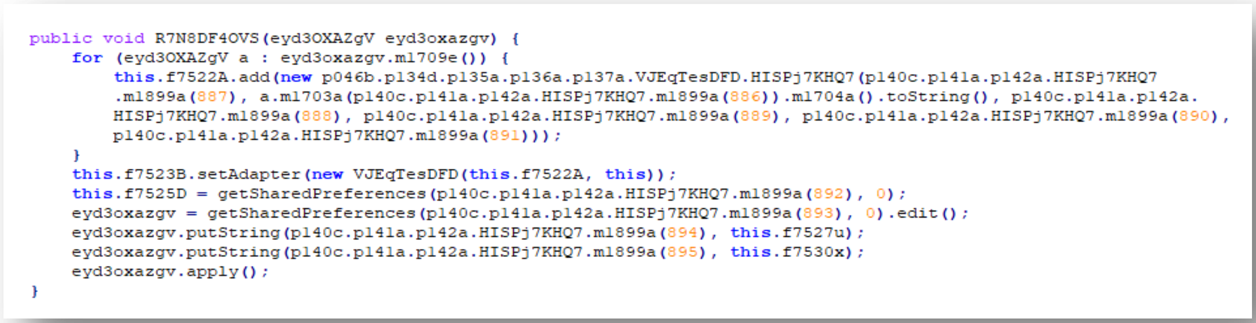
4.1.3.2. Impact of obfuscation dictionaries:

The impact of using class, variables and methods obfuscation dictionary from custom file is illustrated in below figures which show the original code and code after applying obfuscation.



```
public void view_match_slots_list(DataSnapshot dataSnapshot) {

    for (DataSnapshot node : dataSnapshot.getChildren()) {
        String time_slot = node.child("time_slot").getValue().toString();
        UserViewSlotsAdapter.Match_Data obj;
        obj = new UserViewSlotsAdapter.Match_Data("",time_slot,"","","","");
        list.add(obj);
    }

    UserViewSlotsAdapter userViewSlotsAdapter = new UserViewSlotsAdapter(list, this);
    listView.setAdapter(userViewSlotsAdapter);

    sharedpreferences = getSharedPreferences(MY_PREFS_NAME, Context.MODE_PRIVATE);
    SharedPreferences.Editor editor = getSharedPreferences(MY_PREFS_NAME, MODE_PRIVATE).edit();
    editor.putString("ground_name", ground_name);
    editor.putString("date_selected", date_text);
    editor.apply();
}
```

Figure 4.10: Original Code



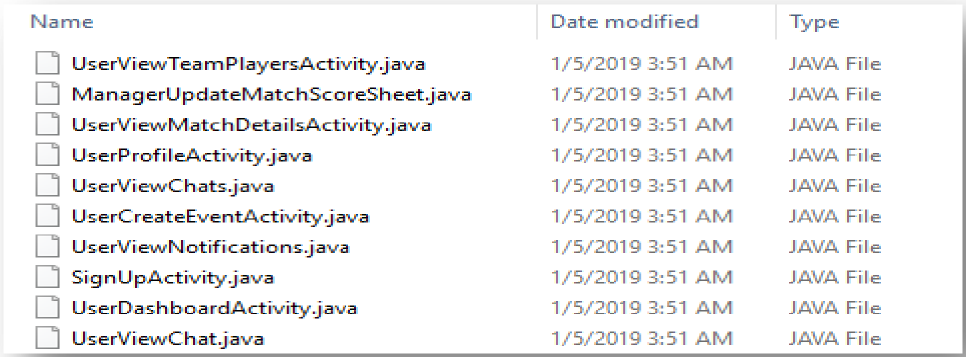
```
public void R7N8DF4OVS(eyd3OXAZgV eyd3oxazgv) {
    for (eyd3OXAZgV a : eyd3oxazgv.m1709e()) {
        this.f7522A.add(new p046b.p134d.p135a.p136a.p137a.VJEqTesDFD.HISPj7KHQ7(p140c.p141a.p142a.HISPj7KHQ7
        .m1899a(887), a.m1703a(p140c.p141a.p142a.HISPj7KHQ7.m1899a(886)).m1704a().toString(), p140c.p141a.p142a.
        HISPj7KHQ7.m1899a(888), p140c.p141a.p142a.HISPj7KHQ7.m1899a(889), p140c.p141a.p142a.HISPj7KHQ7.m1899a(890),
        p140c.p141a.p142a.HISPj7KHQ7.m1899a(891)));
    }
    this.f7523B.setAdapter(new VJEqTesDFD(this.f7522A, this));
    this.f7525D = getSharedPreferences(p140c.p141a.p142a.HISPj7KHQ7.m1899a(892), 0);
    eyd3oxazgv = getSharedPreferences(p140c.p141a.p142a.HISPj7KHQ7.m1899a(893), 0).edit();
    eyd3oxazgv.putString(p140c.p141a.p142a.HISPj7KHQ7.m1899a(894), this.f7527u);
    eyd3oxazgv.putString(p140c.p141a.p142a.HISPj7KHQ7.m1899a(895), this.f7530x);
    eyd3oxazgv.apply();
}
```

Figure 4.11: Obfuscated Code

-- Change starts

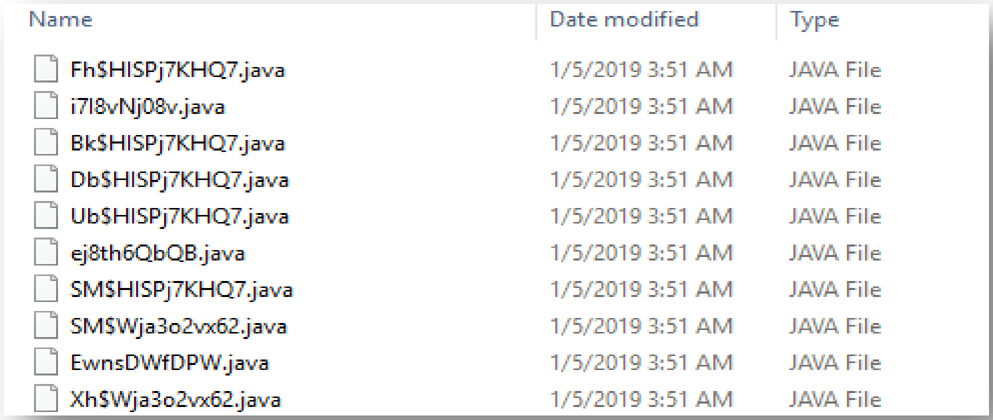
Above obfuscation is the outcome of using randomized dictionary to make it difficult to decompile classes and extract useful information. ProGuard uses all valid identifiers in the file. It ignores empty spaces and lines starting with '#'.

-- Change ends



Name	Date modified	Type
UserViewTeamPlayersActivity.java	1/5/2019 3:51 AM	JAVA File
ManagerUpdateMatchScoreSheet.java	1/5/2019 3:51 AM	JAVA File
UserViewMatchDetailsActivity.java	1/5/2019 3:51 AM	JAVA File
UserProfileActivity.java	1/5/2019 3:51 AM	JAVA File
UserViewChats.java	1/5/2019 3:51 AM	JAVA File
UserCreateEventActivity.java	1/5/2019 3:51 AM	JAVA File
UserViewNotifications.java	1/5/2019 3:51 AM	JAVA File
SignUpActivity.java	1/5/2019 3:51 AM	JAVA File
UserDashboardActivity.java	1/5/2019 3:51 AM	JAVA File
UserViewChat.java	1/5/2019 3:51 AM	JAVA File

Figure 4.12: Original Code



Name	Date modified	Type
Fh$HISPj7KHQ7.java	1/5/2019 3:51 AM	JAVA File
i7l8vNj08v.java	1/5/2019 3:51 AM	JAVA File
Bk$HISPj7KHQ7.java	1/5/2019 3:51 AM	JAVA File
Db$HISPj7KHQ7.java	1/5/2019 3:51 AM	JAVA File
Ub$HISPj7KHQ7.java	1/5/2019 3:51 AM	JAVA File
ej8th6QbQB.java	1/5/2019 3:51 AM	JAVA File
SM$HISPj7KHQ7.java	1/5/2019 3:51 AM	JAVA File
SM$Wja3o2vx62.java	1/5/2019 3:51 AM	JAVA File
EwnsDWfDPW.java	1/5/2019 3:51 AM	JAVA File
Xh$Wja3o2vx62.java	1/5/2019 3:51 AM	JAVA File

Figure4.13: Obfuscated Code

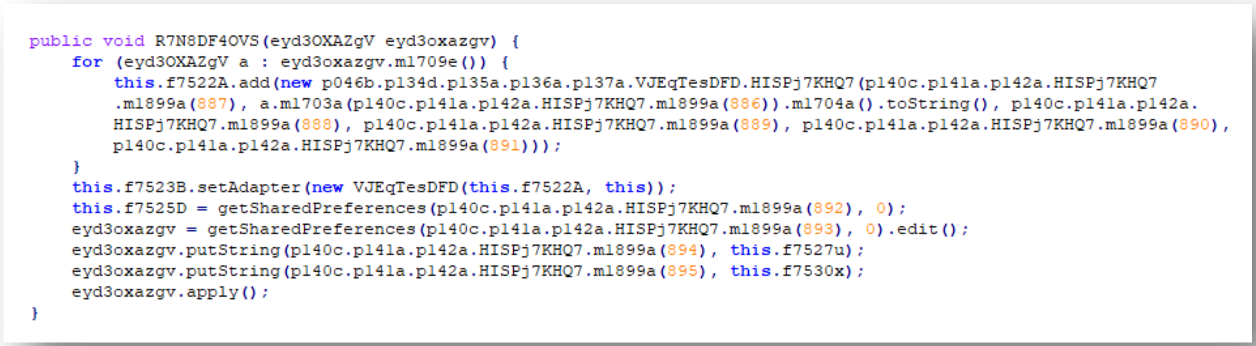
-- Changes start

Above renaming obfuscation of java classes is the result of using randomized dictionary so that every build will have a unique mapping, making it difficult for an attacker to understand and manipulate the code.

-- Changes end

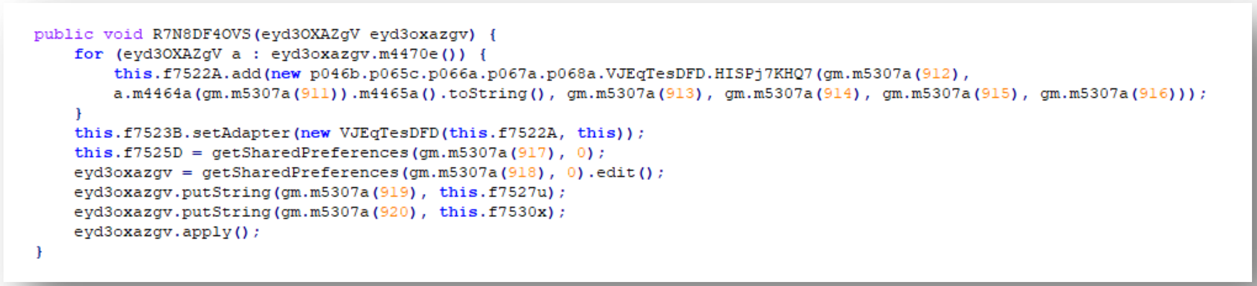
4.1.3.3. Optimizations:

Optimization passes improves code removal and simplification wherever necessary. This directly reduces our application size and performance. Below figures show the difference between codes without and with applying optimizations.



```java
public void R7N8DF4OVS(eyd3OXAZgV eyd3oxazgv) {
    for (eyd3OXAZgV a : eyd3oxazgv.m1709e()) {
        this.f7522A.add(new p046b.p134d.p135a.p136a.p137a.VJEqTesDFD.HISPj7KHQ7(p140c.p141a.p142a.HISPj7KHQ7
        .m1899a(887), a.m1703a(p140c.p141a.p142a.HISPj7KHQ7.m1899a(886)).m1704a().toString(), p140c.p141a.p142a.
        HISPj7KHQ7.m1899a(888), p140c.p141a.p142a.HISPj7KHQ7.m1899a(889), p140c.p141a.p142a.HISPj7KHQ7.m1899a(890),
        p140c.p141a.p142a.HISPj7KHQ7.m1899a(891)));
    }
    this.f7523B.setAdapter(new VJEqTesDFD(this.f7522A, this));
    this.f7525D = getSharedPreferences(p140c.p141a.p142a.HISPj7KHQ7.m1899a(892), 0);
    eyd3oxazgv = getSharedPreferences(p140c.p141a.p142a.HISPj7KHQ7.m1899a(893), 0).edit();
    eyd3oxazgv.putString(p140c.p141a.p142a.HISPj7KHQ7.m1899a(894), this.f7527u);
    eyd3oxazgv.putString(p140c.p141a.p142a.HISPj7KHQ7.m1899a(895), this.f7530x);
    eyd3oxazgv.apply();
}
```

Figure4.14: Without Optimization



```
public void R7N8DF4OVS(eyd3OXAZgV eyd3oxazgv) {
    for (eyd3OXAZgV a : eyd3oxazgv.m4470e()) {
        this.f7522A.add(new p046b.p065c.p066a.p067a.p068a.VJEqTesDFD.HISPj7KHQ7(gm.m5307a(912),
        a.m4464a(gm.m5307a(911)).m4465a().toString(), gm.m5307a(913), gm.m5307a(914), gm.m5307a(915), gm.m5307a(916)));
    }
    this.f7523B.setAdapter(new VJEqTesDFD(this.f7522A, this));
    this.f7525D = getSharedPreferences(gm.m5307a(917), 0);
    eyd3oxazgv = getSharedPreferences(gm.m5307a(918), 0).edit();
    eyd3oxazgv.putString(gm.m5307a(919), this.f7527u);
    eyd3oxazgv.putString(gm.m5307a(920), this.f7530x);
    eyd3oxazgv.apply();
}
```

Figure4.15: With Optimizations

-- Changes start

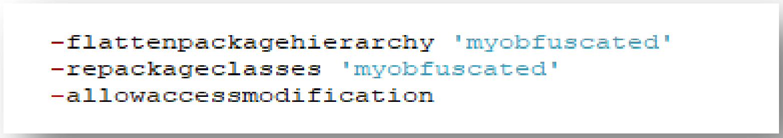
Applied optimization passes and filter settings have led to removal of unused and dead code from original program and simplify complex chunks of code wherever possible.

-- Changes end

4.1.3.4. Package Names Obfuscation:

Names of packages in our code can be obfuscated in multiple ways. All packages are obfuscated by repackaging them into one given package to reduce code size and increase accessibility by specifying any custom name to replace it with (Figure 31). Packaging can be further obfuscated by combining obfuscated classes into a single package (shown in Figure 32).

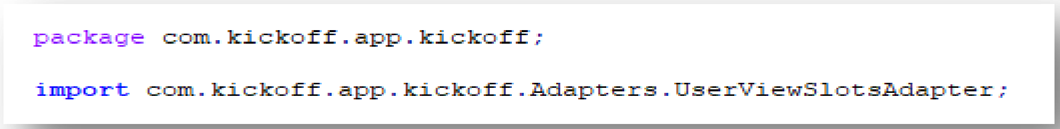
Figure 32 clearly shows how we achieved repackaging of classes and package. Whole package of obfuscated classes can be removed as well by giving no argument or leaving with an empty string ‘’. Another important flag in Pro-Guard is obfuscation of the access modifiers of classes and its members as displayed in Figure 33.



```
-flattenpackagehierarchy 'myobfuscated'
-repackageclasses 'myobfuscated'
-allowaccessmodification
```

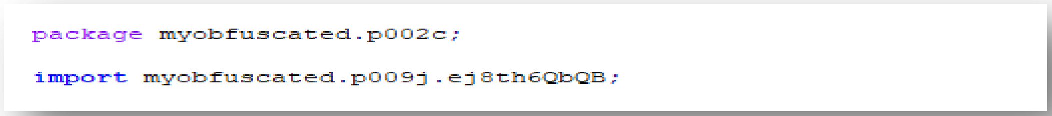
Figure4.16: Properties for Package and Access Modifiers Obfuscation

4.1.3.5. Impact of flattenpackagehierarchy:



```
package com.kickoff.app.kickoff;

import com.kickoff.app.kickoff.Adapters.UserViewSlotsAdapter;
```

Figure 4.17: Original Package naming



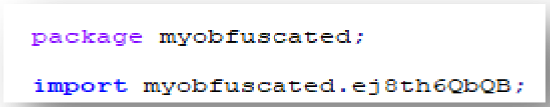
```
package myobfuscated.p002c;

import myobfuscated.p009j.ej8th6QbQB;
```

Figure 4.18: Package Obfuscation Level 1

4.1.3.6. Impact of repackageclasses:

Repackage classes obfuscates all classes into root level package thereby reducing code size as shown in below figure.

