Towards the methods of analysis malicious applications for Android operating system

Sergey Staroletov <serg_soft@mail.ru>
Polzunov Altai State Technical University,
Lenin avenue 46, Barnaul, 656038, Russia

Abstract. It is considered to the problem of analysis of Android applications to study a malicious behaviour. The methods of analysis are presented, the general method, which combines different analysis techniques (static, dynamic, decompilation, debugging, logging) is proposed, and information of our software based on it is given.

Keywords: analysis of program behavior; malicious software; data leaks; Java; software and technical expertise

DOI: 10.15514/ISPRAS-2018-30(1)-4

For citation: Staroletov S.M. Towards the methods of analysis malicious applications for Android operating system. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 1, 2018, pp. 55-68. DOI: 10.15514/ISPRAS-2018-30(1)-4

1. Necessity for the work

In the recent years, mobile phones with a mobile operating system are widely being used, and now we observe the explosive growth of mobile devices in the world.

According to forecasts [1], in 2018 the number of smartphones in the world will be over 50% of the total number of phones (see fig. 1).

More and more people do their daily tasks by using software for phones, including making the financial transactions.

However, the computer literacy for today's smartphone users is not keeping pace with the progress in the field of mobile devices. It can mean that soon we are going to have about 2.5 billion potential victims of intruders; they would use mobile phones as an instrument to steal private data and money.

It is known that in the past year hackers in Russia were stolen 349 million rubles [2] from the owners of phones under Android OS (about 2 rubles in mean from each Russian citizen), which is five times more than a year ago.

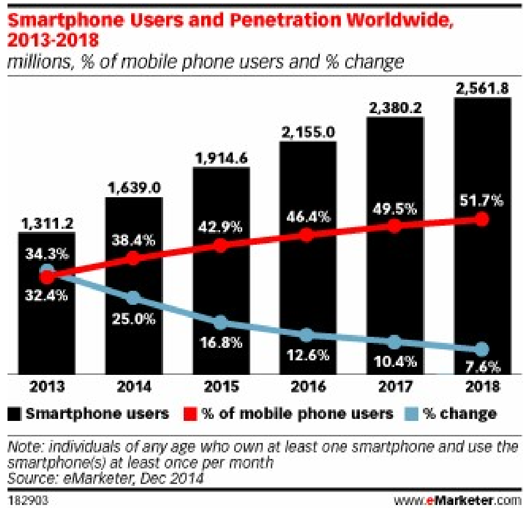


Fig. 1. Number of smartphones in the world will be doubled for the last 5 years [1]

Choosing the Android OS by intruders is primarily caused by its prevalence, availability of cheap Chinese phones under it.

Here we have a problem: typical users (especially from the countryside) are not even aware that they work with a minicomputer which can hook a trojan program and such program could get access to user data, receive commands to work following the hacker's request.

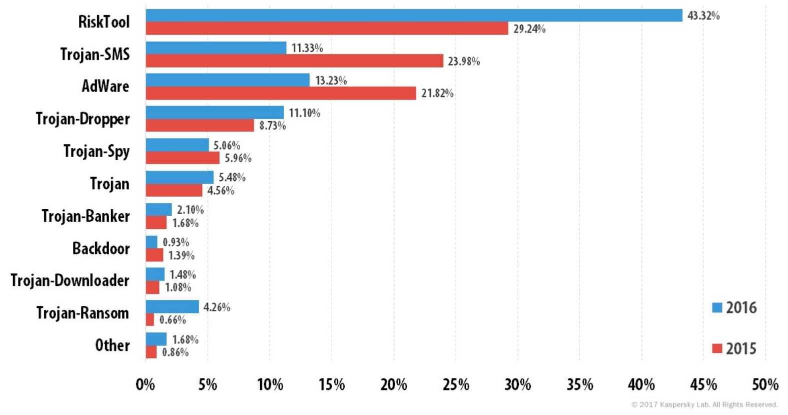


Fig. 2. Distribution of new mobile malware by the type in 2015 and 2016 (Kaspersky)

According to a Kaspersky's report (see fig. 2), the most frequent malware types now are “RiskTools” (special software or software with vulnerabilities used by intruders to enter and control the system), “Trojan-SMS” (trojans for sending SMS to short numbers) and some other trojan types including "Trojan-Banker" (trojans for stealing money from a bank account).

