Reverse engineering of mobile applications

Kirill Zhigalov\textsuperscript{1,2} and Viacheslav Ivanov\textsuperscript{3}

\textsuperscript{1} V A Trapeznikov Institute of Control Sciences of Russian Academy of Sciences, Moscow, Russia
\textsuperscript{2} Moscow Technological Institute, Moscow, Russia.
\textsuperscript{3} Moscow University of the Ministry of Internal Affairs of the Russian Federation n.a. V Ya Kikotya, Moscow, Russia

E-mail: kshakalov@mail.ru

Abstract. Features of obtaining the source code of the application written under Android OS are considered. Most reverse engineering methods focus on automatically extracting information from the source code without taking external human knowledge into account. Most often, however, important external information is only available (for example, developer knowledge or domain knowledge) when human knowledge is used. Much of the knowledge about software systems is implicit and therefore difficult to recover with purely automated methods. There are types of information that are difficult to detect only in the source code, and this problem is solved by additional human knowledge. Typical approaches to the justification of data include coding an explicit meta-model and expressing analyses at that level.

1. Introduction
Reverse engineering is nothing more than the process of extracting knowledge or useful information from any product, regardless of whether it is applied in the restoration of a Bicycle or a car, or any product manufactured by a person. You would come to learn a lot of new and interesting things after you broke the actual product into different parts and reassembled them.

Sometimes, it is very useful to reassemble computer program to find out if it is some spy or virus parts in software. As you can see examples in part 4 of our article some such parts a very dangerous for security of your data or money.

But reverse engineering in software is more difficult, than in hardware and it is needed a lot of code examples database for analyses. For simplifying the analysis of the program during reverse engineering process it is advisable to use any existent databases, as an example – Viper.

Under such a short name (Viper) hides a modular structure for the organization of binary files and their analysis (see figure. 1). Viper is aimed at virus analysts and exploit developers, but it is also useful for conventional reversers because of the interesting modules, the database of which is constantly updated. Also, an important advantage is that it is written in Python and allows you to modify it on the fly for yourself. Viper is an open source, BSD-licensed collaborative development that relies heavily on community input.
2. Getting the source code

We will also analyze the original apk of the application to detect signs of malware. Apk structure: META-INF/ contains the manifest file, the signature and the list of resources in the archive, lib/: native libraries that run on specific device architectures (armeabi-v7a, x86, etc.), res/: resources, such as images, that have not been compiled into resources.arsc AndroidManifest.xml: describes the name, version, and contents of the APK file. classes.dex: compiled Java classes to be run on the device (file .DEX) resources.arsc: Compiled resources, such as strings used by an application (file .ARSC).

The apk file itself is a regular archive that stores the source code of the application that we will analyze. The problem is to get the source code of the application. Applications for android are usually written in the Java programming language.

Java Virtual Machine (Java VM, JVM for short) — the Java virtual machine is the main part of the Java runtime system, the so-called Java Runtime Environment (JRE). The Java virtual machine executes Java bytecode previously generated from the Java source code by the Java compiler (javac). The JVM can also be used to run programs written in other programming languages. For example, the source code in Ada can be compiled into Java bytecode, which can then be executed using the JVM.

The JVM is a key component of the Java platform. Because Java virtual machines are available for many hardware and software platforms, Java can be considered both as a middleware and as a standalone platform. Using a single byte code for many platforms describes Java as» compiled once, run everywhere» (compile once, run anywhere).

But, the standard source code has the extension class, in our case we have files with the extension smali. Smalli is the bytecode of the dalvik virtual machine, which is issued when the apk is decompiled via apktools. Java + Android SDK = Dalvik

Dalvik + baksmali = disassembly of code bytes (smali)
Smali is similar to Java bytecode. But, since there are a lot of registers in ARM-architecture processors, Google corporation decided to save and replace long walks in memory (in stack, as in JVM) with quick trips to registers. Therefore, the main difference between Dalvik bytecode and JVM bytecode is case-oriented. For example:

System.out.println («Hello») in JVM-byte code looks something like this:

getstatic System.out
// PrintWriter on glass
ldc «Hello»
// PrintWriter, « Hello — two objects on the stack
invokevirtual PrintWriter.println(String): void
// both objects are taken from the stack, a virtual method with the «Hello» parameter is executed on the PrintWriter»

In Dalvik-byte-code:
get-object v0, System.out
// PrintWriter in zero case
const-string v1, «Hello»
// «Hello» in the first case
invokevirtual PrintWriter.println(String): void, v0, v1
// call the virtual println method on object V0 with parameter v1.

Actually, move object shifts the object from register to register, and all these v0, v1 and so on — are registers (value registers). In addition to them there is another tool registers - P0, P1, etc.

These registers specify the values passed to the method as parameters. If the method is not static, p0 will contain this. invoke - direct resembles invoke special in the JVM - it calls a non-virtual non-static method. An example is the final (and/or private) instance methods.