This configuration moves all classes to a single root-level package 'myobfuscated'.

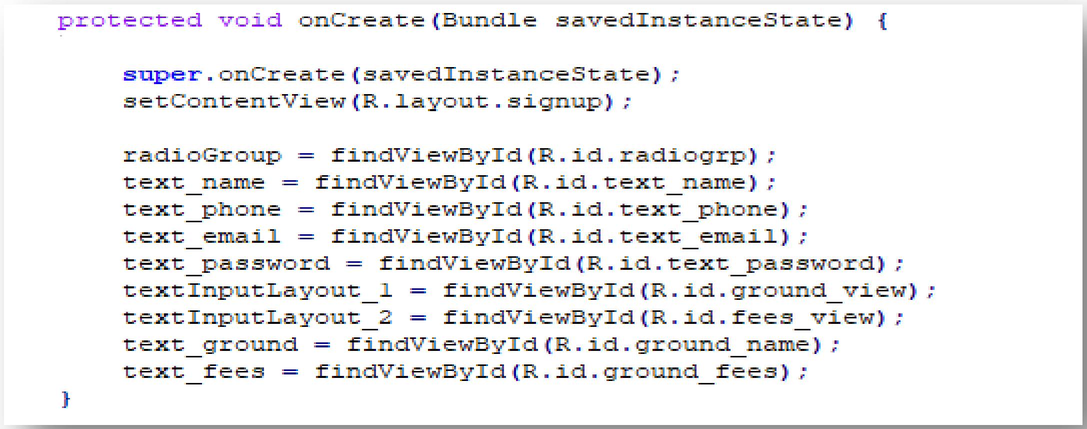


```
package myobfuscated;

import myobfuscated.ej8th6QbQB;
```

Figure 4.19: Package Obfuscation Level 2

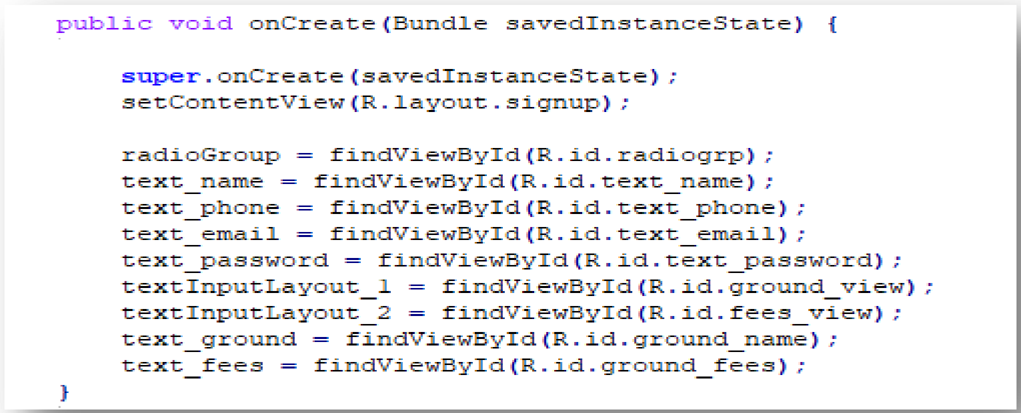
4.1.3.7. Impact of allowaccessmodification:

Below are given figures which are showing the original access modifier and modified access modifier of a class after using 'allowaccessmodification' in Pro-Guard rules file.



```
protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
    setContentView(R.layout.signup);

    radioGroup = findViewById(R.id.radiogrp);
    text_name = findViewById(R.id.text_name);
    text_phone = findViewById(R.id.text_phone);
    text_email = findViewById(R.id.text_email);
    text_password = findViewById(R.id.text_password);
    textInputLayout_1 = findViewById(R.id.ground_view);
    textInputLayout_2 = findViewById(R.id.fees_view);
    text_ground = findViewById(R.id.ground_name);
    text_fees = findViewById(R.id.ground_fees);
}
```

Figure 4.20: Class file with original access modifier



```
public void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
    setContentView(R.layout.signup);

    radioGroup = findViewById(R.id.radiogrp);
    text_name = findViewById(R.id.text_name);
    text_phone = findViewById(R.id.text_phone);
    text_email = findViewById(R.id.text_email);
    text_password = findViewById(R.id.text_password);
    textInputLayout_1 = findViewById(R.id.ground_view);
    textInputLayout_2 = findViewById(R.id.fees_view);
    text_ground = findViewById(R.id.ground_name);
    text_fees = findViewById(R.id.ground_fees);
}
```

Figure4.21: Class file with changed access modifier

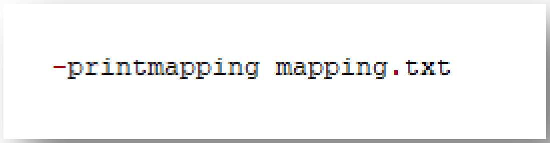
-- Changes start

Allowaccessmodification can improve the results when applying optimization to expand the access modifiers of classes and methods. But this option should not be used if those classes are intended to be private due to API calls.

-- Changes end

4.1.3.8. Obfuscation Mapping File:

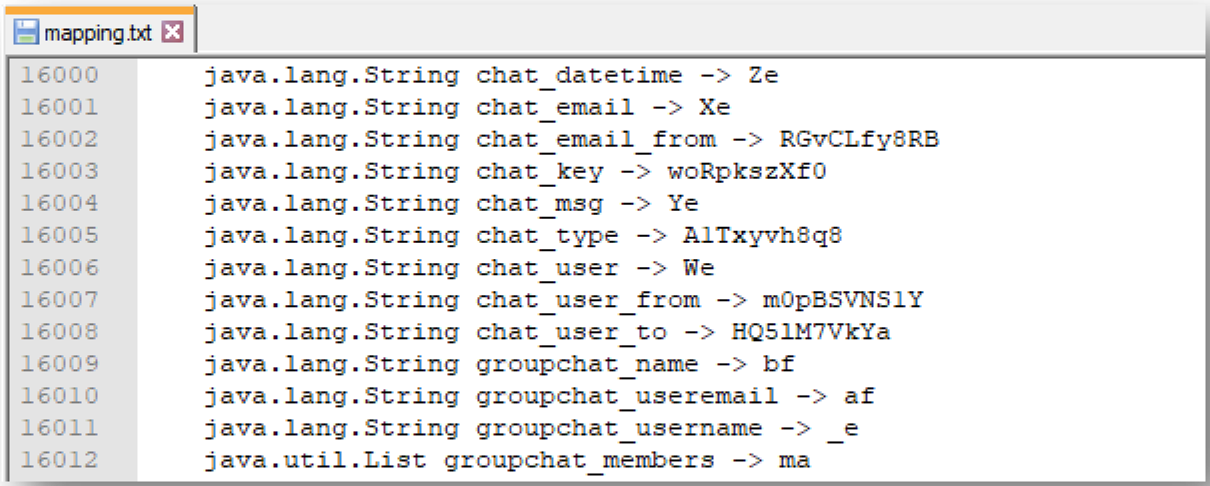
We write a reference obfuscation mapping file in our Pro-Guard rules file. In this mapping file, there is a reference between all original names and their obfuscated names whether it is a variable name, class name or some else. This information proves useful if there is a need to map obfuscated names back to their original ones. Here’s how we print the mapping file:



```
-printmapping mapping.txt
```

Figure 4.22: Property for Mapping Reference File

Below is the mapping reference from original names to obfuscated names in mapping.txt file:

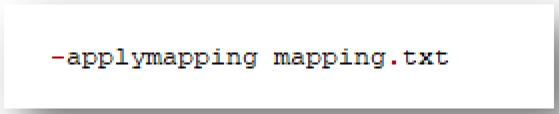


```
    java.lang.String chat_datetime -> Ze
    java.lang.String chat_email -> Xe
    java.lang.String chat_email_from -> RGvCLfy8RB
    java.lang.String chat_key -> woRpkszXf0
    java.lang.String chat_msg -> Ye
    java.lang.String chat_type -> AlTxyvh8q8
    java.lang.String chat_user -> We
    java.lang.String chat_user_from -> m0pBSVNS1Y
    java.lang.String chat_user_to -> HQ51M7VkYa
    java.lang.String groupchat_name -> bf
    java.lang.String groupchat_useremail -> af
    java.lang.String groupchat_username -> _e
    java.util.List groupchat_members -> ma
```

Figure 4.23: Mapping Reference File View

4.1.3.9. Incremental obfuscation:

We can achieve incremental obfuscation by specifying to reuse the mapping name which was printed out in the last obfuscation run of ProGuard. Classes and variables which are listed in the respective mapping file get the names indicated alongside while classes and variables that are not indicated get new names. Here’s how we introduce incremental obfuscation:



```
-applymapping mapping.txt
```

Figure 4.24: Property for incremental obfuscation by reusing mapping file

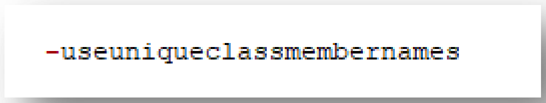
-- Changes start

With this setting provided by ProGuard, incremental obfuscation is achieved as after every build, classes methods variables and packages will get unique names compared to previously used obfuscated names to avoid easy decompilation.

-- Changes end

4.1.3.10. Unique naming of class members:

This property mentions to give the identical obfuscated names to class members that have the same original names, and different obfuscated names to class members that have different names as illustrated in below figure.



```
-useuniqueclassmembernames
```

Figure4.25: Property for unique naming of classes

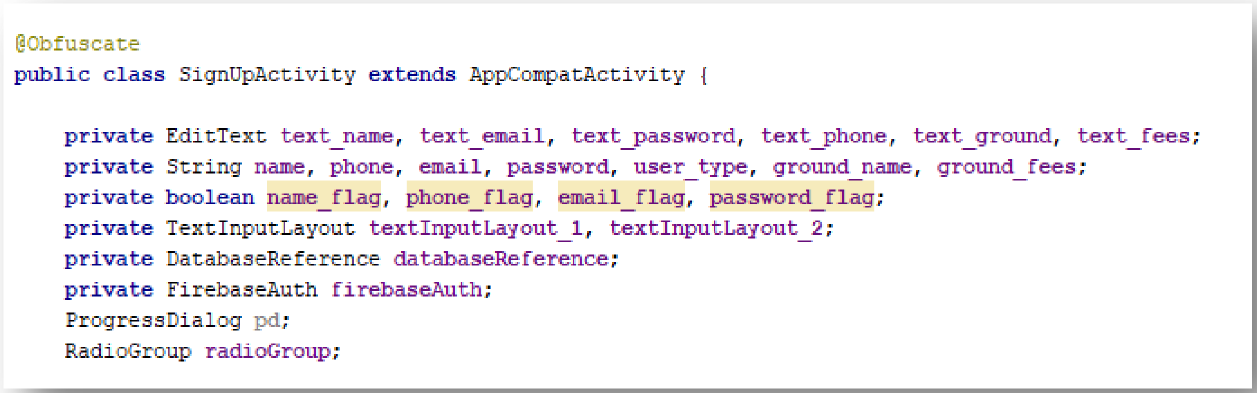
4.1.4. Limitations:

1. ProGuard does not obfuscate hard-coded strings.
2. ProGuard optimization processes assumes that the processed code never goes into busy and non-ending loops.
3. ProGuard optimization processes assumes that the processed code never throws NullPointerExceptions, ArrayIndexoutofBoundsExceptions etc in order to accomplish something useful.

4.2. Paranoid

Pro-Guard has its own limitations and one of them is not obfuscating string literals in code. For this reason, we have used Paranoid library and its Gradle plugin in Android Studio. Paranoid is an open-source third party string obfuscator library specifically for Android Applications.

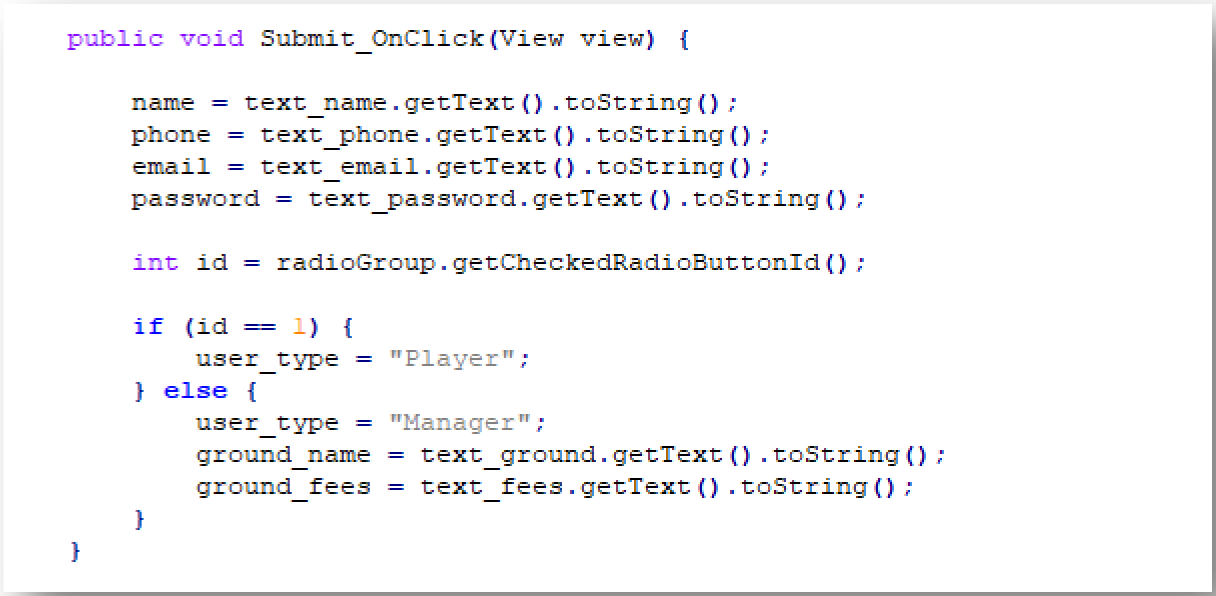
To make Paranoid functional in your code, we must annotate the classes and activities with @Obfuscate (as illustrated in the Figure A) which contain strings and need to be obfuscated. Every string literal and string constant defined in annotated classes or activities will be obfuscated.



```
@Obfuscate
public class SignUpActivity extends AppCompatActivity {

    private EditText text_name, text_email, text_password, text_phone, text_ground, text_fees;
    private String name, phone, email, password, user_type, ground_name, ground_fees;
    private boolean name_flag, phone_flag, email_flag, password_flag;
    private TextInputLayout textInputLayout_1, textInputLayout_2;
    private DatabaseReference databaseReference;
    private FirebaseAuth firebaseAuth;
    ProgressDialog pd;
    RadioGroup radioGroup;
```