In 2015, local representatives of the Police addressed to the Department of Applied Mathematics at Altai State Technical University in order to develop methods, algorithms, and software for the analysis of the smartphones injured by the trojans (like types as "SMS" and "Bankers"), to do digital forensic science to analyze the malicious applications. We have a situation now: there are no generally accepted methods to do it.

2. The goals of the analysis

Android OS itself is a special Linux Kernel, which contains Dalvik virtual machine for execution of Java applications using their own API.

The application for Android is an apk (Android Package) file, which is a zip archive and it contains some compiled Java-classes of the application in the form of .dex file (Dalvik EXecutable), resources and application's descriptor AndroidManifest.xml within.

A trojan here – is a dual-purpose Android application. It looks like an ordinary application but it was created to steal data or money from victims.

A victim – is a people who handed his smartphone with some trojans to the Police for the analysis since he had suffered by unknown intruder's actions to withdraw his money via the phone.

According to the problem described in the previous section, the goals of the analysis are:

- identify malicious applications among given applications from given phones;
- make some proof of harmfulness of given application;
- study algorithms of the work of applications without having sources of it;
- find the remote hosts which application communicates to, discover sending content, format and protocols);
- discover user's private data leaks;
- compare various malicious applications and group them;
- develop and test a general method to make suchlike analysis.

So, we need dynamic and static methods for analysis the behavior of Android applications. About 50 real recent malicious applications from the real victims (who had lost the money because of intruder's actions) were analyzed and identified, and methods for it were developed.

3. Related work

There is a high number of research papers devoted to malicious applications for Android but most often to identify whether or not the given application is a trojan.

For example, in the work [3] authors explain how to detect a malicious program for Android with using machine learning and static analysis methods. But it is only a part of current research, the main our goal is to make the analysis of the behaviour and prove the harmfulness by the application's actions. The fact of malware can be detected by using VirusTotal tool [5], which checks the signature of given application by a lot of modern anti-viruses.

The Pennsylvania State University, Duke University and Intel lab have developed the series of patches [4] for Android Kernel and libraries to track the data passing through the functions calls in running Android applications to prevent privacy leaks (taints). This approach can be applied to our research with some changes to do dynamic analysis of application's behavior.

4. The analysis

4.1 Malicious applications identifying

The detection of malicious applications in a phone is not so difficult. As a rule, such dual-purpose applications have a small size, a name and an icon disguised as the popular applications (WhatsApp, Skype, Flash Player, Kaspersky, Sberbank Online, etc.).

Since the majority of identified by this research viruses carry out sending messages in a background to short phone numbers or performing USSD-requests, it is possible to identify them by looking at the permissions of applications.

If the Android app uses some API, it must request the appropriate permissions from the system. As a rule, such a request is made during the installation of the application; however, users just ignore the warnings, even if they say that the application will send SMS that can cost money or it is going to control the phone as an administrator.

Permissions are set in AndroidManifest.xml and can be seen in the information about an application. An example of malicious application's permissions is given on table 1.

Table 1. Android application's permissions

android.permission.INTERNET
android.permission.WRITE_SMS
android.permission.READ_SMS
android.permission.SEND_SMS
android.permission.RECEIVE_SMS

The application can be sent for an analysis to one of the anti-viruses (for example, DrWeb online), as well as viewed in the Virus Encyclopedia for its possible actions. Now, we are using the online tool VirusTotal [5], which accumulates information about the signatures of malware files from various antivirus software. However, applications are modified, and some recent viruses could not be recognized, so this analysis is made for educational purposes only.

