

REVERSE ENGINEERING AND ANDROID APPLICATION SECURITY

Branko Džakula , Slobodan Đukanović***

Keywords: reverse, engineering, android, application, security.

Abstract: Reverse engineering process in Android applications is, due to the fact that Android applications are interpreted, significantly simplified. The paper illustrates the ease of extraction of the resource files, as well as methods to circumvent the limitations of the encrypted source code. On the example of one Android application, presented are methods for reverse engineering using tools to decompile closed application binaries. The most common threats are described using some of the methods of reverse engineering, as well as methods of protection against such threats.

1. INTRODUCTION

Since the launch of the first smartphone, market has developed and a new branch of programming with the goal of expanding the possibilities of smartphones using applications. Developing applications for smartphones has opened the door to international companies in the form of promoting their services through applications as well as facilitating the provision of services to end users. To meet these needs and remain competent, software companies and individuals are rapidly changing technologies, tools, and learning new programming languages and platforms. The pressure of the market, which dictates the short deadlines definitely leaves severe consequences on the quality of software products, and the prioritization of testing and information security regularly go to the bottom of the to-do list or do not make the list at all.

In scenarios when using a smartphone for private and business e-mails, social networks and e-business, consequences everyone wants to avoid is for someone else to have access to their private data and this is exactly what is a common case in today's world of smartphones. All platforms are affected, and Android is by no means an exception. There is a frightening number of methods that can be used by experienced developers to steal user data and reverse engineering is only one of them.

This paper presents methods of reverse engineering through a case study on a sample Android application. Following the case study, presented are methods of protecting Android applications and user data from reverse engineering.

* Branko Džakula, Spec. Sci, Air Serbia, Information Security Officer, Technology Division (e-mail: branko@dzakula.com)

** Slobodan Đukanović, PhD, Associate Professor, Faculty of Electrical Engineering, University of Montenegro, Montenegro (e-mail: slobdj@ac.me)

A. Reverse engineering

Reverse engineering [1] is a process of analyzing, detecting and dismantling the technical characteristics of a product or service with the goal of revealing the way they function or to detect security vulnerabilities. In a world of software engineering, reverse engineering is a process of discovering the source code of the executable file and is most often analyzed in the bytecode.

B. Goals

The main objective of this paper is finding good engineering practices in the information security field of Android applications. To better understand the techniques of reverse engineering, realistic scenarios are used to demonstrate the capabilities of reverse engineering performed on an Android application using arsenal of free tools available to everyone.

Android developers who want to gain a basic idea of security trends and good practices, this paper may be sufficient. However, for someone who wants to further explore the methods of reverse engineering, this is only the starting point in the world of reverse engineering.

2. ANDROID PLATFORM

Android applications are programmed in Java SDK (Software Development Kit) [2], that is publicly available for download, where Java code is compiled in Android executable package (APK). These APK files are executed in the Dalvik virtual machine (DVM) developed exclusively for the Android system. It is important to note that Android applications are interpreted, which means there is no executable code, and it is much easier to execute an operation of reverse engineering.

Android applications interact with the system, but also among themselves using various inter-process communications. Each component of the APK file plays a different role in the behavior of applications and can be activated separately.

Android operating system is composed of software components which may roughly be divided into five sections [3] and four main layers, as illustrated in Fig. 1.

A. Linux Kernel

At the bottom layer is Linux 2.6 with about 115 extensions. It provides the basic functionality of the system as well as process management, memory management, devices such as camera, keyboard, screen, etc. Also, the kernel handles all the things that Linux is really good at, such as network connectivity and management of a huge range of device drivers.

B. Libraries

At the top of the Linux kernel is a library including the open-source *WebKit* Web browser, well-known library *libc*, *SQLite* database that is useful repository for storing and sharing data controlled by the application, libraries for playback and recording of audio and video recordings, *SSL* library responsible for safety on the Internet and others.

C. Android Runtime

This is the third part of the architecture and is available on the second layer from the bottom. This section provides the key component called DVM (Dalvik Virtual Machine) which is a type of Java Virtual Machine specifically designed and optimized for Android. DVM uses the basic features of Linux such as memory management and multi-threading. It allows any Android application to run as a separate process, with its instance in the DVM. Android Runtime also provides a number of key libraries that allow Android developers to write Android applications using a standard Java programming language.