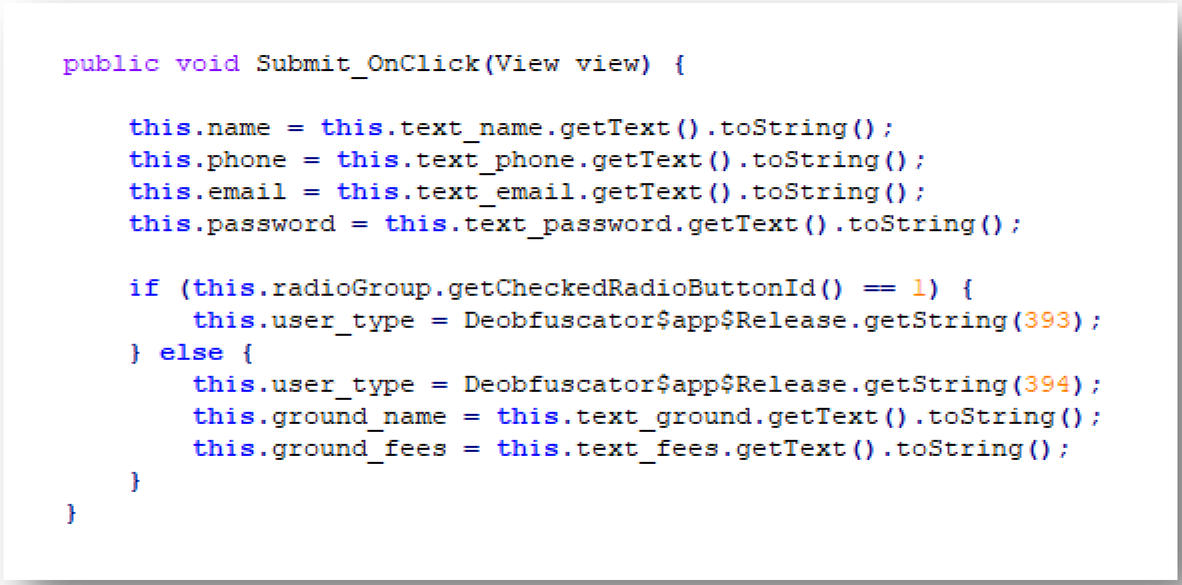
Figure4.26: Declaration of Strings

Below are the given sample string obfuscation screenshots before and after implementation of Paranoid library.



```
public void Submit_OnClick(View view) {

    name = text_name.getText().toString();
    phone = text_phone.getText().toString();
    email = text_email.getText().toString();
    password = text_password.getText().toString();

    int id = radioGroup.getCheckedRadioButtonId();

    if (id == 1) {
        user_type = "Player";
    } else {
        user_type = "Manager";
        ground_name = text_ground.getText().toString();
        ground_fees = text_fees.getText().toString();
    }
}
```

Figure4.27: Code before String obfuscation



```
public void Submit_OnClick(View view) {

    this.name = this.text_name.getText().toString();
    this.phone = this.text_phone.getText().toString();
    this.email = this.text_email.getText().toString();
    this.password = this.text_password.getText().toString();

    if (this.radioGroup.getCheckedRadioButtonId() == 1) {
        this.user_type = Deobfuscator$app$Release.getString(393);
    } else {
        this.user_type = Deobfuscator$app$Release.getString(394);
        this.ground_name = this.text_ground.getText().toString();
        this.ground_fees = this.text_fees.getText().toString();
    }
}
```

Figure4.28: Code after String obfuscation

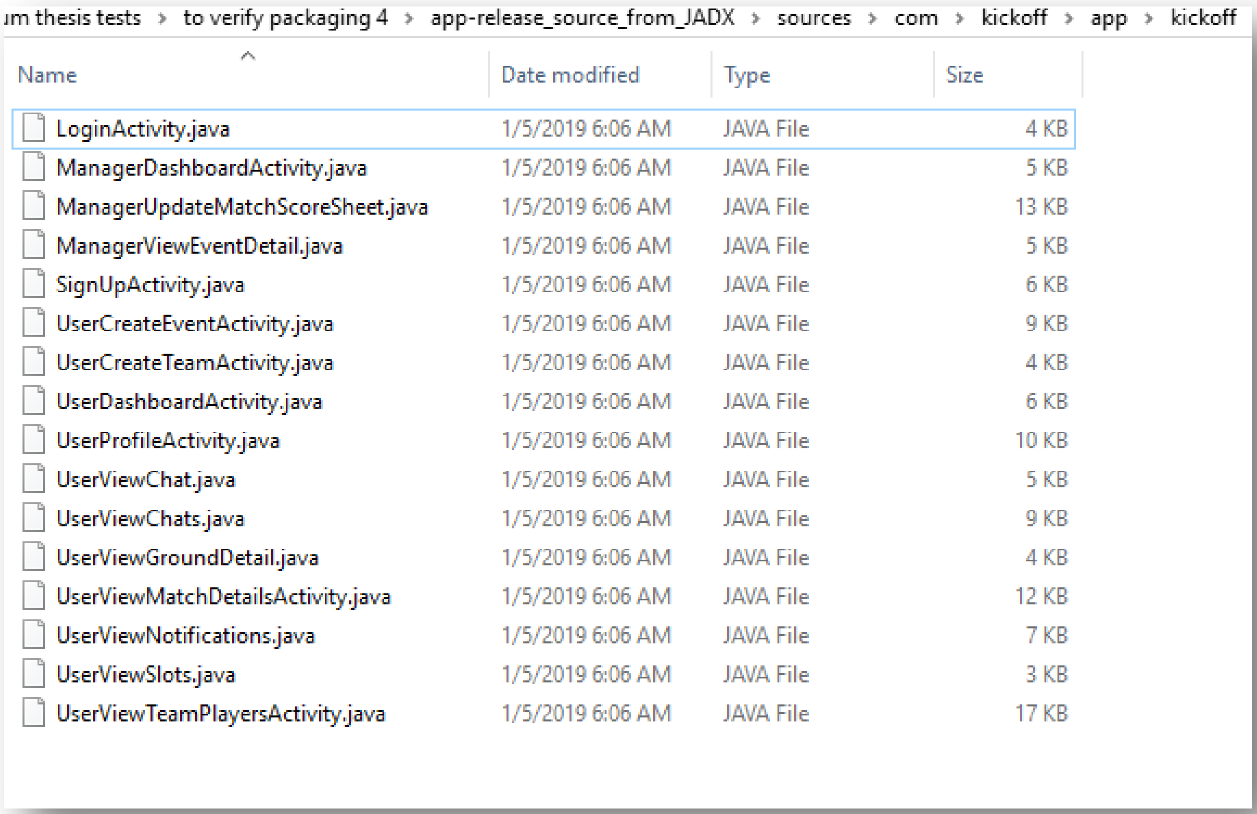
4.2.1. How it works:

Paranoid initializes an array of unique characters from all obfuscated strings and an array of indexes in the character array per each obfuscated string. When a project is compiled all hardcoded strings are swapped with function calls with a parameter which returns that specific string. The outcome is when the code is decompiled it is not possible to simply search for strings.

Major drawback of ProGuard is that it does not obfuscate hardcoded strings present in the code. It alters the class names, method names and variable names but not of variable values. Here the problem originates because anyone can reverse-engineer the apk and get the source code. Even if the code is obfuscated, anybody can get all the hardcoded strings present in the code by searching double quotes. String searching exposes interesting information in code so avoiding this can increase code security and protection significantly for more people. To overcome this problem, we have made the use of Paranoid in our work.

4.3. Allatori:

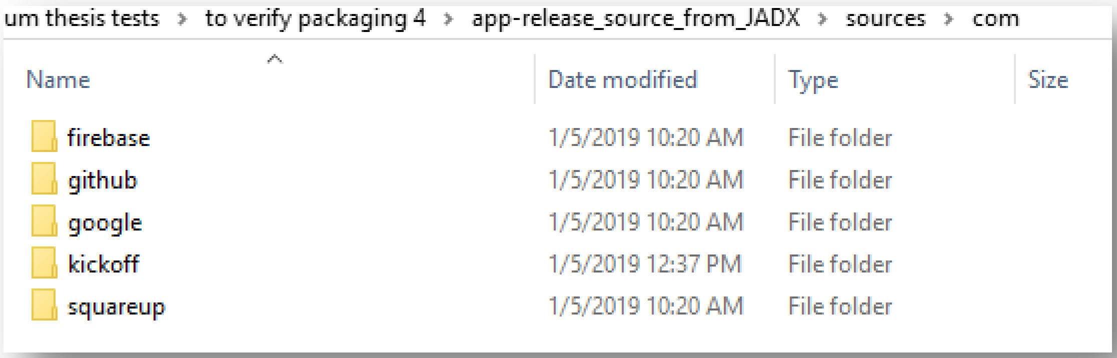
Allatori is also a java obfuscator which offers many features but the reason why we have used this third party utility is its capability to completely hide java files. It does not work with Paranoid library so we excluded Paranoid library to see the impact of Allatori. Allatori also does not obfuscate resources like ProGuard but overall ProGuard takes the lead over obfuscation. Below figures show the result of using Allatori in our project:



um thesis tests > to verify packaging 4 > app-release_source_from_JADX > sources > com > kickoff > app > kickoff

Name	Date modified	Type	Size
LoginActivity.java	1/5/2019 6:06 AM	JAVA File	4 KB
ManagerDashboardActivity.java	1/5/2019 6:06 AM	JAVA File	5 KB
ManagerUpdateMatchScoreSheet.java	1/5/2019 6:06 AM	JAVA File	13 KB
ManagerViewEventDetail.java	1/5/2019 6:06 AM	JAVA File	5 KB
SignUpActivity.java	1/5/2019 6:06 AM	JAVA File	6 KB
UserCreateEventActivity.java	1/5/2019 6:06 AM	JAVA File	9 KB
UserCreateTeamActivity.java	1/5/2019 6:06 AM	JAVA File	4 KB
UserDashboardActivity.java	1/5/2019 6:06 AM	JAVA File	6 KB
UserProfileActivity.java	1/5/2019 6:06 AM	JAVA File	10 KB
UserViewChat.java	1/5/2019 6:06 AM	JAVA File	5 KB
UserViewChats.java	1/5/2019 6:06 AM	JAVA File	9 KB
UserViewGroundDetail.java	1/5/2019 6:06 AM	JAVA File	4 KB
UserViewMatchDetailsActivity.java	1/5/2019 6:06 AM	JAVA File	12 KB
UserViewNotifications.java	1/5/2019 6:06 AM	JAVA File	7 KB
UserViewSlots.java	1/5/2019 6:06 AM	JAVA File	3 KB
UserViewTeamPlayersActivity.java	1/5/2019 6:06 AM	JAVA File	17 KB

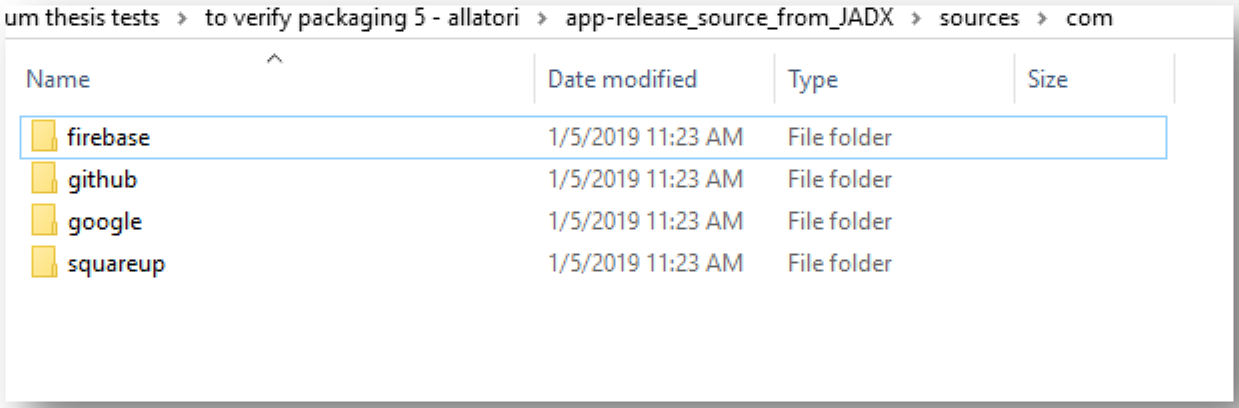
Figure4.29: Java source files inside 'kickoff' folder



um thesis tests > to verify packaging 4 > app-release_source_from_JADX > sources > com

Name	Date modified	Type	Size
firebase	1/5/2019 10:20 AM	File folder	
github	1/5/2019 10:20 AM	File folder	
google	1/5/2019 10:20 AM	File folder	
kickoff	1/5/2019 12:37 PM	File folder	
squareup	1/5/2019 10:20 AM	File folder	

Figure4.30: Java source files folder 'kickoff' before using Allatori


um thesis tests › to verify packaging 5 - allatori › app-release_source_from_JADX › sources › com

Name	Date modified	Type	Size
firebase	1/5/2019 11:23 AM	File folder	
github	1/5/2019 11:23 AM	File folder	
google	1/5/2019 11:23 AM	File folder	
squareup	1/5/2019 11:23 AM	File folder	

Figure4.31: Java source files folder 'kickoff' after using Allatori

CHAPTER 5

RESULTS AND LIMITATIONS

Chapter # 5 Results and Limitation

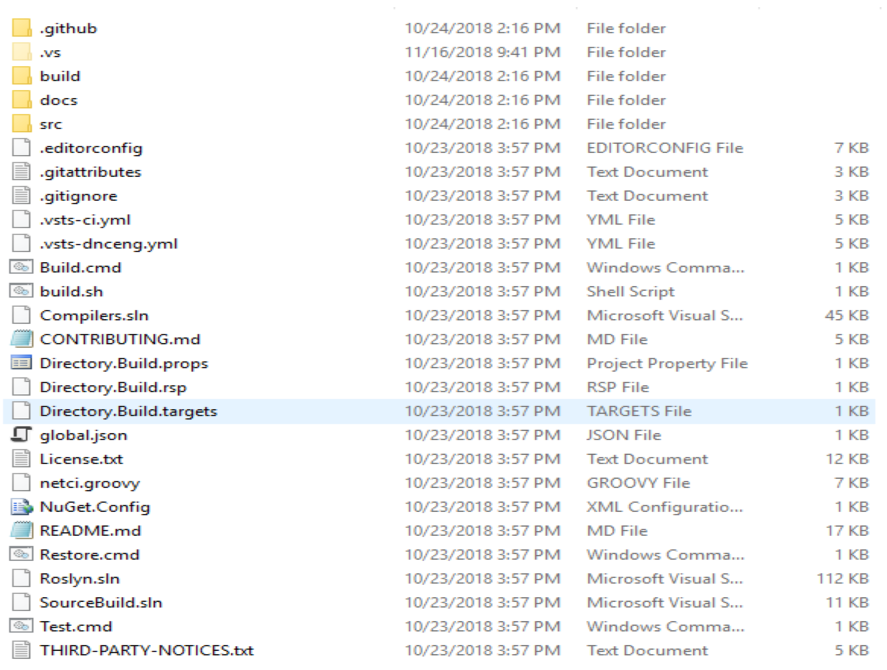
This chapter is designed to elaborate the experiments performed, results concluded and limitations. **Section 5.1** is about Experiments, **Section 5.2** is about Results, **Section 5.3** discusses the limitations of this research.

5.1. Experiments:

5.1.1. VOT4CS:

Initially for the purpose of code obfuscation, we selected .NET platform, VOT4CS tool and Roslyn Code analyzer to improve the Collberg's four taxonomies.