In addition, in smali (as opposed .class) there is no pool of constants, they are set directly in the code. This is because there is only one common pool of constants for all classes that are in the same dex — this reduces the amount compared to .class and saves time spent on interning strings when loading classes.

### 3. Analysis of the application

First, let's study AndroidManifest.xml. Each application must have an Android Manifest file in its root folder.xml (which is exactly what it is called). The manifest file contains important information about the application that is required by the Android system. Only after receiving this information can the system execute any application code. Among other things, the manifest file does the following:

- it specifies the Java package name for the application. This package name is a unique identifier for the application;
- it describes the components of the application—the operations, services, broadcast receivers, and content providers that make up the application;
- it contains the names of the classes that each component implements and publishes their capabilities (such as which Intent messages they can accept). Based on these declarations, the
Android system can determine which components make up the application and under what conditions they can be run.

- it determines which processes will host the application components, declares what permissions must be granted to the application so that it can access the protected parts of the API and interact with other applications.

Also declares the permissions required to interact with the components of this application. Contains a list of Instrumentation classes that provide profile information and other information when the application is run.

These declarations are present in the manifest file only during application development and debugging and are removed before it is published. It declares the minimum level of Android API that the application requires and contains a list of libraries that the application should be associated with.

### 4. The information that we can get after manifest studying

Here is some information, which we can get after manifest program studying:

- **android. permission. CHANGE_WIFI_STATE** allows applications to change the status of the Wi-Fi connection. This permission can be very dangerous because someone can connect to the device under the android operation system by the Wi-Fi;

- **android. permission. CHANGE_CONFIGURATION** allows an application to change the current configuration. This permission can be very dangerous because someone can change some security rules, that can provide wide access for file system or remote control for hole device;

- **android. permission. BROADCAST_STICKY** - allows an application to broadcast sticky intents. This permission can be very dangerous because it can help to distribute virus programs to other devices;

- **android. permission. RECORD_AUDIO** - Allows the program to record audio. This permission can be dangerous because someone can listen all around the phone for the time it is on:

- **android. permission. NFC**- allows applications to perform NFC i/o operations. This permission can be very dangerous because someone can emulate NFC device and force access for wireless banking card system. That can help them to stole money from them;

- **android. permission. RECEIVE_BOOT_COMPLETED** - allows the app to receive the ACTION_BOOT_COMPLETED that is broadcast after the system is booted. This permission can allow any application to emulate boot;

- **android permission. WRITE_SETTINGS**- allows the program to read or write system settings. This permission can provide security overcome;

- **android. permission. DISABLE_KEYGUARD** allows applications to disable the key guard, if it is not protected. This permission can provide key listening functions;

- **com. android. launcher. permission. INSTALL_SHORTCUT** - allows the program to install a shortcut in Launcher. This permission can provide integration any program;

- **com. android. launcher. permission. UNINSTALL_SHORTCUT** allows the program to uninstall a shortcut in Launcher. This permission can help to remove any traces of virus programs in the end of hacking.

As you can see we have rather interesting opportunity for hacking android systems. For example you can use

«com. android. launcher. permission. INSTALL_SHORTCUT» permission for installing «android. permission. NFC» allows applications to perform NFC» for emulate credit card payment and then use «com. android. launcher. permission. UNINSTALL_SHORTCUT» for cleaning logs. Then, there will be no traces of payment method.
5. Conclusions
Initially, the concept of reverse engineering has been used only for hardware projects, but now it is also very widely used in software, improves existing software or duplicates it. The purpose of applying this process in software is to develop software using a programming language (which can be understood by any programmer) that is compiled using compilers and produces binary code (that is, a machine language that can be understood by the system), so the reverse development process appears when this machine language code requires conversion back into readable code using decompiles. It is necessary to say, that software reverse engineering concept it not an exact showing program code inside, because there are a lot of imaginative dimension in the work process and analyzing. But this work is necessary part of virus finding process.

References
[1] Brednik A V 2013 Security issues of cloud computing, Analysis of methods of protection of the clouds from cloud security alliance Almanach of modern science and education 10 35–8
[2] Sommerville I 2011 Software engineering (New York: Pearson, Addison-Wesley)
[3] Hoff C 2011 In: Security guidance for critical areas of focus in cloud computing 12–20
[4] Zhigalov K 2017 Using cloud computing technologies in IP-video surveillance systems with the function of 3d-object modelling ITM Web of Conferences 10 02004
[5] Zhigalov K 2016 Use of the game technologies engines for the purpose of modern geographic information systems creation ITM Web of Conferences 6 030118
[6] Rehman S, Kriebel F, Shafique M and Henkel J 2014 Reliability-driven software transformations for unreliable hardware IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. 33 1597-610
[7] Pluzhnik E, Nikulchev E and Payain S 2014 Optimal Control of Applications for Hybrid Cloud Services IEEE World Congress on Services (SERVICES) (Anchorage, AK: USA) pp 458-61