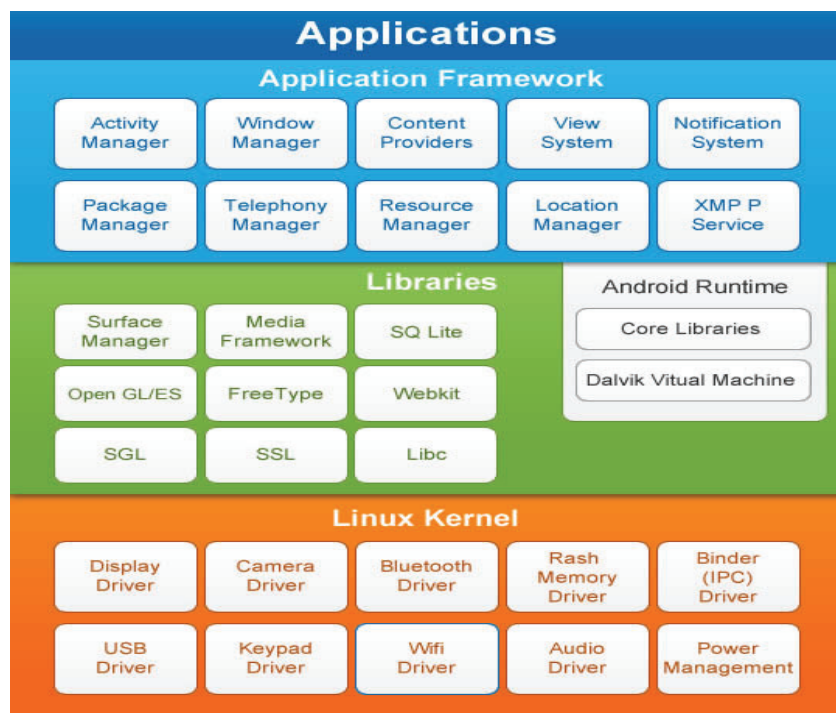


Fig. 1. Android application architecture diagram. [4]

D. Application Framework

Application framework layer provides a much higher level of service applications in the form of Java classes. Developers are allowed to use these services in developing applications.

E. Application

At the supreme layer are applications. They are developed and implemented exclusively in this layer. Examples of such applications are Contact Manager, Web Browser, Games, E-mail clients and others.

3. ANDROID APPLICATION MAIN COMPONENTS AND DEVELOPMENT

Application components [5] are the essential building blocks of an Android application. Each component is a different point through which the system can enter an application. Not all components are actual entry points for the user and some depend on each other, but each one exists as its own entity and plays a specific role. Each component is a unique building block that helps define an applications overall behavior.

Following are five main components of any Android application:

A. Activity

Activities are individual displays of a window with the user interface. Even though they work in conjunction, they are independent of each other. For example, one activity can display a list of new messages, while the other creates a message, and the third shows the individual messages.

B. Service

Services are responsible for the operations carried out for a long time, working as isolated processes and can be started from the activities, but do not require the user interface. For example, service can play music in the background while the user uses the screen to create a text message.

C. Content Provider

Content providers manipulate data sets that are shared between applications and act as bridges to other application components. For example, many applications with data input such as Notes use content providers for data storage.

D. Broadcast receiver

Broadcast receivers respond to the broadcast system releases. For example, they inform the device that display has been switched off or the battery is low. Even though they do not display user interface, broadcast receivers can create a notification in the status bar.

E. AndroidManifest.xml

All Android applications must have a manifest file in their root directory that provides all the basic and necessary information for the application to work such as: application name, package name, version, minimum API used to run, components, necessary permissions, required libraries and such.

Tools used for Android application development are included in Android SDK. After SDK installation it is possible to access development tools through Eclipse IDE (Integrated Development Environment), ADT (Android Development Tools) plugin or from the command line. Developing applications using the Eclipse programming environment is recommended for efficiency and simplicity. However, it is possible to develop applications with other IDEs or via a simple text editor.

The basic steps in the development of applications (with or without Eclipse) are shown in the diagram in Fig. 2.