1. Firstly Roslyn Git library was integrated to VS2015, VS2107 as shown below:



.github	10/24/2018 2:16 PM	File folder	
.vs	11/16/2018 9:41 PM	File folder	
build	10/24/2018 2:16 PM	File folder	
docs	10/24/2018 2:16 PM	File folder	
src	10/24/2018 2:16 PM	File folder	
.editorconfig	10/23/2018 3:57 PM	EDITORCONFIG File	7 KB
.gitattributes	10/23/2018 3:57 PM	Text Document	3 KB
.gitignore	10/23/2018 3:57 PM	Text Document	3 KB
.vsts-ci.yml	10/23/2018 3:57 PM	YML File	5 KB
.vsts-dnceng.yml	10/23/2018 3:57 PM	YML File	5 KB
Build.cmd	10/23/2018 3:57 PM	Windows Comma...	1 KB
build.sh	10/23/2018 3:57 PM	Shell Script	1 KB
Compilers.sln	10/23/2018 3:57 PM	Microsoft Visual S...	45 KB
CONTRIBUTING.md	10/23/2018 3:57 PM	MD File	5 KB
Directory.Build.props	10/23/2018 3:57 PM	Project Property File	1 KB
Directory.Build.rsp	10/23/2018 3:57 PM	RSP File	1 KB
Directory.Build.targets	10/23/2018 3:57 PM	TARGETS File	1 KB
global.json	10/23/2018 3:57 PM	JSON File	1 KB
License.txt	10/23/2018 3:57 PM	Text Document	12 KB
netci.groovy	10/23/2018 3:57 PM	GROOVY File	7 KB
NuGet.Config	10/23/2018 3:57 PM	XML Configuratio...	1 KB
README.md	10/23/2018 3:57 PM	MD File	17 KB
Restore.cmd	10/23/2018 3:57 PM	Windows Comma...	1 KB
Roslyn.sln	10/23/2018 3:57 PM	Microsoft Visual S...	112 KB
SourceBuild.sln	10/23/2018 3:57 PM	Microsoft Visual S...	11 KB
Test.cmd	10/23/2018 3:57 PM	Windows Comma...	1 KB
THIRD-PARTY-NOTICES.txt	10/23/2018 3:57 PM	Text Document	5 KB

Figure5.1: Roslyn Git-hub Plugin

2. Roslyn Code analyzer was integrated with VS2015 for syntax analyzer by using nugget packages, and performed analysis of code as shown below:

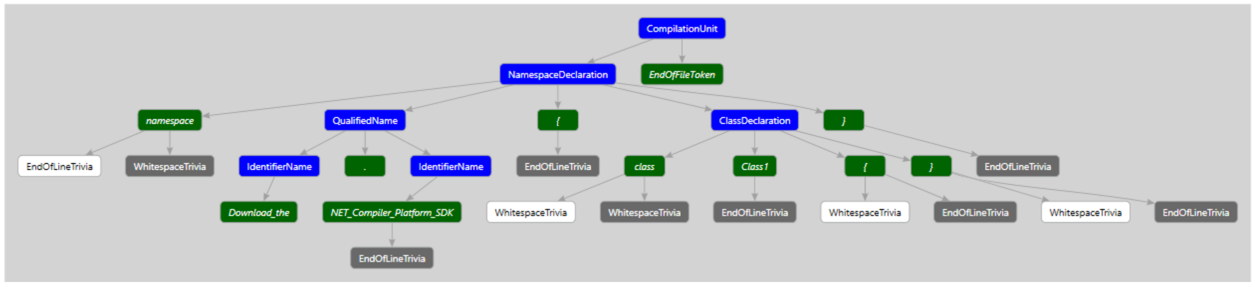
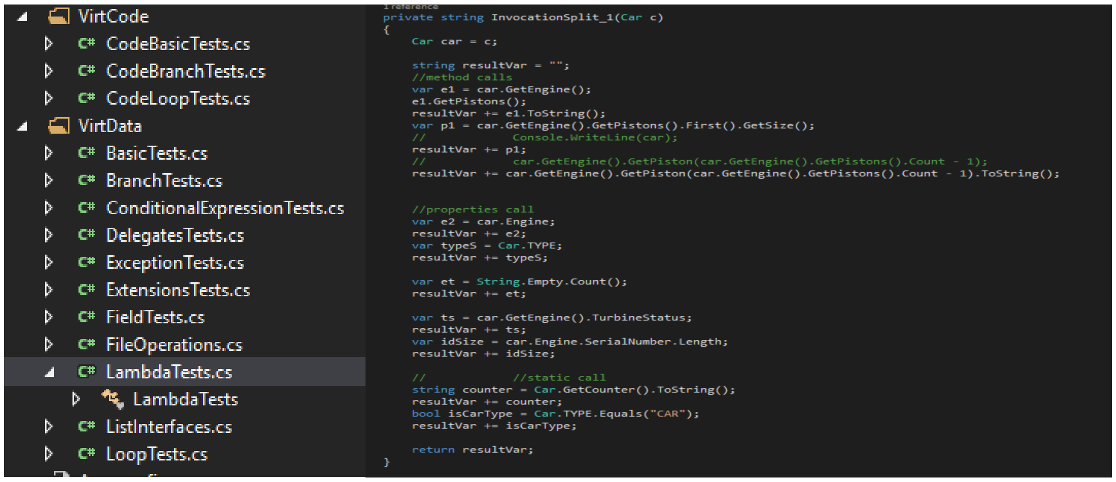


Figure5.2: Syntax Graph

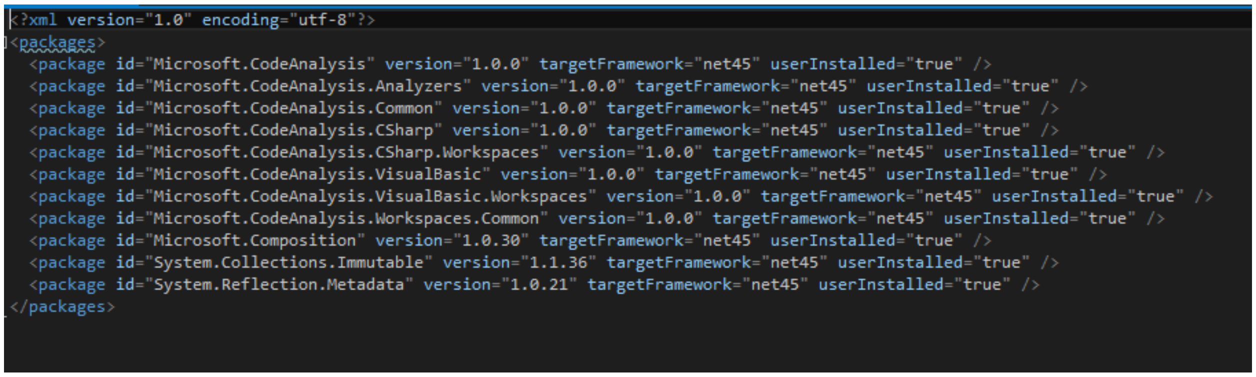
3. After that VOT4CS was integrated with VS 2015, set of rules were defined for the sake of improvement of Collberg’s taxonomy. Code Structure, app Setting and .cs is shown in figure below



```
◢ VirtCode
  ▷ C# CodeBasicTests.cs
  ▷ C# CodeBranchTests.cs
  ▷ C# CodeLoopTests.cs
◢ VirtData
  ▷ C# BasicTests.cs
  ▷ C# BranchTests.cs
  ▷ C# ConditionalExpressionTests.cs
  ▷ C# DelegatesTests.cs
  ▷ C# ExceptionTests.cs
  ▷ C# ExtensionsTests.cs
  ▷ C# FieldTests.cs
  ▷ C# FileOperations.cs
  ◢ C# LambdaTests.cs
      ▷ LambdaTests
  ▷ C# ListInterfaces.cs
  ▷ C# LoopTests.cs
```

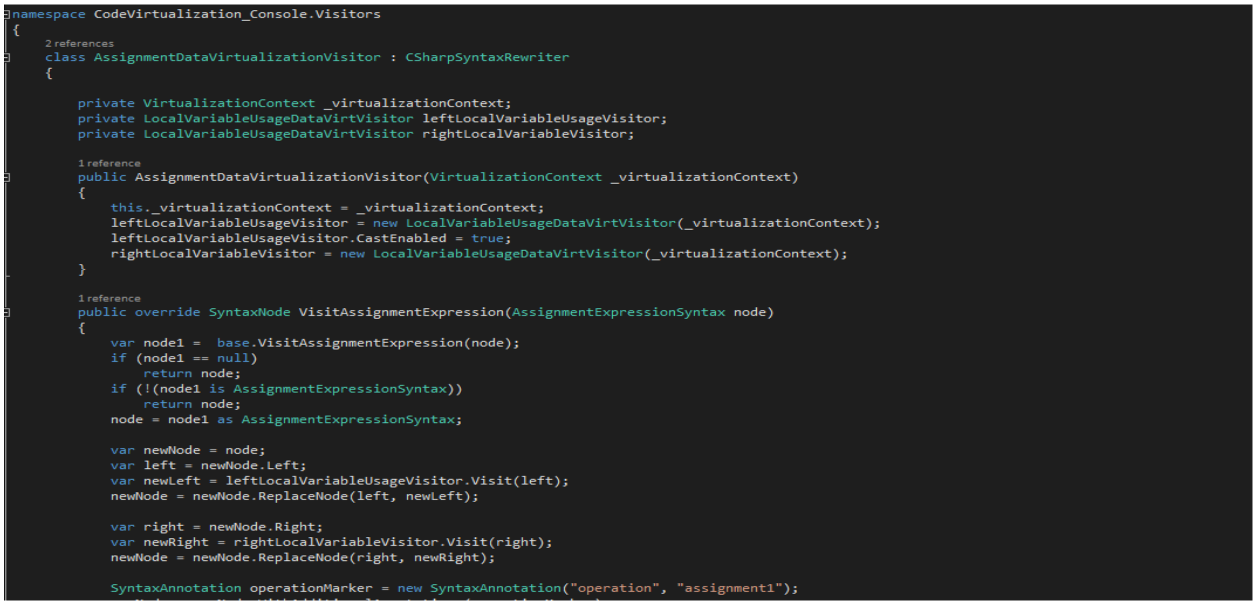
```
1 reference
private string InvocationSplit_1(Car c)
{
    Car car = c;

    string resultVar = "";
    //method calls
    var e1 = car.GetEngine();
    e1.GetPistons();
    resultVar += e1.ToString();
    var p1 = car.GetEngine().GetPistons().First().GetSize();
    //              Console.WriteLine(car);
    resultVar += p1;
    //              car.GetEngine().GetPiston(car.GetEngine().GetPistons().Count - 1);
    resultVar += car.GetEngine().GetPiston(car.GetEngine().GetPistons().Count - 1).ToString();


    //properties call
    var e2 = car.Engine;
    resultVar += e2;
    var typeS = Car.TYPE;
    resultVar += typeS;

    var et = String.Empty.Count();
    resultVar += et;

    var ts = car.GetEngine().TurbineStatus;
    resultVar += ts;
    var idSize = car.Engine.SerialNumber.Length;
    resultVar += idSize;

    //              //static call
    string counter = Car.GetCounter().ToString();
    resultVar += counter;
    bool isCarType = Car.TYPE.Equals("CAR");
    resultVar += isCarType;

    return resultVar;
}
```

Figure5.3: Setting Project and Defining Rules



```xml
<?xml version="1.0" encoding="utf-8"?>
<packages>
  <package id="Microsoft.CodeAnalysis" version="1.0.0" targetFramework="net45" userInstalled="true" />
  <package id="Microsoft.CodeAnalysis.Analyzers" version="1.0.0" targetFramework="net45" userInstalled="true" />
  <package id="Microsoft.CodeAnalysis.Common" version="1.0.0" targetFramework="net45" userInstalled="true" />
  <package id="Microsoft.CodeAnalysis.CSharp" version="1.0.0" targetFramework="net45" userInstalled="true" />
  <package id="Microsoft.CodeAnalysis.CSharp.Workspaces" version="1.0.0" targetFramework="net45" userInstalled="true" />
  <package id="Microsoft.CodeAnalysis.VisualBasic" version="1.0.0" targetFramework="net45" userInstalled="true" />
  <package id="Microsoft.CodeAnalysis.VisualBasic.Workspaces" version="1.0.0" targetFramework="net45" userInstalled="true" />
  <package id="Microsoft.CodeAnalysis.Workspaces.Common" version="1.0.0" targetFramework="net45" userInstalled="true" />
  <package id="Microsoft.Composition" version="1.0.30" targetFramework="net45" userInstalled="true" />
  <package id="System.Collections.Immutable" version="1.1.36" targetFramework="net45" userInstalled="true" />
  <package id="System.Reflection.Metadata" version="1.0.21" targetFramework="net45" userInstalled="true" />
</packages>
```

Figure5.4: Adding Version of Code Analyzer to app Setting



```
namespace CodeVirtualization_Console.Visitors
{
    2 references
    class AssignmentDataVirtualizationVisitor : CSharpSyntaxRewriter
    {

        private VirtualizationContext _virtualizationContext;
        private LocalVariableUsageDataVirtVisitor leftLocalVariableUsageVisitor;
        private LocalVariableUsageDataVirtVisitor rightLocalVariableVisitor;

        1 reference
        public AssignmentDataVirtualizationVisitor(VirtualizationContext _virtualizationContext)
        {
            this._virtualizationContext = _virtualizationContext;
            leftLocalVariableUsageVisitor = new LocalVariableUsageDataVirtVisitor(_virtualizationContext);
            leftLocalVariableUsageVisitor.CastEnabled = true;
            rightLocalVariableVisitor = new LocalVariableUsageDataVirtVisitor(_virtualizationContext);
        }

        1 reference
        public override SyntaxNode VisitAssignmentExpression(AssignmentExpressionSyntax node)
        {
            var node1 =  base.VisitAssignmentExpression(node);
            if (node1 == null)
                return node;
            if (!(node1 is AssignmentExpressionSyntax))
                return node;
            node = node1 as AssignmentExpressionSyntax;

            var newNode = node;
            var left = newNode.Left;
            var newLeft = leftLocalVariableUsageVisitor.Visit(left);
            newNode = newNode.ReplaceNode(left, newLeft);

            var right = newNode.Right;
            var newRight = rightLocalVariableVisitor.Visit(right);
            newNode = newNode.ReplaceNode(right, newRight);

            SyntaxAnnotation operationMarker = new SyntaxAnnotation("operation", "assignment1");
```

Figure5.5: Virtualization Rules

Results:

After performing experiments, we identified that, there were certain improvements needed for VOT4CS.

1. VOT4CS don't handle lambda expression

Limitation 1:

Roslyn doesn't support to implement the lambda expression.

2. VOT4CS don't handle try/catch

Limitation 2:

Roslyn doesn't support to implement the finally statement.

3. VOT4CS can't store data in struct

Limitation 3:

It gives error after compilation when we are trying to access the attribute of struct.

5.1.2. Hybrid Approach for Android:

Finally, we selected hybrid method to obfuscate the android application by using pro-guard, paranoid and allatori to compile all feature set in one package. Hence pro-guard provided for class, package, method, statements and variables obfuscation. Where pro-guard doesn't facilitates the string obfuscation, so to compensate this gap we used paranoid. And for the sake of control flow and de-obfuscation java files, we used allatori. We also used de-obfuscator "http://www.javadecompilers.com/" to de-obfuscate our APK and performed reverse engineering to get back the algorithm. Below are results.

5.1.2.1. Case Studies:

Three projects have been selected for experimentation, where set of rules are implemented with configuration settings.

