Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 4, Issue. 4, April 2015, pg.347 – 352

RESEARCH ARTICLE

Securing Android Code Using White Box Cryptography and Obfuscation Techniques

Ashish Anand
University School of Information and Communication Technology
Guru Govind Singh Indraprastha University, NEW DELHI
ashishwrk@gmail.com

ABSTRACT: Code obfuscation is a set of program transformations that make program code and program execution difficult to analyze. First of all, obfuscation hinders manual inspection of program internals. By renaming variables and functions, and breaking down structures, it protects against reverse-engineering. It protects both storage and usage of keys, and it can hide certain properties such as a software fingerprint or a watermark, or even the location of a flaw in case of an obfuscated patch. However, code obfuscation itself does not protect from code lifting or software piracy. It merely strengthens built-in protection mechanisms, e.g. against tampering or piracy so we propose a strong method for code obfuscation with white box cryptography as White box cryptographic algorithms aim to denying the key readout even if the source code embedding the key is disclosed Combination of these two concepts gives a new level in modern cryptography as well as optimizes its performance and additionally we will make Client and server more secure. Obfuscation itself does not prevent tampering, but hinders the preceding analysis phase. In our proposed method we will use code obfuscation with white box cryptography and apply on android based application, for this purpose we will use proguard.
Keywords: Software quality, Improvement, Testing, reliability.

I. INTRODUCTION

Android is an open source platform of Google that has become one of the most popular operating system. As a result, protecting applications running on Android becomes of interest. Currently, Reverse engineering of Android applications is much easier than on other architectures, due to the high level but simple byte code language used. Hence, the goal is to minimize and manage risks of software flaws. Anti-Reverse engineering techniques can be used to prevent reverse engineering. Obfuscation [1] is the paradigm of hiding program semantics through choosing semantically equivalent but complex and ambiguous representations in order to aggravate analysis. In order to achieve this protection, obfuscation transforms program code of an application in a way that it is "hard to reverse engineer" but without changing the behavior of this application. Hard to reverse engineer means that automated programs cannot produce good abstractions of the analyzed program and the results of the analysis become harder to understand for a human analyst. The initial objective of cryptography has been to design algorithms and protocols to protect a communication channel next to eavesdropping. In black box cryptography the attacker only has access to the input/output of the algorithm. Today we even encounter an even worse attack model, where cryptography is deployed in applications that are executed on open devices (such as Personal Computers or on a tablet, Smartphone

without exploiting secure elements [6]. White box cryptography is the new technique against attacks on white box attack environments. In white box attack model, the attacker is even stronger than in black box attack model, and the attacker can monitor all intermediate values [7]. Therefore, safety algorithms are needed against all operation steps being exposure. In white box attack, an attacker has full access to the software implementation of a cryptographic algorithm where the binary is completely visible and alterable by the attacker and the attacker has full control over the execution platform. A good obfuscation is composed of one or more code transformations that transform a program's control and data. While code obfuscation techniques do not guarantee perfect security, a combination of several transformation techniques can lead to sufficient practical protection against reverse-engineering and tampering attack flow such that it becomes harder to reverse engineer. The only restriction for these transformations is preserving the functionality of the original program.
Thus, obfuscation is a collection of many techniques that are useful for program transformation, obfuscation or randomization. While code obfuscation techniques do not guarantee perfect security, a combination of several transformation techniques can lead to sufficient practical protection against reverse-engineering and tampering attack for this purpose we use white box cryptography.

II. RELATED WORK

Obfuscation techniques are used in the Android platform to protect applications against reverse engineering [1]. In order to achieve this protection, the obfuscation methods transform the program code without changing its behavior. ProGuard [2] is applied to obfuscate program code and protect the Android application. In this paper, we study the obfuscation techniques used in malware context to evade detection of private data leakage in the android system. Security by obscurity attempts to establish security by keeping the design or the implementation to provide it secret [4]. This concept is usually disdained by security experts. Collberg [3] confirms this statement: "As far as we know, there do not exist any techniques for preventing attacks by reverse engineering stronger than what is afforded by obscuring the purpose of the code". Researchers have proposed various anti-reversing techniques to prevent reverse engineering. Currently there are so many techniques available but none of them provides 100% protection against reverse engineering. "Code Protection in Android by Patrick Schulz"[8] discusses the possible code obfuscation methods on the Android platform using Identifier mangling ,String obfuscation , Dynamic code loading , dead code, Self modifying code. Anti-reversing techniques [5] are defence techniques implemented in software in order to protect it from malicious attacks. It has become a challenge for the software community to protect software from attackers and to prevent its misuse. The patent system is not quite as effective with software as it is with traditionally engineered tangible artifacts. While a patent mandates Intellectual Property (IP) protection – it is next to impossible to prove or even suspect any IP theft in a software product that might have been the result of a malicious reverse engineering attack on a patented competitor. A lot of research is being done in the software field in order to find out successful ways of protecting software from reverse engineering attacks. The techniques proposed to make reverse engineering difficult include obfuscating the code protecting the computing platform physically, encryption of executables, and watermarking. The term "white-box cryptography" describes a secure implementation of cryptographic algorithms in an execution environment, such as on a desktop computer or a mobile device that is fully observable and modifiable by an attacker [9]. It is different from black-box cryptography, where an algorithm's internal processing data is unavailable to an attacker. The white-box environment puts hard additional restrictions on implementations of the cryptographic algorithms [10]. In traditional cryptography, a black-box attack describes the situation where the attacker tries to obtain the cryptographic key by knowing the algorithm and monitoring the inputs and outputs, but without the execution being visible [11]. White box cryptography addresses the much more severe threat model where the attacker can observe everything, can access all aspects of the target system application, and may have the black-box knowledge of the crypto algorithm. White-box cryptography is aimed at protecting secret keys from being disclosed in a software implementation [12].