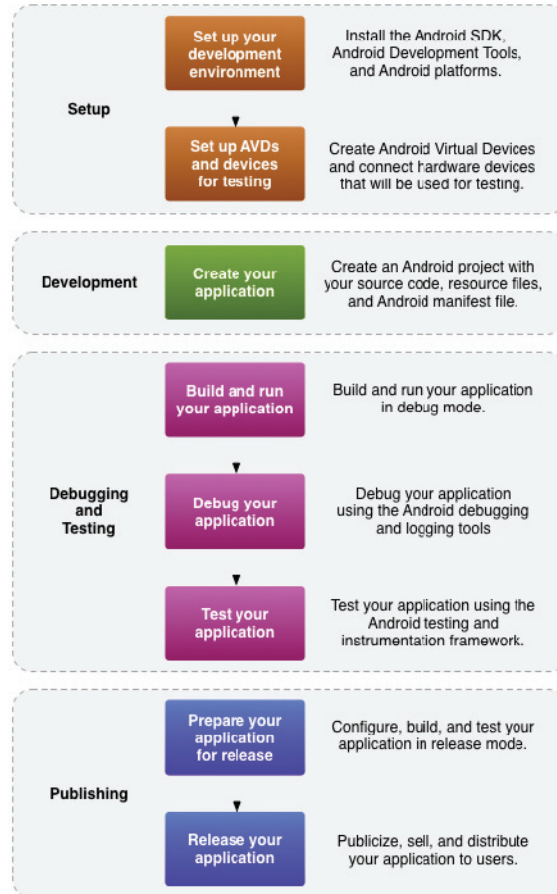


Fig. 2. Basic steps in application development [6]

During the setup step, all needed tools are installed and configured along with virtual Android devices that are later used for application testing. Development step focuses on coding the actual application. Debugging and testing step is the most important step in application development process and often lasts the longest. In this step all mistakes, bugs and later discovered errors are being detected and removed. Application is tested on the virtual platform as well as actual Android devices. Debugging tools used in this step are usually integrated debugging tools in Android SDK along with JDWP compliant debugger. As this paper focuses on application security it is gravely noted that security testing is often times skipped and overlooked as it takes a lot of time to perform and detected security vulnerabilities appropriately patched. The last step is alpha testing and finally publishing the finished product to end users.

4. REVERSE ENGINEERING OF AN ANDROID APPLICATION CASE STUDY

As previously defined, reverse engineering of an Android application is a process of discovering its source code and its further analysis. Main focus of this paper is using reverse engineering to determine if a potential Android application has any security vulnerabilities or unnecessary requests for security permissions that can lead to malicious activities such as personal user data collection and processing without the knowledge of the user.

Many free and paid tools are available to correctly perform the deepest reverse engineering on an Android application. Free tools were used to demonstrate reverse engineering in this paper's case study. AppSec Labs [7] developed a virtual machine called AppUse, a unique Linux-based platform for security testing of Android applications. In this paper, all case study examples are performed in AppUse platform run on a regular Windows PC's VMWare Player [8]. AppUse platform has built-in tools and add-ons to complete certain tasks such as decompiling, conversion, extraction, vulnerability scan, virtual proxy server routing and so on with the goal always being Android application reverse engineering and security.

To start with the process, we need to introduce a test application that will be used for testing purposes such as Kotor Guide Android application developed by one of the authors of this paper. Application is a virtual tourist guide developed to help tourists organize their visit in the Montenegrin town of Kotor. Main method used for developing this application is Web View which dictates that many resources this application uses are located on the web. Performing reverse engineering on this specific application will be able to tell if the application has any hidden gems or potential security vulnerabilities that could be exploited. Welcome screen of the application and main design are shown in Fig. 3.



Fig. 3. Kotor Guide Android application interface

A. APK file extraction

Extraction of the APK file is a process of file decompression from one to multiple readable objects. After the APK file extraction, that is a simple unzip or extract here operation, depending on the compression tool available at the moment (7zip, WinRAR, etc.) many readable objects are already present such as multimedia files used by the application (icons, photos, video files, etc.), although all of the application logic and graphics interface of the application are hidden in the binary archives such as classes.dex and AndroidManifest.xml. For these files to be readable further decompiling is needed.

Note: If the goal was to simply read the multimedia files that was already achieved reading through the `./res/drawable-*` directories.

B. Apktool decompiling (disassemble)

Apktool [9] is preinstalled in the AppUse platform and available to use using simple shell scripting. The decompiled code can be later modified and repacked again using the same tool. This is generally used for pirating purposes or competition rivalry later described in section 5. This process is represented in the diagram below in Fig. 4. and can be viewed in the code extracted from shell in Fig. 5.