Sr. #	Test	Before obfuscation Cost	After obfuscation Cost
1	Project 1	0	- 0.08
2	Project 2	0	- 0.05
3	Project 3	0	- 0.01

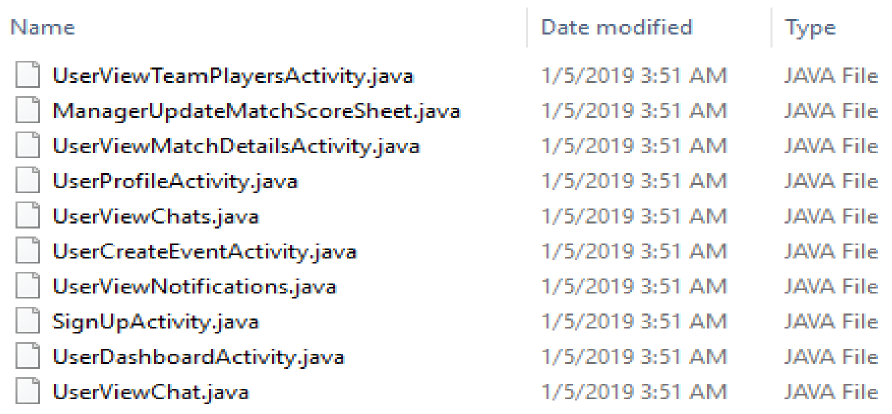
Table 5.1: Before and After Code Obfuscation

Test Case 1:

Class name obfuscation:

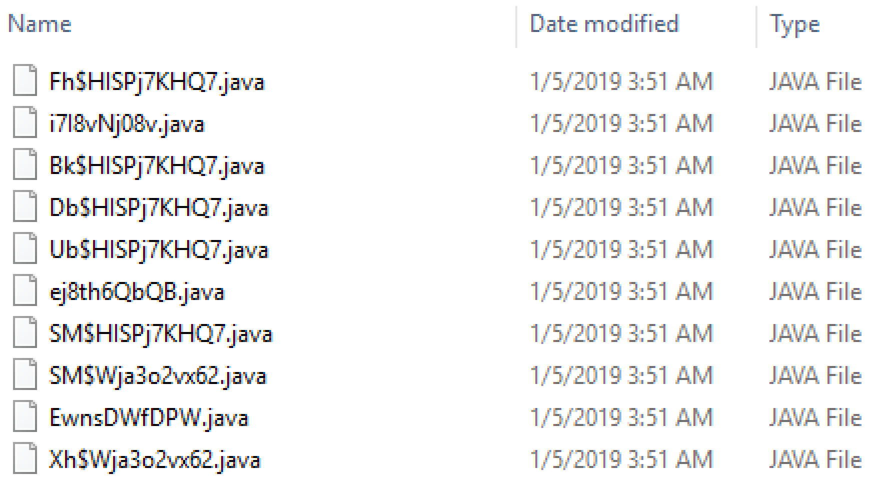
Project 1:

Before Obfuscation:



Name	Date modified	Type
UserViewTeamPlayersActivity.java	1/5/2019 3:51 AM	JAVA File
ManagerUpdateMatchScoreSheet.java	1/5/2019 3:51 AM	JAVA File
UserViewMatchDetailsActivity.java	1/5/2019 3:51 AM	JAVA File
UserProfileActivity.java	1/5/2019 3:51 AM	JAVA File
UserViewChats.java	1/5/2019 3:51 AM	JAVA File
UserCreateEventActivity.java	1/5/2019 3:51 AM	JAVA File
UserViewNotifications.java	1/5/2019 3:51 AM	JAVA File
SignUpActivity.java	1/5/2019 3:51 AM	JAVA File
UserDashboardActivity.java	1/5/2019 3:51 AM	JAVA File
UserViewChat.java	1/5/2019 3:51 AM	JAVA File

After Obfuscation:



Name	Date modified	Type
Fh$HISPj7KHQ7.java	1/5/2019 3:51 AM	JAVA File
i7l8vNj08v.java	1/5/2019 3:51 AM	JAVA File
Bk$HISPj7KHQ7.java	1/5/2019 3:51 AM	JAVA File
Db$HISPj7KHQ7.java	1/5/2019 3:51 AM	JAVA File
Ub$HISPj7KHQ7.java	1/5/2019 3:51 AM	JAVA File
ej8th6QbQB.java	1/5/2019 3:51 AM	JAVA File
SM$HISPj7KHQ7.java	1/5/2019 3:51 AM	JAVA File
SM$Wja3o2vx62.java	1/5/2019 3:51 AM	JAVA File
EwnsDWfDPW.java	1/5/2019 3:51 AM	JAVA File
Xh$Wja3o2vx62.java	1/5/2019 3:51 AM	JAVA File

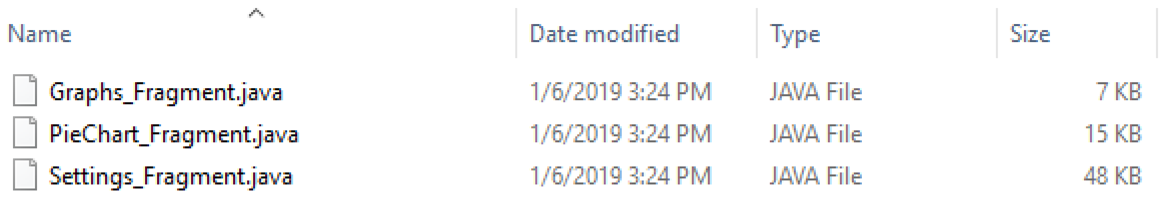
Class names have been obfuscated aggressively by using randomized dictionary placed in our project. No two classes can get identical name, making it difficult for an attacker to understand and manipulate the code. This specific type of renaming obfuscation is enabled by using following command in proguard rules file:

classobfuscationdictionary dictionary.txt

Exactly same class obfuscation technique has been sued for other two test projects.

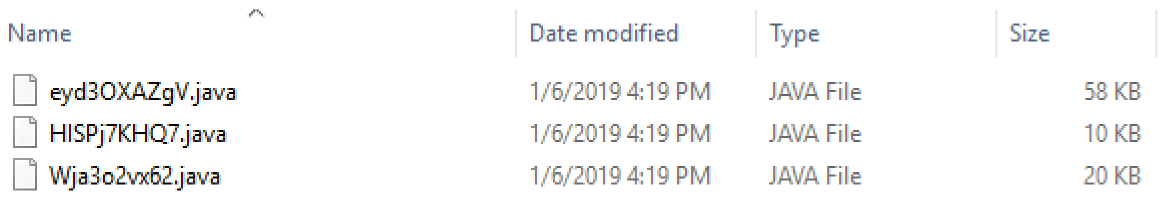
Project 2:

Before Obfuscation:



Name	Date modified	Type	Size
Graphs_Fragment.java	1/6/2019 3:24 PM	JAVA File	7 KB
PieChart_Fragment.java	1/6/2019 3:24 PM	JAVA File	15 KB
Settings_Fragment.java	1/6/2019 3:24 PM	JAVA File	48 KB

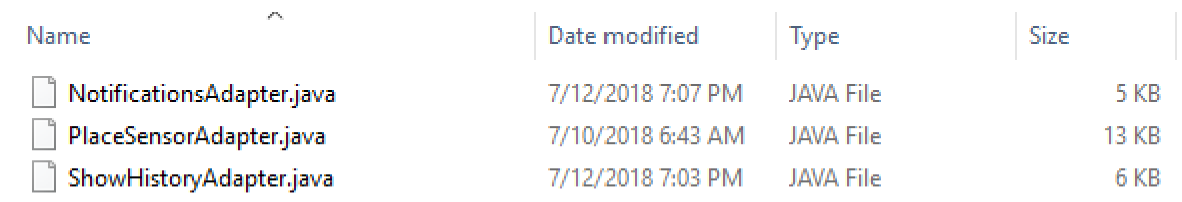
After Obfuscation:



Name	Date modified	Type	Size
eyd3OXAZgV.java	1/6/2019 4:19 PM	JAVA File	58 KB
HISPj7KHQ7.java	1/6/2019 4:19 PM	JAVA File	10 KB
Wja3o2vx62.java	1/6/2019 4:19 PM	JAVA File	20 KB

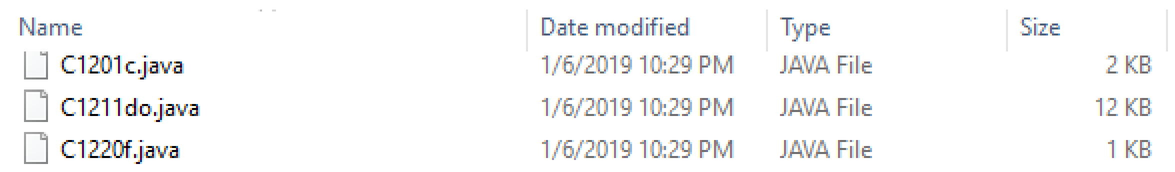
Project 3:

Before Obfuscation:



Name	Date modified	Type	Size
NotificationsAdapter.java	7/12/2018 7:07 PM	JAVA File	5 KB
PlaceSensorAdapter.java	7/10/2018 6:43 AM	JAVA File	13 KB
ShowHistoryAdapter.java	7/12/2018 7:03 PM	JAVA File	6 KB

After Obfuscation:



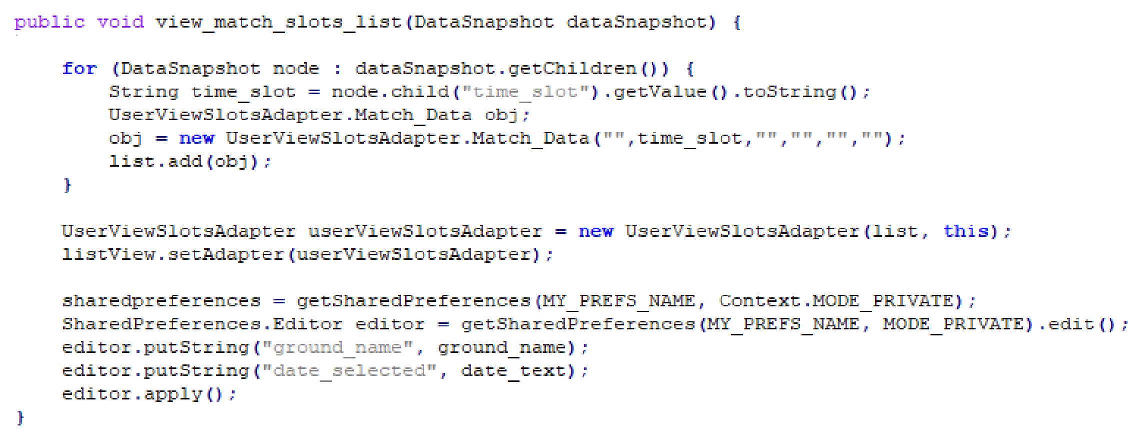
Name	Date modified	Type	Size
C1201c.java	1/6/2019 10:29 PM	JAVA File	2 KB
C1211do.java	1/6/2019 10:29 PM	JAVA File	12 KB
C1220f.java	1/6/2019 10:29 PM	JAVA File	1 KB

Test Case 2:

Code obfuscation:

Project 1:

Before Obfuscation:



```
public void view_match_slots_list(DataSnapshot dataSnapshot) {

    for (DataSnapshot node : dataSnapshot.getChildren()) {
        String time_slot = node.child("time_slot").getValue().toString();
        UserViewSlotsAdapter.Match_Data obj;
        obj = new UserViewSlotsAdapter.Match_Data("",time_slot,"","","","");
        list.add(obj);
    }

    UserViewSlotsAdapter userViewSlotsAdapter = new UserViewSlotsAdapter(list, this);
    listView.setAdapter(userViewSlotsAdapter);

    sharedpreferences = getSharedPreferences(MY_PREFS_NAME, Context.MODE_PRIVATE);
    SharedPreferences.Editor editor = getSharedPreferences(MY_PREFS_NAME, MODE_PRIVATE).edit();
    editor.putString("ground_name", ground_name);
    editor.putString("date_selected", date_text);
    editor.apply();
}
```

After Obfuscation:

```
public void R7N8DF4OVS(eyd3OXAZgV eyd3oxazgv) {
    for (eyd3OXAZgV a : eyd3oxazgv.m1709e()) {
        this.f7522A.add(new p046b.p134d.p135a.p136a.p137a.VJEqTesDFD.HISPj7KHQ7(p140c.p141a.p142a.HISPj7KHQ7
        .m1899a(887), a.m1703a(p140c.p141a.p142a.HISPj7KHQ7.m1899a(886)).m1704a().toString(), p140c.p141a.p142a.
        HISPj7KHQ7.m1899a(888), p140c.p141a.p142a.HISPj7KHQ7.m1899a(889), p140c.p141a.p142a.HISPj7KHQ7.m1899a(890),
        p140c.p141a.p142a.HISPj7KHQ7.m1899a(891)));
    }
    this.f7523B.setAdapter(new VJEqTesDFD(this.f7522A, this));
    this.f7525D = getSharedPreferences(p140c.p141a.p142a.HISPj7KHQ7.m1899a(892), 0);
    eyd3oxazgv = getSharedPreferences(p140c.p141a.p142a.HISPj7KHQ7.m1899a(893), 0).edit();
    eyd3oxazgv.putString(p140c.p141a.p142a.HISPj7KHQ7.m1899a(894), this.f7527u);
    eyd3oxazgv.putString(p140c.p141a.p142a.HISPj7KHQ7.m1899a(895), this.f7530x);
    eyd3oxazgv.apply();
}
```

Class names have been obfuscated aggressively by using randomized dictionary placed in our project. No two classes can get identical name, making it difficult for an attacker to understand and manipulate the code. This specific type of renaming obfuscation is enabled by using following command in proguard rules file:

classobfuscationdictionary dictionary.txt

Exactly same class obfuscation technique has been sued for other two test projects.

Project 2:

Before Obfuscation:

```
public void setup_chart2(){

    List<BarEntry> chartEntryList2 = new ArrayList<>();

    for(int i=8; i<itemWeightList.size()-16; i++){
        chartEntryList2.add(new BarEntry(Float.parseFloat(itemWeightList.get(i).getItem())
    }

    BarDataSet barDataSet2 = new BarDataSet(chartEntryList2,"Items Weight");
    barDataSet2.setColors(ColorTemplate.JOYFUL_COLORS);

    BarData barData2 = new BarData(barDataSet2);
    Chart2.getDescription().setText("Scale 8-16");
    Chart2.getDescription().setTextSize(14f);
    Chart2.setData(barData2);
    Chart2.animateY(1000);
    Chart2.invalidate();
}
```

After Obfuscation:

```
public void m10c() {
    List arrayList = new ArrayList();
    for (int i = 16; i < this.f402c.size() - 8; i++) {
        arrayList.add(new eyd3OXAZgV(Float.parseFloat(((myobfuscated.HISPj7KHQ7) this.f402c.get(i)).HISPj7KHQ7()), Fl
    }
    new Wja3o2vx62(arrayList, HISPj7KHQ7.HISPj7KHQ7.HISPj7KHQ7.HISPj7KHQ7.HISPj7KHQ7(13)).HISPj7KHQ7(com.github.mikep
    com.github.mikephil.charting.DxDJysLV5r.cWbN6pumKk hISPj7KHQ7 = new com.github.mikephil.charting.DxDJysLV5r.HISPj
    this.f400a.getDescription().HISPj7KHQ7(HISPj7KHQ7.HISPj7KHQ7.HISPj7KHQ7.HISPj7KHQ7.HISPj7KHQ7(14));
    this.f400a.getDescription().eyd3OXAZgV(14.0f);
    this.f400a.setData(hISPj7KHQ7);
    this.f400a.HISPj7KHQ7(1000);
    this.f400a.invalidate();
}
```

Project 3:

Before Obfuscation:

```
public void Submit_OnClick(View view){

    text_password = edittext_password.getText().toString();
    text_new_password = edittext_new_password.getText().toString();

    if (text_password.equals("") || text_new_password.equals("")) {
        Toast.makeText(getApplicationContext(), "Please enter all fields", Toast.LENGTH_SHORT).show();
    } else if(text_password.equals(text_new_password)){
        dbHelper.update_user_password("admin",text_password);
        Toast.makeText(getApplicationContext(), "Password changed", Toast.LENGTH_SHORT).show();
        finish();
    } else{
        Toast.makeText(getApplicationContext(), "Passwords do not match", Toast.LENGTH_SHORT).show();
    }
}
```

After Obfuscation:

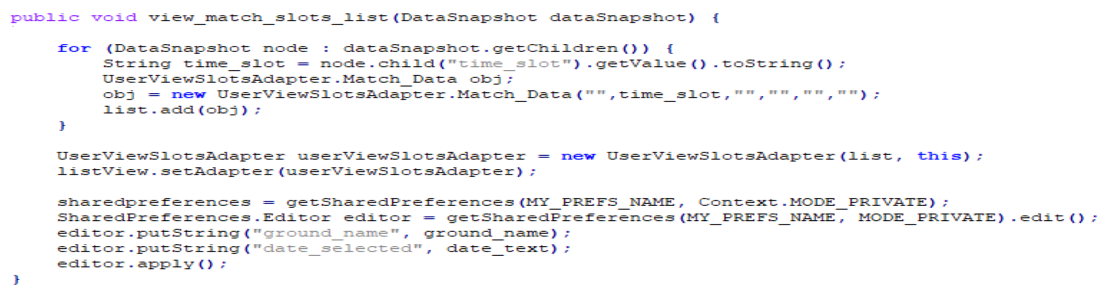
```
public void Submit_OnClick(View view) {
    this.XDUV3WhqEn = this.MWRUB7CEQh.getText().toString();
    this.UiUtJawIF9 = this.W8U6BvWWNv.getText().toString();
    if (!this.XDUV3WhqEn.equals(jk.HISPj7KHQ7(1))) {
        if (!this.UiUtJawIF9.equals(jk.HISPj7KHQ7(2))) {
            if (this.XDUV3WhqEn.equals(this.UiUtJawIF9)) {
                this.MOkcCailD0.DxDJysLV5r(jk.HISPj7KHQ7(4), this.XDUV3WhqEn);
                Toast.makeText(getApplicationContext(), jk.HISPj7KHQ7(5), 0).show();
                finish();
                return;
            }
            Toast.makeText(getApplicationContext(), jk.HISPj7KHQ7(6), 0).show();
            return;
        }
    }
    Toast.makeText(getApplicationContext(), jk.HISPj7KHQ7(3), 0).show();
}
```