III. CODE OBFUSCATION

Code obfuscation [13] is a set of program transformations that make program code and program execution difficult to analyze. First of all, obfuscation hinders manual inspection of program internals. By renaming variables and functions, and breaking down structures, it protects against reverse-engineering. However, code obfuscation itself does not protect from code lifting or software piracy. Obfuscation could be used to hide vulnerabilities as well. However, *security through obscurity* is often not considered a good practice. Kerckhoffs' principle [14] states that a system should be secure, even if everything is known about the system, except the key. In the case of obfuscation, the 'key' can specify which transformations were performed, in what order, and on which section of the code. This key allows the software owner to reconstruct in a deterministic way a user's obfuscated, or even personalized, version of the binary. These keys can be kept in a database until required for maintenance or analysis of bug reports.

Reverse Engineering: The processes of compilation and reverse engineering are illustrated in Figure 1. Compilation refers to the translation of a source-language program to machine code; it consists of a series of steps, each producing successively lower-level program representations.

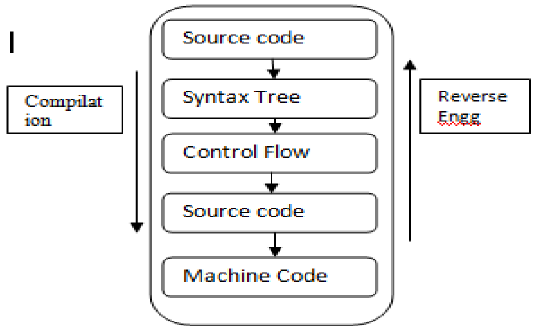


Fig1: Reverse Engineering process

Reverse engineering is the dual process of recovering higher-level structure and semantics from a machine code program. Most of the prior work on code obfuscation and tamper-proofing focus on various aspects of decompilation.

ProGuard: ProGuard is a free Java class file shrinker, optimizer, obfuscator, and preverifier. It detects and removes unused classes, fields, methods, and attributes. It optimizes bytecode and removes unused instructions. It renames the remaining classes, fields, and methods using short meaningless names. Finally, it preverifies the processed code for Java 6 or higher, or for Java Micro Edition.

Some uses of ProGuard are:

- Creating more compact code, for smaller code archives, faster transfer across networks, faster loading, and smaller memory footprints.
- Making programs and libraries harder to reverse-engineer.
- Listing dead code, so it can be removed from the source code.
- Retargeting and preverifying existing class files for Java 6 or higher, to take full advantage of their faster class loading.

IV. WHITE BOX CRYPTOGRAPHY

A major issue when dealing with security programs is the protection of sensitive data embedded in the code. The usual solution consists in encrypting the data but the legitimate user needs to get access to the decryption key, which also needs to be protected. This is even more challenging in a software-*only* solution, running on a non-trusted host.
White-box cryptography is aimed at protecting secret keys from being disclosed in a *software* implementation. In such a context, it is assumed that the attacker (usually a \legitimate" user or malicious software) may also control the execution environment.

V. PROPOSED SYSTEM ARCHITECTURE

According to previous studies, it was obvious that using a technique to protect java's source file rather than class file is most effective, even though it has some problems, but these problems can be solved as development grows further.
Proposed technique is still under process. It is based on several steps; first step, we use white box encryption for encrypt text or source which will help to transform the look and form of the code which will confuse the reader. Second step use obfuscation technique for make this code more difficult to reverse engineering by using mathematical formula to convert messages into unreadable form in which can be read while printing. The mathematical formula helps transforming the code while reversing. Third Step, is to convert variables into binary code to confuse the reader. These steps will give the revere very hard and long time to understand the obfuscated code. Fig. 2 shows flow chart of proposed method.