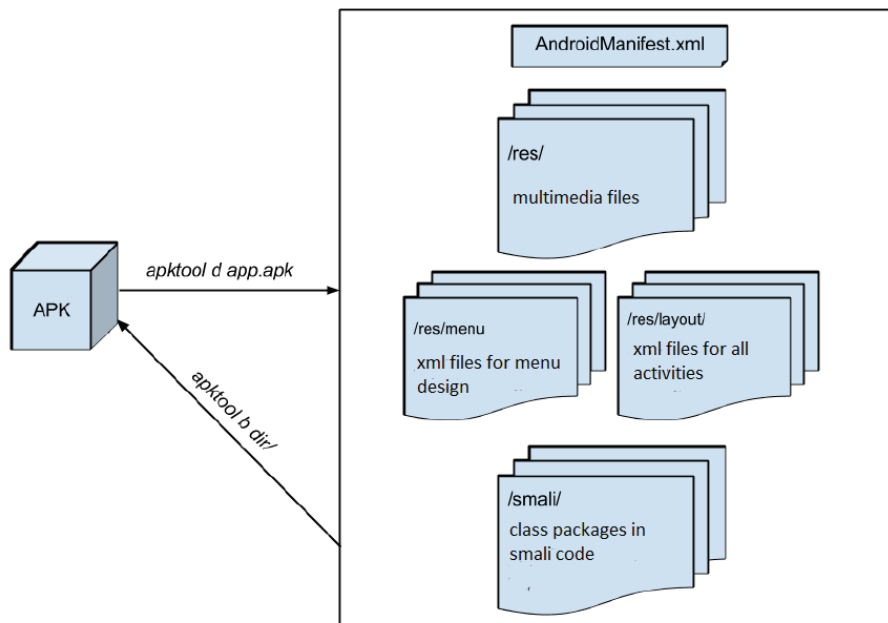


Fig. 4. Diagram representation of apktool decompile and compile option on any apk file

```

root@dev-virtual-machine:~/Desktop/AppUse/Targets# apktool d
./KotorGuide.apk ./OutputFolder
I: Baksmaling...
I: Loading resource table...
I: Loaded. Decoding AndroidManifest.xml with resources...
I: Loading resource table from file:
/root/apktool/framework/1.apk
I: Loaded.
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Done.
I: Copying assets and libs...
root@dev-virtual-machine:~/Desktop/AppUse/Targets#

```

Fig. 5. Apktool decompile option used on KotorGuide.apk file

After the decompile process is finished AndroidManifest.xml file is clearly visible and readable in the native xml format. Fig. 6. compares AndroidManifest.xml file before and after the decompile process using Apktool. The before screenshot (left) shows the AndroidManifest.xml file after the extraction of KotorGuide.apk file using simple extract here or unzip option explained in step A. The after screenshot (right) shows the AndroidManifest.xml file after the decompile process using Apktool explained in step B.

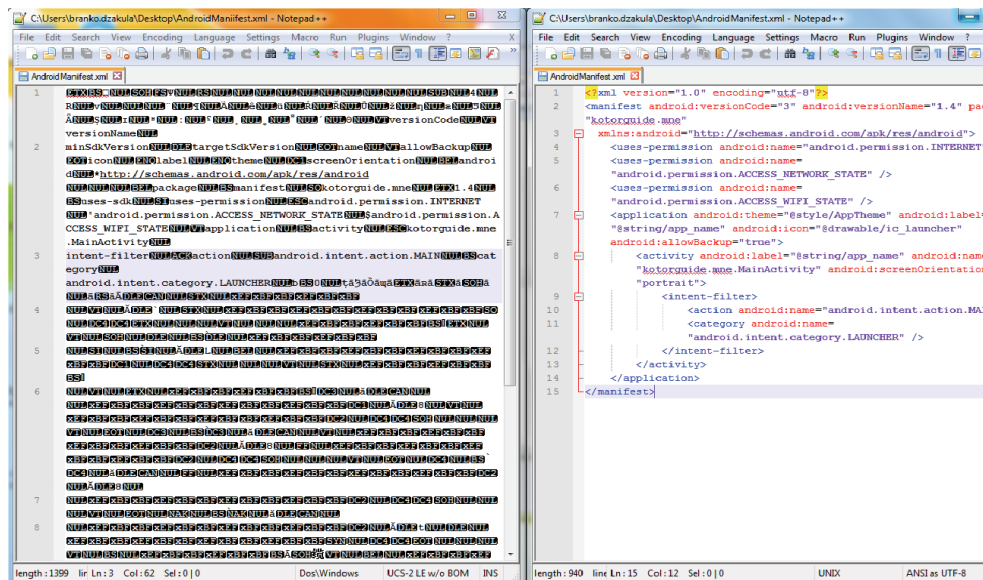


Fig. 6. AndroidManifest.xml file before (left) and after (right) decompile process using Apktool

Decompiling the AndroidManifest.xml is not the only byproduct of the Apktool decompile process, class files of the application are also decompiled in the different format called *smali* [10] code (assembler/disassembler for dex format used by Dalvik). In the class directories MainActivity.smali and MainActivity\$.smali can be viewed and understood following basic programming logic. Most programmers settle with this simple reverse engineering method to understand the logic behind the application and search for security vulnerabilities.