Test Case 3:

Method obfuscation:

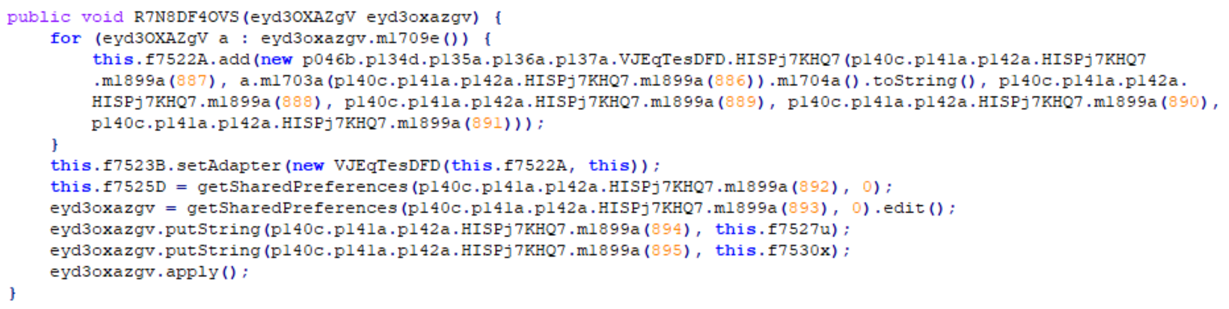
Project 1:

Before Obfuscation:



```
public void view_match_slots_list(DataSnapshot dataSnapshot) {

    for (DataSnapshot node : dataSnapshot.getChildren()) {
        String time_slot = node.child("time_slot").getValue().toString();
        UserViewSlotsAdapter.Match_Data obj;
        obj = new UserViewSlotsAdapter.Match_Data("",time_slot,"","","","");
        list.add(obj);
    }

    UserViewSlotsAdapter userViewSlotsAdapter = new UserViewSlotsAdapter(list, this);
    listView.setAdapter(userViewSlotsAdapter);

    sharedpreferences = getSharedPreferences(MY_PREFS_NAME, Context.MODE_PRIVATE);
    SharedPreferences.Editor editor = getSharedPreferences(MY_PREFS_NAME, MODE_PRIVATE).edit();
    editor.putString("ground_name", ground_name);
    editor.putString("date_selected", date_text);
    editor.apply();
}
```

After Obfuscation:



```
public void R7N8DF4OVS(eyd3OXAZgV eyd3oxazgv) {
    for (eyd3OXAZgV a : eyd3oxazgv.m1709e()) {
        this.f7522A.add(new p046b.p134d.p135a.p136a.p137a.VJEqTesDFD.HISPj7KHQ7(p140c.p141a.p142a.HISPj7KHQ7
        .m1899a(887), a.m1703a(p140c.p141a.p142a.HISPj7KHQ7.m1899a(886)).m1704a().toString(), p140c.p141a.p142a.
        HISPj7KHQ7.m1899a(888), p140c.p141a.p142a.HISPj7KHQ7.m1899a(889), p140c.p141a.p142a.HISPj7KHQ7.m1899a(890),
        p140c.p141a.p142a.HISPj7KHQ7.m1899a(891)));
    }
    this.f7523B.setAdapter(new VJEqTesDFD(this.f7522A, this));
    this.f7525D = getSharedPreferences(p140c.p141a.p142a.HISPj7KHQ7.m1899a(892), 0);
    eyd3oxazgv = getSharedPreferences(p140c.p141a.p142a.HISPj7KHQ7.m1899a(893), 0).edit();
    eyd3oxazgv.putString(p140c.p141a.p142a.HISPj7KHQ7.m1899a(894), this.f7527u);
    eyd3oxazgv.putString(p140c.p141a.p142a.HISPj7KHQ7.m1899a(895), this.f7530x);
    eyd3oxazgv.apply();
}
```

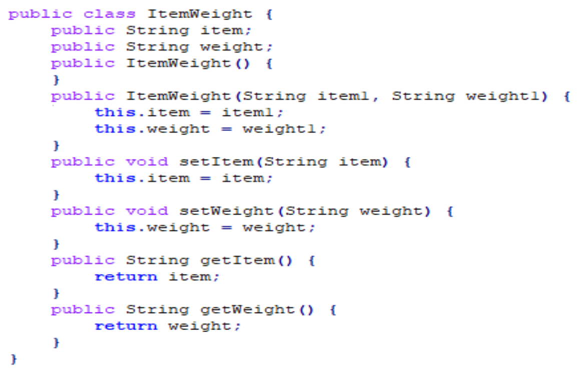
Class member variables and methods have been obfuscated aggressively by using randomized dictionary placed in our project. No two methods or variables can get identical names, making it quite difficult for an attacker to understand and manipulate the code. This specific type of renaming obfuscation is enabled by using following command in proguard rules file:

obfuscationdictionary dictionary.txt

Exactly same class methods and variables obfuscation technique has been used for other two test projects.

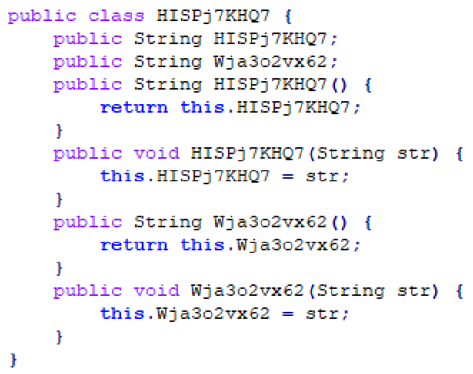
Project 2:

Before Obfuscation:



```
public class ItemWeight {
    public String item;
    public String weight;
    public ItemWeight() {
    }
    public ItemWeight(String item1, String weight1) {
        this.item = item1;
        this.weight = weight1;
    }
    public void setItem(String item) {
        this.item = item;
    }
    public void setWeight(String weight) {
        this.weight = weight;
    }
    public String getItem() {
        return item;
    }
    public String getWeight() {
        return weight;
    }
}
```

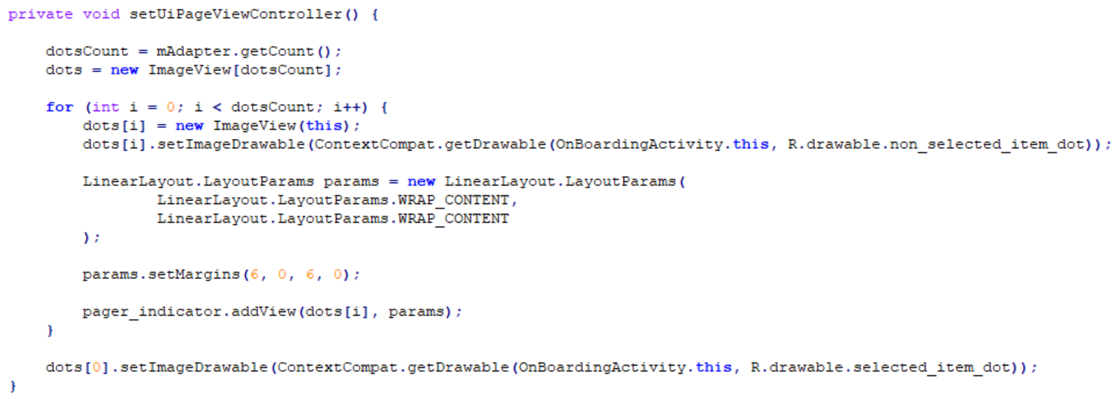
After Obfuscation:



```
public class HISPj7KHQ7 {
    public String HISPj7KHQ7;
    public String Wja3o2vx62;
    public String HISPj7KHQ7() {
        return this.HISPj7KHQ7;
    }
    public void HISPj7KHQ7(String str) {
        this.HISPj7KHQ7 = str;
    }
    public String Wja3o2vx62() {
        return this.Wja3o2vx62;
    }
    public void Wja3o2vx62(String str) {
        this.Wja3o2vx62 = str;
    }
}
```

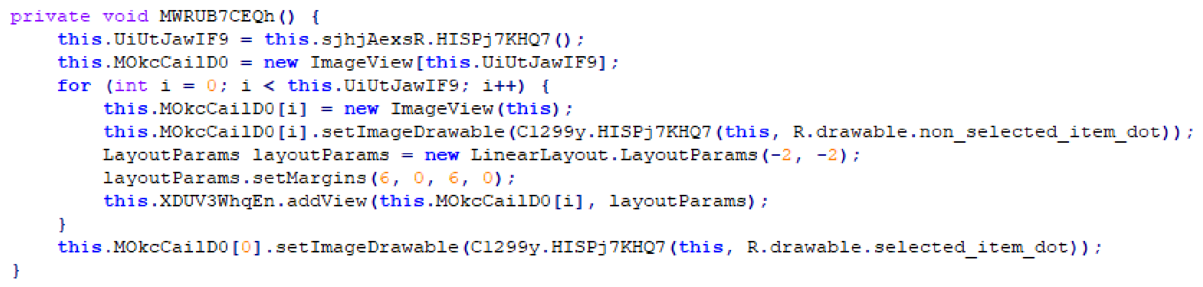
Project 3:

Before Obfuscation:



```
private void setUiPageViewController() {

    dotsCount = mAdapter.getCount();
    dots = new ImageView[dotsCount];

    for (int i = 0; i < dotsCount; i++) {
        dots[i] = new ImageView(this);
        dots[i].setImageDrawable(ContextCompat.getDrawable(OnBoardingActivity.this, R.drawable.non_selected_item_dot));

        LinearLayout.LayoutParams params = new LinearLayout.LayoutParams(
                LinearLayout.LayoutParams.WRAP_CONTENT,
                LinearLayout.LayoutParams.WRAP_CONTENT
        );

        params.setMargins(6, 0, 6, 0);

        pager_indicator.addView(dots[i], params);
    }

    dots[0].setImageDrawable(ContextCompat.getDrawable(OnBoardingActivity.this, R.drawable.selected_item_dot));
}
```

After Obfuscation:



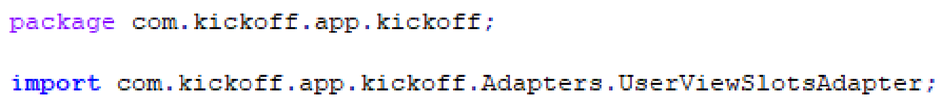
```
private void MWRUB7CEQh() {
    this.UiUtJawIF9 = this.sjhjAexsR.HISPj7KHQ7();
    this.MOkcCailD0 = new ImageView[this.UiUtJawIF9];
    for (int i = 0; i < this.UiUtJawIF9; i++) {
        this.MOkcCailD0[i] = new ImageView(this);
        this.MOkcCailD0[i].setImageDrawable(C1299y.HISPj7KHQ7(this, R.drawable.non_selected_item_dot));
        LayoutParams layoutParams = new LinearLayout.LayoutParams(-2, -2);
        layoutParams.setMargins(6, 0, 6, 0);
        this.XDUV3WhqEn.addView(this.MOkcCailD0[i], layoutParams);
    }
    this.MOkcCailD0[0].setImageDrawable(C1299y.HISPj7KHQ7(this, R.drawable.selected_item_dot));
}
```

Test Case 4:

Package obfuscation:

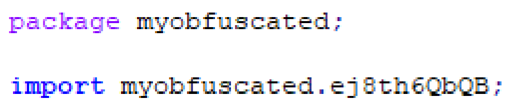
Project 1:

Before Obfuscation:



```
package com.kickoff.app.kickoff;

import com.kickoff.app.kickoff.Adapters.UserViewSlotsAdapter;
```

After Obfuscation:



```
package myobfuscated;

import myobfuscated.ej8th6QbQB;
```

All packages are obfuscated by repackaging them into one given package. Packaging can be further obfuscated by combining obfuscated classes into a single package. This specific type of renaming obfuscation is enabled by using following commands in proguard rules file:

Flattenpackagehierarchy – myobfuscated

Repackageclasses - myobfuscated

Exactly same package obfuscation technique has been used for other two test projects.

Project 2:

Before Obfuscation:

```
package com.weightscale.mar.weightscales;

import com.weightscale.mar.weightscales.Helper.DBHelper;
```

After Obfuscation:

```
package myobfuscated;

import myobfuscated.eyd3OXAZgV;
```

Project 3:

Before Obfuscation:

```
package com.pirsensor.app.pirsensor;

import com.pirsensor.app.pirsensor.LoginActivity;
```

After Obfuscation:

```
package myobfuscated;

import myobfuscated.eyd3OXAZgV;
```

Test Case 5:

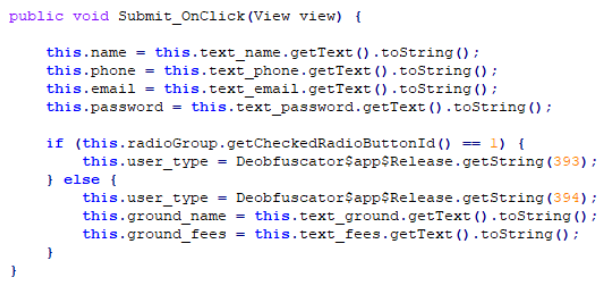
String obfuscation:

Project 1:

Before Obfuscation:

```
public void Submit_OnClick(View view) {

    name = text_name.getText().toString();
    phone = text_phone.getText().toString();
    email = text_email.getText().toString();
    password = text_password.getText().toString();

    int id = radioGroup.getCheckedRadioButtonId();

    if (id == 1) {
        user_type = "Player";
    } else {
        user_type = "Manager";
        ground_name = text_ground.getText().toString();
        ground_fees = text_fees.getText().toString();
    }
}
```

After Obfuscation:



```
public void Submit_OnClick(View view) {

    this.name = this.text_name.getText().toString();
    this.phone = this.text_phone.getText().toString();
    this.email = this.text_email.getText().toString();
    this.password = this.text_password.getText().toString();

    if (this.radioGroup.getCheckedRadioButtonId() == 1) {
        this.user_type = Deobfuscator$app$Release.getString(393);
    } else {
        this.user_type = Deobfuscator$app$Release.getString(394);
        this.ground_name = this.text_ground.getText().toString();
        this.ground_fees = this.text_fees.getText().toString();
    }
}
```

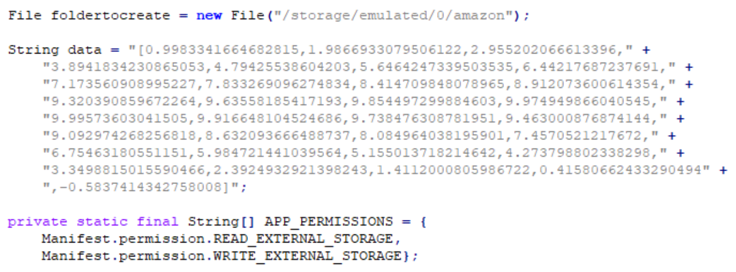
Hard-coded literals and strings cannot be obfuscated by ProGuard so to achieve string obfuscation we have used third party library i.e. Paranoid. Paranoid initializes an array of unique characters from all obfuscated strings and an array of indexes in the character array per each obfuscated string. When a project is compiled all hardcoded strings are swapped with function calls with a parameter which returns that specific string. The outcome is when the code is decompiled it is not possible to simply search for strings. This specific type of string obfuscation is enabled by using following line above every class name in each java file:

@Obfsucate

Exactly same string encryption/obfuscation technique has been used for other two test projects.