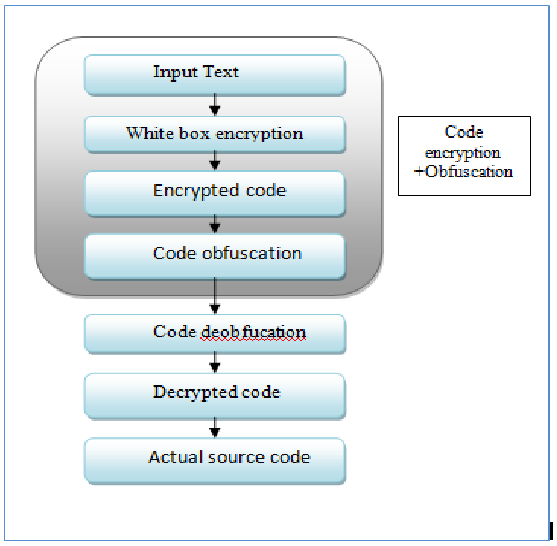
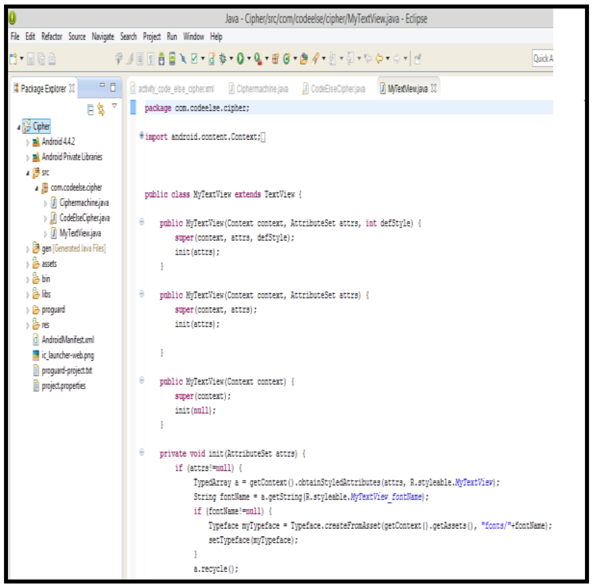


Fig2: Proposed system flow

In the above flowchart first of all we give input source code. After that, the code will be compiled into class file which contains garbage code, then a reversing tool will be used to test how much can be uncovered from the code, then will inject the new algorithm into the original code, and test again by using the same reversing tool to find the difference with and without using the algorithm. The environment used for this implementation is Eclipse. Fig. 3 shows the original code before injecting the algorithm and before reversing.

A. Sample of Code before Using the Algorithm

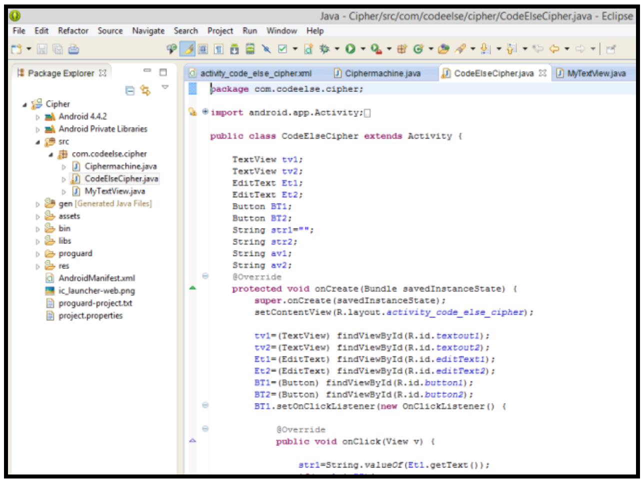

Java - Cipher/src/com/codeelse/cipher/MyTextView.java - Eclipse
File Edit Refactor Source Navigate Search Project Run Window Help
Package Explorer
activity_code_else_cipher.xml
Ciphermachine.java
CodeElseCipher.java
MyTextView.java

```
package com.codeelse.cipher;

import android.content.Context;



public class MyTextView extends TextView {

    public MyTextView(Context context, AttributeSet attrs, int defStyle) {
        super(context, attrs, defStyle);
        init(attrs);
    }

    public MyTextView(Context context, AttributeSet attrs) {
        super(context, attrs);
        init(attrs);

    }

    public MyTextView(Context context) {
        super(context);
        init(null);
    }

    private void init(AttributeSet attrs) {
        if (attrs!=null) {
             TypedArray a = getContext().obtainStyledAttributes(attrs, R.styleable.MyTextView);
             String fontName = a.getString(R.styleable.MyTextView_fontName);
             if (fontName!=null) {
                 Typeface myTypeface = Typeface.createFromAsset(getContext().getAssets(), "fonts/"+fontName);
                 setTypeface(myTypeface);
             }
             a.recycle();
```