Note: Using Apktool decompile process smali code is produced to represent the class file while using the unzip/extract here method, dex files represent the class files of the application that will be later decompiled to demonstrate alternate methods of Android application reverse engineering using dex2jar [11], dexdexer [12] or dexdump [13] tools.

C. Source code analysis

Detecting security vulnerabilities might not come as an easy job, however before using complex tools to search for suspicious code programmers look for the key objects in the code to detect most common mistakes that are made in Android application development such as username or password hardcoding, links, database encryption, design flaws that could open backdoors in the application and so on.

In Fig. 7. highlighted code represents hardcoded links in the WebView method that might be accessed from the application. One of those links points to an admin page with a login screen <http://m.kotorguide.me/administrator>. This information should not be visible in the application code and admin page should be hidden limiting the users to a user only interface thus lowering chances of someone breaking into website admin panel and gaining access to entire website content and databases.

```
.line 30
    iget-object v1, p0, Lkotorguide/mne/MainActivity;-
>mainWebView:Landroid/webkit/WebView;

    const-string v2, "http://m.kotorguide.me"

    invoke-virtual {v1, v2}, Landroid/webkit/WebView;-
>loadUrl(Ljava/lang/String;)V

    .line 31
    iget-object v1, p0, Lkotorguide/mne/MainActivity;-
>mainWebView:Landroid/webkit/WebView;

const-string v1, "http://m.kotorguide.me/administrator"
```

Fig. 7. Hardcoded links in the application smali code

Further analyzing the smali code and application permissions additional findings are shown in Fig. 8. Application collects information about users Wifi connection and handles

the IP address. According to the purpose of the application and its description this is unnecessary action and could put user at risk of Wifi attacks and traffic interception.

```
    invoke-virtual {v0}, Landroid/net/wifi/WifiManager;-  
>getConnectionInfo()Landroid/net/wifi/WifiInfo;  
  
    move-result-object v0  
  
    invoke-virtual {v0}, Landroid/net/wifi/WifiInfo;-  
>getIpAddress()I  
  
    move-result v0
```

Fig. 8. Wifi information is handled and stored in the application

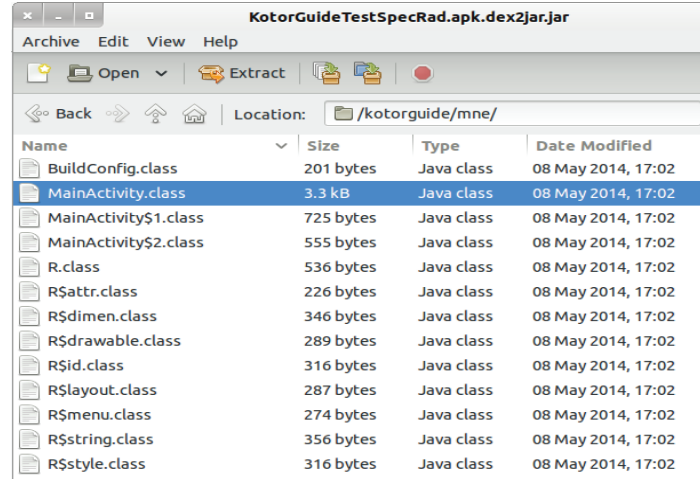
Unfortunately, it is no rare case that Android developers hardcode username and password data into the application source code leaving it completely unprotected and fully accessible by anyone with basic knowledge of Android programming. Using apktool is only one method of exposing hardcoded data and many more tools are freely available online to achieve same results.

One of the alternate methods used for Android application reverse engineering with the goal to produce readable and native Java code is using dex2jar tool. Similar to apktool, dex2jar is also preinstalled in AppUse platform and accessed through shell scripting as shown in Fig. 9.

```
root@dev-virtual-machine:~/Desktop/AppUse/Targets# dex2jar  
./KotorGuide.apk  
version:0.0.7.8-SNAPSHOT  
3 [main] INFO pxb.android.dex2jar.v3.Main - dex2jar  
./KotorGuide.apk -> ./KotorGuide.apk.dex2jar.jar  
Done.  
root@dev-virtual-machine:~/Desktop/AppUse/Targets#
```