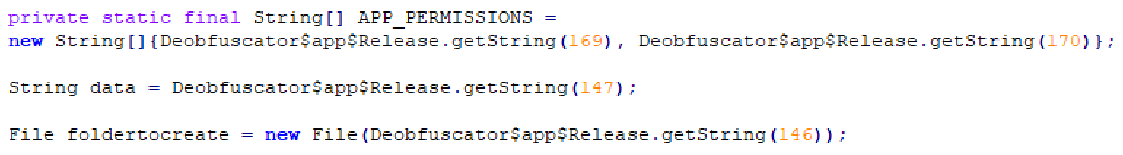
Project 2:

Before Obfuscation:



```
File foldertocreate = new File("/storage/emulated/0/amazon");

String data = "[0.9983341664682815,1.9866933079506122,2.955202066613396," +
    "3.8941834230865053,4.79425538604203,5.6464247339503535,6.44217687237691," +
    "7.173560908995227,7.833269096274834,8.414709848078965,8.912073600614354," +
    "9.320390859672264,9.63558185417193,9.854497299884603,9.974949866040545," +
    "9.9957360304150,9.916648104524686,9.738476308781951,9.463000876874144," +
    "9.092974268256818,8.632093666488737,8.084964038195901,7.457052121767," +
    "6.754631805511510,5.984721441039564,5.155013718214642,4.273798802338298," +
    "3.3498815015590466,2.3924932921398243,1.4112000805986722,0.41580662433290494" +
    ",-0.5837414342758008]";

private static final String[] APP_PERMISSIONS = {
    Manifest.permission.READ_EXTERNAL_STORAGE,
    Manifest.permission.WRITE_EXTERNAL_STORAGE};
```

After Obfuscation:



```
private static final String[] APP_PERMISSIONS =
new String[]{Deobfuscator$app$Release.getString(169), Deobfuscator$app$Release.getString(170)};

String data = Deobfuscator$app$Release.getString(147);

File foldertocreate = new File(Deobfuscator$app$Release.getString(146));
```

Project 3:

Before Obfuscation:

```
save_btn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

        if(audible_alarm_flag)
            dbHelper.update_setting_record("Audible_Alarm", audible_alarm);
        if(alert_vibrate_flag)
            dbHelper.update_setting_record("Alert_Vibrate", alert_vibrate);
        if(alert_sound_flag)
            dbHelper.update_setting_record("Alert_Sound", alert_sound);
        if(alert_volume_flag)
            dbHelper.update_setting_record("Alert_Volume", alert_volume);

        Toast.makeText(getApplicationContext(),"Settings Saved",Toast.LENGTH_SHORT).show();
    }
});
```

After Obfuscation:

```
public void onClick(View view) {
    if (this.HISPj7KHQ7.Z7Tb6ro9iU.booleanValue()) {
        this.HISPj7KHQ7.KelmFaNFJ.BsUTWEAMAI(jk.HISPj7KHQ7(324), this.HISPj7KHQ7.MOkcCailD0);
    }
    if (this.HISPj7KHQ7.c0IrT2pre0.booleanValue()) {
        this.HISPj7KHQ7.KelmFaNFJ.BsUTWEAMAI(jk.HISPj7KHQ7(325), this.HISPj7KHQ7.CQM3fxIO39);
    }
    if (this.HISPj7KHQ7.z1EN3svxlp.booleanValue()) {
        this.HISPj7KHQ7.KelmFaNFJ.BsUTWEAMAI(jk.HISPj7KHQ7(326), this.HISPj7KHQ7.sjhjAexsR);
    }
    if (this.HISPj7KHQ7.FPyqhFg65J.booleanValue()) {
        this.HISPj7KHQ7.KelmFaNFJ.BsUTWEAMAI(jk.HISPj7KHQ7(327), this.HISPj7KHQ7.YZnWBkArAW);
    }
    Toast.makeText(this.HISPj7KHQ7.getApplicationContext(), jk.HISPj7KHQ7(328), 0).show();
}
```

Test Case 6:

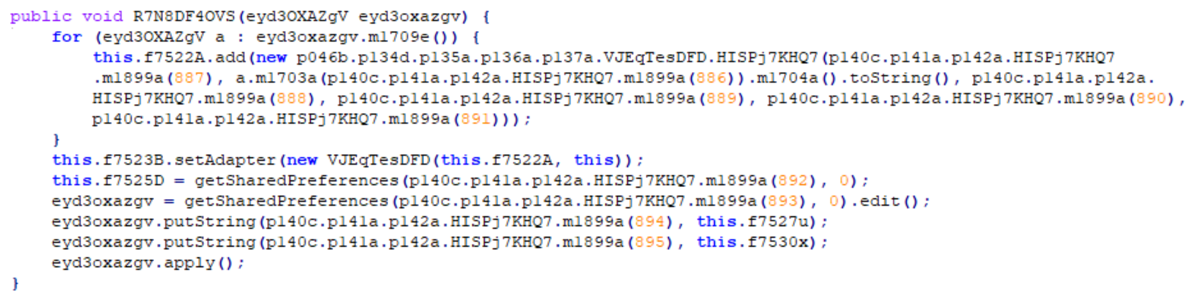
Code Optimization:

Project 1:

With Optimization:

```
public void R7N8DF4OVS(eyd3OXAZgV eyd3oxazgv) {
    for (eyd3OXAZgV a : eyd3oxazgv.m4470e()) {
        this.f7522A.add(new p046b.p065c.p066a.p067a.p068a.VJEqTesDFD.HISPj7KHQ7(gm.m5307a(912),
        a.m4464a(gm.m5307a(911)).m4465a().toString(), gm.m5307a(913), gm.m5307a(914), gm.m5307a(915), gm.m5307a(916)));
    }
    this.f7523B.setAdapter(new VJEqTesDFD(this.f7522A, this));
    this.f7525D = getSharedPreferences(gm.m5307a(917), 0);
    eyd3oxazgv = getSharedPreferences(gm.m5307a(918), 0).edit();
    eyd3oxazgv.putString(gm.m5307a(919), this.f7527u);
    eyd3oxazgv.putString(gm.m5307a(920), this.f7530x);
    eyd3oxazgv.apply();
}
```

Without Optimization:



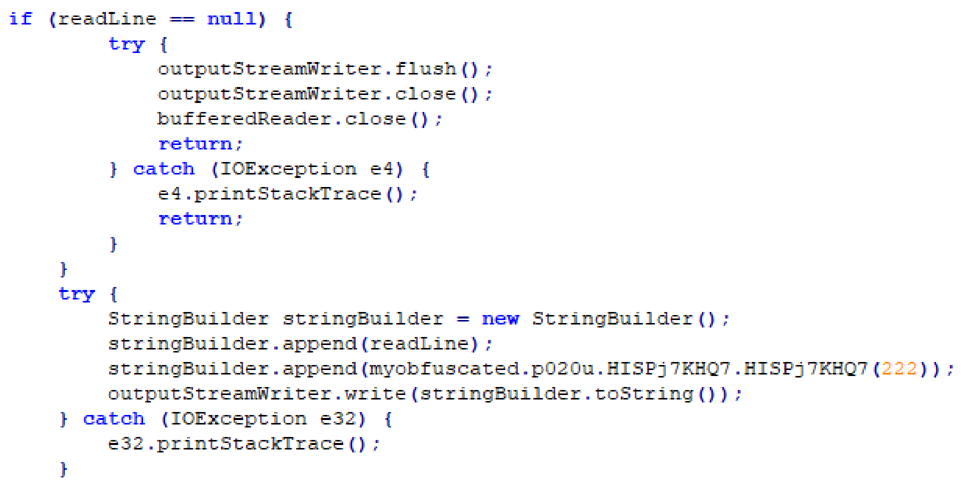
```
public void R7N8DF4OVS(eyd3OXAZgV eyd3oxazgv) {
    for (eyd3OXAZgV a : eyd3oxazgv.m1709e()) {
        this.f7522A.add(new p046b.p134d.p135a.p136a.p137a.VJEqTesDFD.HISPj7KHQ7(p140c.p141a.p142a.HISPj7KHQ7
        .m1899a(887), a.m1703a(p140c.p141a.p142a.HISPj7KHQ7.m1899a(886)).m1704a().toString(), p140c.p141a.p142a.
        HISPj7KHQ7.m1899a(888), p140c.p141a.p142a.HISPj7KHQ7.m1899a(889), p140c.p141a.p142a.HISPj7KHQ7.m1899a(890),
        p140c.p141a.p142a.HISPj7KHQ7.m1899a(891)));
    }
    this.f7523B.setAdapter(new VJEqTesDFD(this.f7522A, this));
    this.f7525D = getSharedPreferences(p140c.p141a.p142a.HISPj7KHQ7.m1899a(892), 0);
    eyd3oxazgv = getSharedPreferences(p140c.p141a.p142a.HISPj7KHQ7.m1899a(893), 0).edit();
    eyd3oxazgv.putString(p140c.p141a.p142a.HISPj7KHQ7.m1899a(894), this.f7527u);
    eyd3oxazgv.putString(p140c.p141a.p142a.HISPj7KHQ7.m1899a(895), this.f7530x);
    eyd3oxazgv.apply();
}
```

Code gets optimized intensively after going through numerous passes with filters applied to remove dead unused code and simplify loophole and infinite loops.

Exactly same optimization techniques have been used for other two test projects.

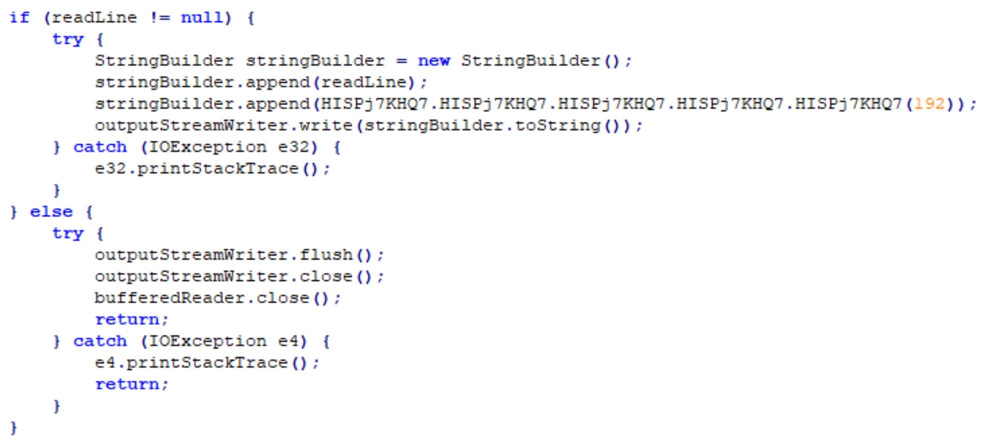
Project 2:

With Optimization:



```
if (readLine == null) {
        try {
            outputStreamWriter.flush();
            outputStreamWriter.close();
            bufferedReader.close();
            return;
        } catch (IOException e4) {
            e4.printStackTrace();
            return;
        }
    }
    try {
        StringBuilder stringBuilder = new StringBuilder();
        stringBuilder.append(readLine);
        stringBuilder.append(myobfuscated.p020u.HISPj7KHQ7.HISPj7KHQ7(222));
        outputStreamWriter.write(stringBuilder.toString());
    } catch (IOException e32) {
        e32.printStackTrace();
    }
```

Without Optimization:

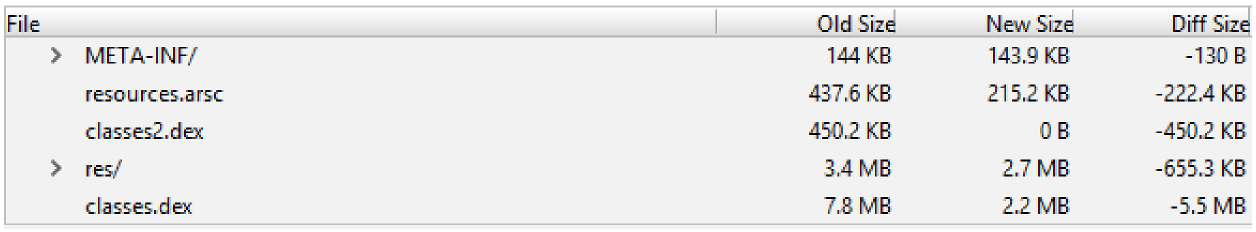


```
if (readLine != null) {
    try {
        StringBuilder stringBuilder = new StringBuilder();
        stringBuilder.append(readLine);
        stringBuilder.append(HISPj7KHQ7.HISPj7KHQ7.HISPj7KHQ7.HISPj7KHQ7.HISPj7KHQ7(192));
        outputStreamWriter.write(stringBuilder.toString());
    } catch (IOException e32) {
        e32.printStackTrace();
    }
} else {
    try {
        outputStreamWriter.flush();
        outputStreamWriter.close();
        bufferedReader.close();
        return;
    } catch (IOException e4) {
        e4.printStackTrace();
        return;
    }
}
```

Test Case 7:

APK Size Comparison:

Project 1:

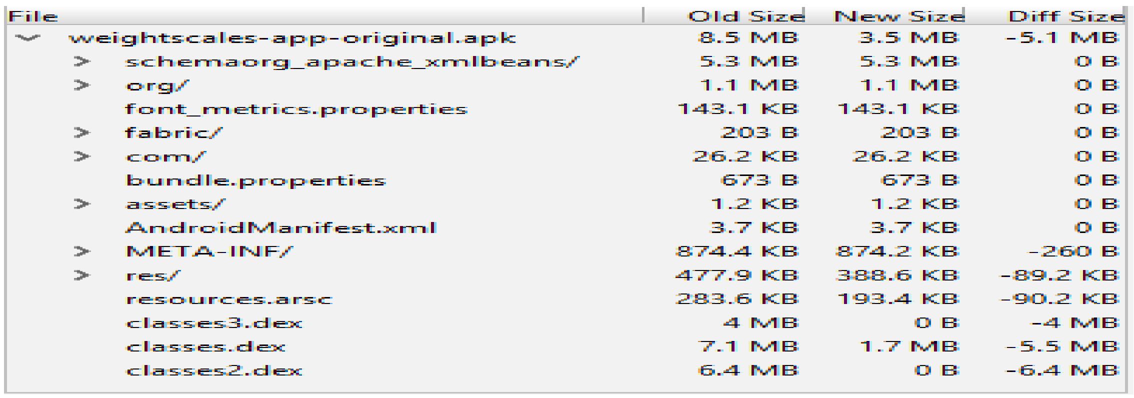


File	Old Size	New Size	Diff Size
> META-INF/	144 KB	143.9 KB	-130 B
resources.arsc	437.6 KB	215.2 KB	-222.4 KB
classes2.dex	450.2 KB	0 B	-450.2 KB
> res/	3.4 MB	2.7 MB	-655.3 KB
classes.dex	7.8 MB	2.2 MB	-5.5 MB

It can be clearly seen that application size has been significantly decreased by applying multiple code shrinking, optimization and obfuscation techniques altogether through different means i.e. ProGuard, Allatori and Paranoid.

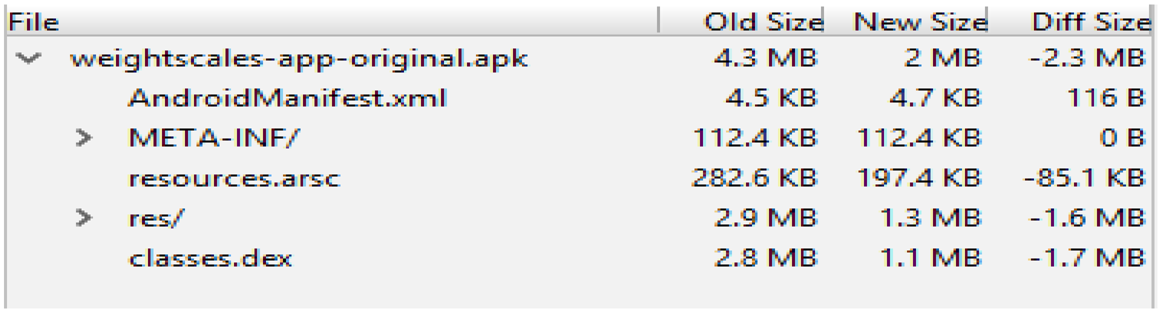
Exactly same techniques have been used for other two test projects.