Fig3:actual code for input

This is actual source code first this code has been encrypted with the help of white box cryptography as it is more secure than other cryptographic algorithm

B.OBFUSCATION PROCESS


Java - Cipher/src/com/codeelse/cipher/CodeElseCipher.java - Eclipse
File Edit Refactor Source Navigate Search Project Run Window Help
Package Explorer
activity_code_else_cipher.xml
Ciphermachine.java
CodeElseCipher.java
MyTextView.java

```
package com.codeelse.cipher;

import android.app.Activity;

public class CodeElseCipher extends Activity {

    TextView tv1;
    TextView tv2;
    EditText Et1;
    EditText Et2;
    Button BT1;
    Button BT2;
    String str1="";
    String str2;
    String av1;
    String av2;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_code_else_cipher);

        tv1=(TextView) findViewById(R.id.textout1);
        tv2=(TextView) findViewById(R.id.textout2);
        Et1=(EditText) findViewById(R.id.editText1);
        Et2=(EditText) findViewById(R.id.editText2);
        BT1=(Button) findViewById(R.id.button1);
        BT2=(Button) findViewById(R.id.button2);
        BT1.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {

                str1=String.valueOf(Et1.getText());
```

Fig3: Obfuscation code

When this program is compiled, a class file will be generated, it contains the garbage code which is unreadable. The displayed garbage code is a mixture English words, symbols and numbers. Fig. 3 shows the code after compiling. To demonstrate theory of the ability to break java’s class file, a reversing tool is used to break the application after compiling, this tool is ProGuard. The class file will be dragged into ProGuard platform, then, the original code will be generated. Fig. 4 shows the code for processing in ProGuard after reversing.

VI. RESULT

The code after obfuscation has changed in form, variables names have changed as well. But both show the same result for output. It's very hard to read and difficult to reverse to actual code .but for comparison it had included extra class for the application. Number of lines methods and libraries has changed as well.

VII. CONCLUSION

Implementing a defensive technique to protect the code might not be very useful, although it can protect the code up to some level. The value of the protection level will fluctuate to how strong or weak the technique is. It is very important to admit that all applications need to be protected specially the ones with special business rules and secret ideas, reverser usually attacks these applications. Current state indications that all reveres and companies are more interested to break complicated expert systems rather than implementing fresh ones, to save the writing time. Implementing such technique will make them to struggle to understand the obfuscated code and it will require more time to find out the lifecycle of the implemented logic as well.

REFERENCES

[1] Patrick Schulz, "Code protection in android," 2012.
[2] Eric Lafortune et al., "Proguard," 2006
[3] Christian S. Collberg and Clark Thomborson. Watermarking, tamper-proofing, and obfuscation-tools for software protection. Software Engineering, IEEE Transactions on, 28(8):735–746, 2002.
[4] By Simson Garfinkel, Gene Spafford, and Second Edition. Table of Contents Part IV: Network and Internet Security. Number April. 1996.
[5] Kundu, Deepti, "JShield: A Java Anti-Reversing Tool" (2011).Master's Projects. Paper 161.
[6] Brecht Wyseur Brecht, "White-Box Cryptography: Hiding Keys In Software", NAGRA Kudelski Group, Switzerland, 2009
[7] S. Chow, P. Eisen, H. Johnso1, P.C. van Oorschot,2002 ," A White-Box DES Implementation for DRM Applications", Cloakware Corporation, Ottawa, Canad , Carleton University, Ottawa, Canada
[8] Patrick Schulz "Code Protection in Android " 2012-Schulz-Code_Protection_in_Android.pdf
[9] S. Chow, P. Eisen, H. Johnson, P.C. van Oorschot , 16 August 2002,"White-Box Cryptography and an AES Implementation", (SAC'02), Ottawa, Canada.
[10] Olivier Billet, Henri Gilbert,"Cryptanalysis of a White Box AES Implementation", Charaf Ech-Chatbi France T´el´ecom R&D, France, 2009
[11] Wil Michiels, "Mechanism for Software Tamper Resistance: An Application of White-Box Cryptography",Philips Research Laboratories Eindhoven, The Netherlands Department of Mathematics and Computer Science, DRM'07, Alexandria, Virginia, USA,Copyright 2007 ACM, October 29, 2007
[12] Marc Joye,2008, "On White-Box Cryptography" ,Thomson R&D France, Technology Group, Corporate Research, Security Laboratory 1 avenue de Belle Fontaine, France Security of Information and Networks,Tra®ord Publishing.
[13] C. S. Collberg and C. D. Thomborson. Watermarking, tamper-proofing, and obfuscation - tools for software protection. volume 28, pages 735–746, 2002.
[14] A. Kerckhoffs. La cryptography militaire. *Journal des sciences militaires*, IX:5–38, Janvier 1883.http://www.petitcolas.net/fabien/kerckhoffs/ (consulted on February 10th, 2012).