Fig. 9. Dex2jar tool used on KotorGuide APK file

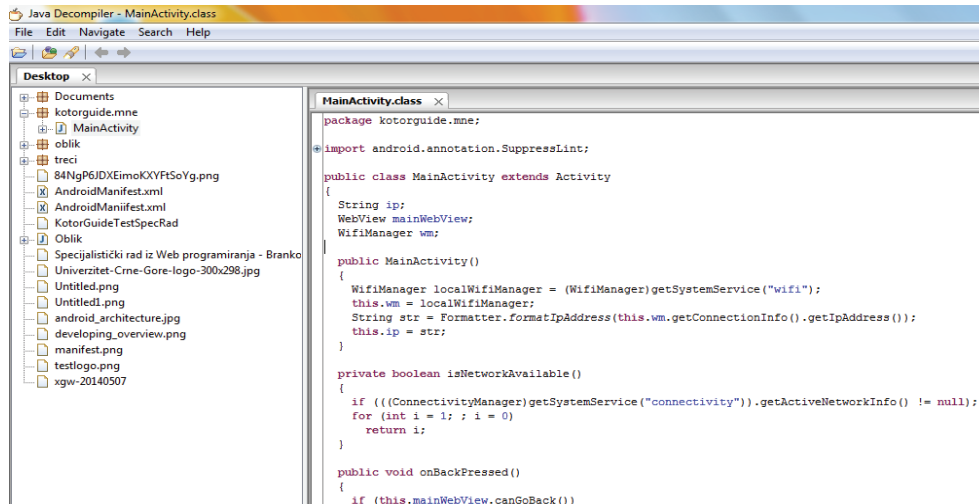
Result is shown in Fig. 10. as a directory containing application class files in recognizable Java class type format.



Name	Size	Type	Date Modified
BuildConfig.class	201 bytes	Java class	08 May 2014, 17:02
MainActivity.class	3.3 kB	Java class	08 May 2014, 17:02
MainActivity\$1.class	725 bytes	Java class	08 May 2014, 17:02
MainActivity\$2.class	555 bytes	Java class	08 May 2014, 17:02
R.class	536 bytes	Java class	08 May 2014, 17:02
R\$attr.class	226 bytes	Java class	08 May 2014, 17:02
R\$dimen.class	346 bytes	Java class	08 May 2014, 17:02
R\$drawable.class	289 bytes	Java class	08 May 2014, 17:02
R\$id.class	316 bytes	Java class	08 May 2014, 17:02
R\$layout.class	287 bytes	Java class	08 May 2014, 17:02
R\$menu.class	274 bytes	Java class	08 May 2014, 17:02
R\$string.class	356 bytes	Java class	08 May 2014, 17:02
R\$style.class	316 bytes	Java class	08 May 2014, 17:02

Fig. 10. Class files produced using dex2jar tool on KotorGuide.apk file

In order to finally open these class files and read the source code, additional tool is needed called JD-GUI tool [14] used for decompiling class files into readable Java format. In Fig. 11. JD-GUI does a very good job presenting decompiled classes in almost native Java development environment such as Eclipse or IntelliJ IDEA. It is important to note that code produced from decompile process is tools best guess at the encrypted source code making it only partially reliable. All native programming structure and logic developed by the original programmer is partially lost but does not stand in the way of detecting security vulnerabilities and/or exploiting the applications data.



```

package kotorguide.mne;

import android.annotation.SuppressLint;

public class MainActivity extends Activity
{
    String ip;
    WebView mainWebView;
    WifiManager wm;

    public MainActivity()
    {
        WifiManager localWifiManager = (WifiManager) getSystemService("wifi");
        this.wm = localWifiManager;
        String str = Formatter.formatIpAddress(this.wm.getConnectionInfo().getIpAddress());
        this.ip = str;
    }

    private boolean isNetworkAvailable()
    {
        if (((ConnectivityManager) getSystemService("connectivity")).getActiveNetworkInfo() != null);
        for (int i = 1; i = 0)
            return i;
    }

    public void onBackPressed()
    {
        if (this.mainWebView.canGoBack())

```

Fig. 11. Decompiled MainActivity.class file using JD-GUI tool

5. THREATS

Case study performed in the previous section demonstrates only a small segment in the entire Android platform security subject. In order to assess the big picture, this section will refer to most important security issues identified on Android platform.

Following are the most recognizable security threats in Android applications:

A. Reversible code

Undeniable fact states that Android application are developed in Java programming language that in its core simplifies the process of reverse engineering. Java architecture along with Java virtual machine was always the best environment for decompiling executable applications.

B. Root access

Android operating system is Linux based that strictly defines access controls of all applications and users to the system. Performing a well-known *rooting* process on the Android device (also commonly known as *Jailbreaking* on Apple devices) user gains root admin access to all data, services, application and hardware on the device. This option is regularly used by researchers, developers and security analysts, however regular user might undergo this process for many reasons such as gaining access to some third-party applications, removing the GSM operator lock or performing hardware upgrades. Rooting will make the device highly vulnerable to hacker attacks and malware as device access dictates that any unauthorized access to the device will grant the attacker or malicious application the same root admin rights as the user.