Project 2:



File	Old Size	New Size	Diff Size
˅ weightscales-app-original.apk	8.5 MB	3.5 MB	-5.1 MB
> schemaorg_apache_xmlbeans/	5.3 MB	5.3 MB	0 B
> org/	1.1 MB	1.1 MB	0 B
font_metrics.properties	143.1 KB	143.1 KB	0 B
> fabric/	203 B	203 B	0 B
> com/	26.2 KB	26.2 KB	0 B
bundle.properties	673 B	673 B	0 B
> assets/	1.2 KB	1.2 KB	0 B
AndroidManifest.xml	3.7 KB	3.7 KB	0 B
> META-INF/	874.4 KB	874.2 KB	-260 B
> res/	477.9 KB	388.6 KB	-89.2 KB
resources.arsc	283.6 KB	193.4 KB	-90.2 KB
classes3.dex	4 MB	0 B	-4 MB
classes.dex	7.1 MB	1.7 MB	-5.5 MB
classes2.dex	6.4 MB	0 B	-6.4 MB

Project 3:



File	Old Size	New Size	Diff Size
weightscales-app-original.apk	4.3 MB	2 MB	-2.3 MB
AndroidManifest.xml	4.5 KB	4.7 KB	116 B
META-INF/	112.4 KB	112.4 KB	0 B
resources.arsc	282.6 KB	197.4 KB	-85.1 KB
res/	2.9 MB	1.3 MB	-1.6 MB
classes.dex	2.8 MB	1.1 MB	-1.7 MB

5.1.3. Functional Testing:

Functional testing of any program is aimed at testing whether functionality of the program remains same after obfuscation. Below result shows that there is no effect on functionality of tested features after obfuscation:

Sr. #	Test	Function Name	Function Conformity after obfuscation
1	Project 1	Login	No change
2	Project 2	Registration	No change
3	Project 3	Settings	No change

Table5.2: Sample Test Projects with Function conformity after obfuscation

5.1.4. Limitation:

We performed code obfuscation on android application by using hybrid approach of pro-guard, paranoid and allatori. After complete implementation and experiments we are successful to obfuscate all java source files. But resource files <xml> are left to be obfuscated due to the limitation of scripting language, as we performed obfuscation on managed code.

5.1.5. Comparisons:

Below section elaborates the libraries and their feature set, availability, platform and its evaluation.

Obfuscator	**Platform Supported**	**Price**	**License**	**Evaluated**	**Feature Set**
Arxan	Web, Android, iOS	Trial Version	Free	No	White box cryptography
Cloakware	Web, Android, iOS	Not available	Not available	No	Data, function and control flow transformations, anti-debug, white box cryptography
Zelix Klassmaster	Java	$ 479	Commercial	No	Flow, String and Reference Obfuscation
JBCO	Java	Free	Open-Source	No	Adding dead code, combining try with catch blocks, reordering conditional statements
JMOT	Java	Free	Open-Source	No	Name and string obfuscation
Sandmark	Java	Free	Open-Source	No	Watermarking, tamper-proofing and code obfuscation, code optimizing
Jfuscator	Java	Trial Version	Free	No	Flow, API call, string, debug info obfuscation
JOAD	Java	Free	Open-Source	No	Name, control-flow and string obfuscation, code optimization
JShield	Java	Not available	Not available	No	Layout, control and data obfuscation
Smokescreen	Java	Not available	Not available	No	Name, control-flow, string obfuscation, removal of unused resources
CodeShield	Java	Not available	Not available	No	Name and control-flow obfuscation, rearranging statements
VOT4CS	C#			Yes	

ProGuard	Java and Android	Free	Open-Source	Yes	Name, debug info, incremental obfuscation and optimization and shrinking of unused resources
Paranoid	Android	Free	Open-Source	Yes	String obfuscation
Allatori	Java and Android	Trial Version	Free	Yes	Name, flow, debug info, string, incremental obfuscation
DashO	Java and Android	$ 895	Commercial	No	Name obfuscation and removal of unused class, method and field.
Hybrid	**Android**	**Free**	**Proposed Approach**	YES	**Name, debug info, incremental obfuscation, optimization, shrinking of unused resources, String Obfuscation, Flow Obfuscation, prevention of java file after de-obfuscation, dead code insertion,**

Table 5.3: Comparison of Libraries

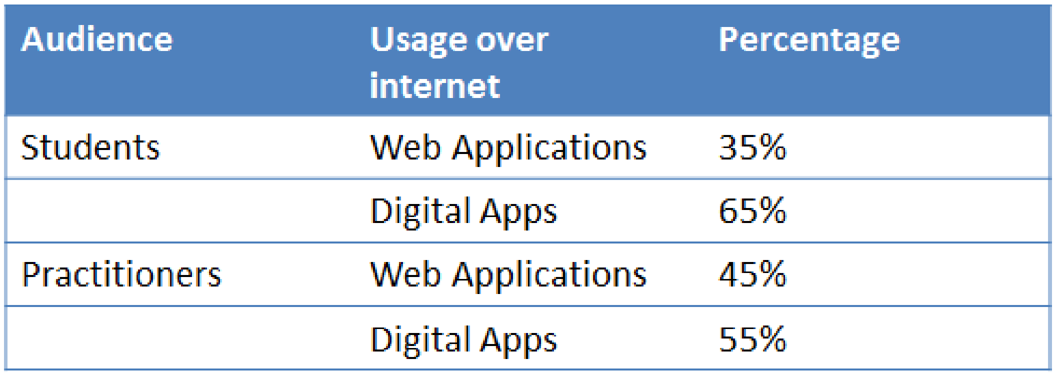
CHAPTER 6

CONCLUSION AND FUTURE WORK

Chapter # 6 Conclusion and Future Work

6.1. Conclusion:

With the paradigm shift of digital era, software applications are light weight. It is more demanding to use in-hand applications. This trend sets need of emerging mobile applications; so when application is installed on end-user device then it is mandatory to secure it. If application is not secure, by using reverse engineering process competitor recognize the behavior of application and re-use the algorithm where traditional approaches are not enough to secure applications. Process is initiated by collecting feedback from students and practitioner's and results are below



Audience	Usage over internet	Percentage
Students	Web Applications	35%
	Digital Apps	65%
Practitioners	Web Applications	45%
	Digital Apps	55%

Figure 6.1: Feedback

The main contribution to this research was securing android application by comparing tools and techniques for code obfuscation precisely. Many experiments are performed by using tools like VOT4CS in .net and allatori, paranoid and pro-guard in android to provide all in one package. Moreover to provide open source package by using hybrid approaches to protect the software application against duplication of algorithms. The main purpose of this research was preventing reverse engineering attacks and raising the bars against attackers. Our research is divided into extensive literature with practitioner feedbacks, using hybrid approaches for obfuscation and performing experiments on selected algorithms. The purpose of code obfuscation was to shield against malicious reverse engineering attacks. We presented hybrid methodology to mitigate these attacks based on experimentations. Obviously with all efforts, we cannot claim to obfuscate the application 100% but we can attain maximum results to make it harder for attacker to trace it back. With the practices of reverse engineering, attacker would need cost and time with useless efforts to get back the algorithms of application. So by concluding the research work, we can claim (i) we provided one open source hybrid obfuscated method, (ii) we improved the efficiency of four taxonomies of Collberg's i.e. resilience, cost, stealth and potency whereas stealth as low impact on proposed algorithm. But cost has negative whereas resilience and potency has negative impact (iii) We evaluated code optimization and shrinking apk size. We obfuscated android application, but we left resource file obfuscation <xml>, because our main focus was on managed code obfuscation. As resource file is scripting language. Below is given result.

Sr. #	Test	$S_{quality}$ before obfuscation	$S_{quality}$ after obfuscation	Overall $S_{quality}$
1	Project 1	0	0.4 (0.75) + 0.6 (0.6) – (–0.08) = 0.3 + 0.36 + 0.08	0.74
2	Project 2	0	0.4 (0.68) + 0.6 (0.4) – (–0.05) = 0.27 + 0.24 + 0.05	0.56
3	Project 3	0	0.4 (0.65) + 0.6 (0.5) – (–0.01) = 0.26 + 0.3 + 0.01	0.57

Table 6.1: Quality of Project

6.2. Future Work:

We successfully obfuscated all java files, and also performed de-obfuscation; in the result we were not able to get back the original code. Where-as, resource files <xml> are left to be obfuscate because they fall in scripting code that does not lie in the boundary of obfuscation for managed code.

References

[1]. B. Anckaert , M. Jakubowski, “Proteus: Virtualization for Diversified Tamper-Resistance” , ACM , 2006

[2].

[3]. P. Sivadasan, P.Sojan Lal, “Suggesting Potency Measures for Obfuscated Arrays and Usage of Source Code Obfuscators for Intellectual Property Protection of Java Products”, IEEE, 2011

[4]. M. Rostami, F. Koushanfar, “Hardware Security: Threat Models and Metrics”, IEEE, 2013

[5]. S. Khummanee, A. Khumseela, S. Puangpronpitag, “Towards a New Design of Firewall: Anomaly Elimination and Fast Verifying of Firewall Rules”, IEEE, 2013

[6]. Alex X. Liu, Mohamed G. Gouda, “Firewall Policy Queries”, IEEE, 2009

[7]. A. Herzberg, SS. Pinte, “Public Protection of Software”, ACM, 1987

[8]. U.G. Wilhelm, “Cryptographically protected Objects”, 1999

[9]. C. Collberg, C. Thomborson, D. Low, “Breaking Abstraction and Unstructuring Data Structure”, IEEE, 2009

[10]. B. Pingle, A. Mairaj, A. Y. Javaid, “Real-world Man-in-the-middle (MITM) Attack Implementation Using Open Source Tools for Instructional Use”, IEEE, 2018

[11]. M. Ceccato, M. Penta, P, Falcarin, F. Ricca, M. Torchiano, P. Tonella, “A family of experiments to assess the effectiveness and efficiency of source code obfuscation techniques”, Springer, 2014

[12]. M. Franz, “E unibus pluram: Massive-Scale Software Diversity as a Defense Mechanism”, ACM, 2010

[13]. S. Banescu, C, Lucaci, B. Krämer, A. Pretschner, “VOT4CS: A Virtualization Obfuscation Tool for C#”, ACM, 2016

[14]. P. SamUetson, “Reverse-Engineering Someone Else’s Software: Is It Legal”, IEEE, 1990

[15]. K. Gallagher, J. Deignan, “The Law and Reverse Engineering”, IEEE, 2012

[16]. S. Sebastian, S. Malgaonkar, P. Shah, M. Kapoor, T. Parekhji, “A Study & Review on Code Obfuscation”, IEEE, 2016

[17]. C. Collberg, C. Thomborson, “Watermarking, Tamper-Proofing, and Obfuscation Tools for Software Protection”, IEEE, 2002

[18]. A. Viticchi, L. Regano, M. Torchiano, C. Basile, M. Ceccato, P. Tonella, R. Tiella, “Assessment of Source Code Obfuscation Techniques”, IEEE, 2016

[19]. A. Kuma, S. Sundar, S. Kumar, “A Code Obfuscation Technique to Prevent Reverse Engineering”, IEEE, 2017

[20]. A. Sheneamer, S. Roy, J. Kalita, “A detection framework for semantic code clones and obfuscated code”, Elsevier, 2017

[21]. J. Garcia, M. Hammad, S. Malek, “Lightweight, Obfuscation-Resilient Detection and Family Identification of Android Malware”, ACM, 2018

[22]. C. Collberg, C. Thomborson, D. Low, “Manufacturing Cheap, Resilient and Stealthy Opaque Constructs”, ACM, 1998

[23]. S. Schrittwieser, S, Katzenbeisse, “Code Obfuscation against Static and Dynamic Reverse Engineering”, Springer, 2011

[24]. W. Xingkui, P. Xinguang, “Research on Data Leak Protection Technology Based on TPM”, IEEE, 2013

[25]. A. Ramakic, Z. Bundalo, “DATA PROTECTION IN MICROCOMPUTER SYSTEMS AND NETWORKS”, 2014

[26]. M. Markovi, “Data Protection Techniques, Cryptographic Protocols and PKI Systems in Modern Computer Networks”, IEEE, 2007

[27]. Z. Bundalo, A. Ramakic, D. Bundalo, “Increasing Desktop Application and User Data Protection Using Smartphone”, IEEE, 2016

[28]. W. Stallings, “Cryptography and Network Security” Fourth Edition, pg 234-345

[29]. A. Berghe1, R. Scandariato, K.Yskout, W, Joosen, “Design notations for secure software: a systematic literature review”, Springer, 2015

[30]. E. Bertino, G. Ghinita, A. Kamra, “Access Control for Databases: Concepts and Systems”, 2011

[31]. W. Sujansky, S. Faus, E. Stone, P. Brennan, “A method to implement fine-grained access control for personal health records through standards relational database queries”, Elsevier, 2010

[32]. J. Berrington, Databases, Elsevier, 2007

[33]. L. Fu Lu, J. Zhang, M. Huang, L. Fu, “String alignment pre-detection using unique subsequences for FPGA-based network intrusion detection”, Elsevier, 2010

[34]. J. Serrano, J. Palancar, “String alignment pre-detection using unique subsequences for FPGA-based network intrusion detection”, Elsevier, 2012

[35]. A. Sallam, E. Rabaie, O. Faragallah, “Encryption-based multilevel model for DBMS”, Elsevier, 2012

[36]. T. Toland, C. Farkas, C. Eastman, “The inference problem: maintaining maximal availability in the presence of database updates”, Elsevier, 2010

[37]. X. Feng, Q. Zeng, “Application of Firewall Technology and Research”, 2016

[38]. G. Bhatnagar, Q. Jonathan, B. Raman, “Biometric Template Security based on Watermarking”, Elsevier, 2010

[39]. Xu He, W. Suo-ping, W. Ru-chuan, W. Zhong-qin, “Efficient P2P-based mutual authentication protocol for RFID system security of EPC network using asymmetric encryption algorithm”, Elsevier, 2011

[40]. L. Zhang, Q. Wu, Bo Qin, J. Domingo, U. Nicolas, “Asymmetric group key agreement protocol for open networks and its application to broadcast encryption”, Elsevier, 2011

[41]. R. Madhusudhan, R. Mittal, “Dynamic ID-based remote user password authentication schemes using smart cards: A review”, Elsevier, 2012

[42]. S. Khan, R. Gupta, “Future Aspect of Firewall in Internet Security”, 2014

[43]. P. Bera, S. Ghosh, P. Dasgupta, “Formal Verification of Security Policy Implementations in Enterprise Networks”, Springer, 2009

[44]. B. Yadegari, B. Johannesmeyer, B.Whitely, S. Debray, “A Generic Approach to Automatic Deobfuscation of Executable Code”, IEEE, 2015

[45]. E. Shaer, H. Hamed, “Management and Translation of Filtering Security Policies”, IEEE, 2003

[46]. S. Banescu, M. Ochoa, A. Pretschner, “A Framework for Measuring Software Obfuscation Resilience against Automated Attacks”, ACM, 2015

[47]. M. Ulum, “Cyber Security - Literature Review”, 2017

[48]. J. Cazalas, J. McDonald, T. Andel, N. Stakhanova, “Probing the Limits of Virtualized Software Protection”, ACM, 2014

[49]. T. Toland, C. Farkas, C. Eastman, “Taxonomy of obfuscation transformation”, ACM, 1998

[50]. K. Coogan, G. Lu, S. Debray, “De obfuscation of Virtualization-Obfuscated Software”, ACM, 2011

[51]. S. Forrest, A. Somayaji, D. Ackley, “Building Diverse Computer Systems”, IEEE, 1997

[52]. J. Kinder, “Towards Static Analysis of Virtualization-Obfuscated Binaries”, IEEE, 2012

[53]. T. Laszlo, “Obfuscating C++ Programs via Control Flow Flattening”, 2007

[54]. C. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, “Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation”, ACM, 2005

[55]. R. Rolles, “Unpacking Virtualization Obfuscator”, ACM, 2009

[56]. M. Sharif, A. Lanzi, J. Giffin, W. Lee, “Automatic Reverse Engineering of Malware Emulators”, IEEE, 2009

[57]. J. Xu, Z. Kalbarczyk, R. Lyer, “Transparent runtime randomization for security”, IEEE, 2003

[58]. X. Jiang, H. Wangz, D. Xu, Y. Wang, “Thwarting code injection attacks with system service interface”, IEEE, 2007

[59]. S. Drape, A. Majumdar, C. Thomborson, “Slicing aided design of obfuscating transforms”, IEEE, 2007

[60]. R. Chakraborty, S. Narasimhan, S. Bhunia, “Embedded Software Security through Key-Based Control Flow Obfuscation”, Springer, 2011