C. Tools

Following the examples demonstrated in this paper using free tools it is easy to assume that availability of these tools is high on the internet and present in large numbers. Following the tools availability many step by step tutorials can also be found alongside the tool download page only making more and more users capable of performing malicious attacks and reverse engineering with unethical goals on Android applications.

D. Piracy and competition rivalry

Android applications are no strangers to being targets of piracy mainly to unlock payed features of the application for free and offer such modified and fully unlocked application to users via torrent sites or forums. This process is widely known as “cracking” and is achieved by performing reverse engineering on the application with code modification and repacking as shown in Fig. 4. Section 4. b.

Competition rivalry is more often than not a kick starter of performing reverse engineering on competitions products for research and sometimes unethical source code and algorithm unauthorized use to boost the performance of their own products and stay ahead of the game.

E. Malicious software

When discussing security malwares, viruses, spywares and any other malicious software are unavoidable subject and very much present in Android applications field. Hackers intend to use reverse engineering on well-known and trusted applications to inject malicious code and trick the user into installing such applications to gain remote access to the user's device. Consequence may vary depending on the type of malicious software installed but it's never in the best interests of the user.

6. METHODS OF PROTECTION

Complete protection from reverse engineering is unfortunately non-existent. The moment the application is stored in the memory of the device, regardless of the platform, it's service instructions are readable and with more or less work it can be decompiled to its source code. With Java virtual machine the whole process is even simpler and only methods that can be used to protect it are based on code masking and complex coding to make it more difficult to decompile and understand. Following are best practices in code protection against reverse engineering and all malicious activities that can occur during and after the process:

A. Code masking

Code masking is a process of modifying the source code in its development phase with the goal to reduce logic and understanding of the code to the third party without the loss of functionality on a basic level.

Some of the usual techniques include string cyphering, randomly generated code segments, misleading algorithms that have no functionality and so on. Goal is to confuse the programmer trying to reverse engineer the product and make the source code unreadable. These methods have their ups and downs, while delivering security, they reduce the performance of the application and extend the hardware requirements to run it.

There are many tools developed to help with code masking and most popular are ProGuard [15] and DexGuard [16] that mask the code, remove unnecessary steps, cypher strings and much more.

B. Separating the logic of the application to the cloud

If the applications functionality depends on internet connection, it is widely suggested to separate the entire logic segment of the application to a cloud server thus protecting the most important part of the application with many authentication mechanisms and intrusion detection systems on an enterprise level that simply cannot fit or be handled by the ordinary Android device. Cloud servers can provide many other functionalities such as license verifications and better protection of the customer's personal data.

C. Digital signatures

Widely popular and known feature of any secure communication are digital signatures that use certificates to ensure that the author of the application is genuine. Digital signatures do not protect the application from reverse engineering but provide security to the end user that the application is trusted and from the original developer.

7. LEGAL ASPECTS

Concept of reverse engineering outside the information technologies is valid for a very accepted skill. It is a process of learning, and exploring the world around us, despite the fact that we live in a world that is substantially created by humans. In this context, each different attitude may sound illogical, but in the last few decades, the application of these techniques are being challenged, and the court proceedings stopped being a rarity.

The universal solution does not exist, and every legal system defines its own rules. It is important to note that the turbulence in this field is still common, which only shows that the system applied is far from perfect.

In the United States [17], for example, even though the product is protected with business secrets, it's "dismantling" is still legal as long as the product was purchased legally. The system of protection for patents and copyrighted material in the United States much relies on reverse engineering techniques, using the information obtained as evidence in legal proceedings. On the other hand, it is interesting to note that this practice is a direct violation of the EULA (End User License Agreement) agreement, which often explicitly prohibits practices for a deeper understanding of software solutions.

The European Union [18] has adopted a slightly clearer laws, which allow reverse engineering for purposes of interoperability software, but forbid it to create a competing product.

It also prohibits any public disclosure of information obtained by these techniques.

8. ETHICS

This problem was and remains the subject of long debates, without clear prospects for a definitive answer. The main argument [19] against reverse engineering is intellectual property. If an individual or organization has created a product or idea, it is okay to "dismantle" the product in order to discover its secrets? Large companies that invest significant time and money in the development of their products generally do not think so. Why should manufacturers invest their resources in building intellectual property, if competitors can use reverse engineering to steal their results with minimal effort?

On the other hand, there are also numerous benefits of such procedures. Reverse engineering can be used for the purposes of interoperability of products, and perhaps more importantly, in order to check the software on any harmful, unethical or illegal activities that otherwise might not be discovered.

The global trend in the world that promotes freedom, transparency and liberalized market can perhaps predict the resolution of such issues. Until then, the topic is open for debate.

9. CONCLUSION

Android platform has certainly become an interesting field of research and a valuable market. As applications for this platform are becoming more complex and demanding, the data protection needs to keep getting better, and companies are slowly but surely starting to recognize that.

Experience has shown that it cannot be expected from developers to have a thorough understanding of security methods, especially if the project managers do not insist on them and do nothing to educate the staff.

Awareness is slowly changing and more and more companies are thinking in advance and talk about security as a topic that should not be ignored. Companies understand that security methods must be included directly in the initial cycle of software development, and that constant training and education become necessary practice.

Reverse engineering methods described in this paper deliver a clear picture that information security is greatly neglected and customers are being offered more and more solutions to automate their everyday activities and digitalize their personal communications and data without the actual security ever being a guarantee. This is a global issue that requires immediate attention along with major investments.

There is no perfect solution to security, since it is clear that the attackers are always willing to go a step further, but the risks can be significantly reduced even with basic knowledge and understanding of the platform and the most common attacks.

Reverse engineering is only one of the methods that may affect the security of Android applications and this paper has put strict focus on it. It must not mislead the reader into the understanding that this is the field of security vulnerabilities with the highest risks, nor the most widespread. There are still many aspects of the attacks on the Android platform, which are not mentioned here, and this work would likely be more complete if they were devoted the least bit of attention.

It should be noted that reverse engineering techniques fall under static program analysis, and there are certainly other ways of observing the behavior of applications through dynamic analysis of their activities at the time of execution. This topic may be a logical continuation of this paper in the future.

REFERENCES

- [1] Amalfitano, D. (2011). *“Reverse Engineering and Testing of Rich Internet Applications “*. PDF file. Naples: University of Naples Federico II.
- [2] Google. (2015). *“Android Studio IDE - SDK”*. [Online]. Available: <http://developers.android.com> . [Accessed: Sep. 19, 2015].
- [3] Stack Exchange. [Online] Available: <http://programmers.stackexchange.com/> . [Accessed: Sep. 28, 2015]
- [4] Developer.com, Android Application Architecture Diagram Source [Online]. Available: <http://www.developer.com> . [Accessed: Sep. 27, 2015].
- [5] Google. (2015). *“Android Application Fundamentals”*. [Online]. Available: <http://developers.android.com> . [Accessed: Sep. 19, 2015].
- [6] Google. (2015). *“Developer Workflow”*. [Online]. Available: <http://developers.android.com> . [Accessed: Sep. 19, 2015].

- [7] AppSec Labs. (2015). "App Use Platform" . [Online]. Available: <https://appsec-labs.com/> . [Accessed: Sep. 28, 2015]
- [8] VMWare (2015). „VMWare Workstation Player“, [Online]. Available: https://my.vmware.com/web/vmware/free#desktop_end_user_computing/vmware_workstation_player/12_0
- [9] Xdadevelopers Forum (2015). „Apktool v2.0.2.– A tool for reverse engineering of apk files“, [Online]. Available: <http://forum.xda-developers.com/showthread.php?t=1755243>
- [10] Github. (2015). „smali“ [Online]. Available: <https://github.com/JesusFreke/smali>
- [11] Github. (2015). „dex2jar“ [Online]. Available: <https://github.com/pxb1988/dex2jar>
- [12] SourceForge (2015). „dedexer“ [Online]. Available: <http://dedexer.sourceforge.net/>
- [13] Github. (2015). „dexdump“ [Online]. Available: github.com/android/platform_dalvik/
- [14] Java Decompiled (2015) „JD GUI“ [Online]. Available: <http://jd.benow.ca/>
- [15] SourceForge (2015). „ProGuard“ [Online]. Available: <http://proguard.sourceforge.net/>
- [16] Guard Square (2015). „Dex Guard“ [Online]. Available: <https://www.guardsquare.com/dexguard>
- [17] WikiBooks (2015). „Reverse Engineering/Legal Aspects“ [Online]. Available: https://en.wikibooks.org/wiki/Reverse_Engineering/Legal_Aspects
- [18] Woodmann (2015). „Is reverse engineering legal?“ [Online]. Available: <http://www.woodmann.com/fravia/legal.htm>
- [19] Karole Blythe, Jason Mantey (2012). „The Ethics of Reverse Engineering“, PDF file May 2014