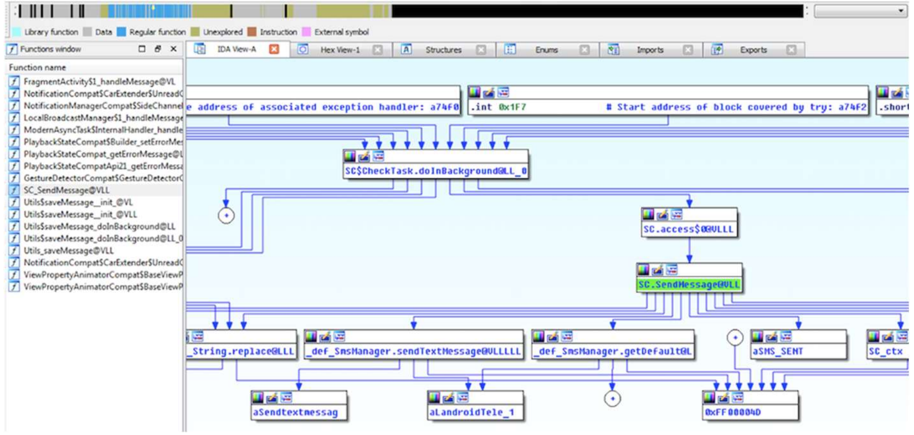
When the malicious application is detected, a further analysis on the phone would be inappropriate. As a rule, the application as an apk file is retrieved, and the browser's logs and download history in the phone are analyzed to search from where such application could be obtained.

Next, the application is analyzing on a desktop computer running Android OS emulator (included in the Android Studio application development tools). Application analysis is complicated because we deal with a compiled application without the presence of the source code.

4.2 Low-level debugging the Android applications

Without the presence of the source code, there is a way to understand the logic of the application – to analyze it with using a debugger. Starting with version 6.6, an interactive debugger IDA Pro [6] supports debugging applications for Dalvik virtual machine.

It is possible to trace the logic of the application, explorer function calls from a Java library, set breakpoints on them and analyze the data transferred as parameters, etc.



```
FragmentActivity$1_handleMessage@VL
NotificationCompat$CarExtender$UnreadC
NotificationManagerCompat$SideChannel
LocalBroadcastManager$1_handleMessage
ModernAsyncTask$InternalHandler_handle
PlaybackStateCompat$Builder_setErrorMes
PlaybackStateCompat_getErrorMessage@L
PlaybackStateCompatApi21_getErrorMess
GestureDetectorCompat$GestureDetectorC
SC_SendMessage@VLL
Utils$saveMessage__init_@VL
Utils$saveMessage__init_@VLL
Utils$saveMessage_doInBackground@LL
Utils$saveMessage_doInBackground@LL_0
Utils_saveMessage@VLL
NotificationCompat$CarExtender$UnreadC
ViewPropertyAnimatorCompat$BaseViewP
ViewPropertyAnimatorCompat$BaseViewP
```

```
e address of associated exception handler: a74F0
.int 0x1F7                # Start address of block covered by try: a74F2
.short
SC$CheckTask.doInBackground@LL_0
SC.access$0@VLLL
SC.SendMessage@VLL
_String.replace@LLL
_def_SmsManager.sendTextMessage@VLLLLL
_def_SmsManager.getDefault@L
aSMS_SENT
SC_ctx
aSendtextmessag
aLandroidTele_1
0xFF00004D
```

Fig 3. Discovering a behaviour graph with IDA

With using IDA we can use a feature "proximity browser" for some kind of static analysis (see fig. 3). Once .dex file is loaded, the app creates a graph containing functions, variables, other objects and the relationships between them. Without actual running the debugging it is possible to find a suspicious function call (like sending the SMS) and view a path for its execution.

With using the debugger we can identify, for example, process of interaction with an intruder's server (in this case, addresses and sending data were not explicitly

programmed in the code but collected in the values of local variables), and this approach was used to prove the harmfulness work with SMS messaging with a remote banking in one trojan application.

However, using this method of analysis requires knowledge of the virtual machine assembler and approaches of low-level debugging, the process of analysis is long and complicated. In addition, IDA Pro is a paid product.

The future research work may be applied here to create own solution to detect calls from application's disassembled code started from some usable functions from Android API liked by the intruders (send and receive SMS, networking, etc) and programmable create a graph to do future analysis.

4.3 Analysis of network interactions in the Android applications

To determine what data is passed over the network to/from the analysing application, we can propose a very simple way – to install a proxy server on a host that is running the phone emulator, and make the settings in the Android emulator phone instance to specify the address and port of the proxy server.

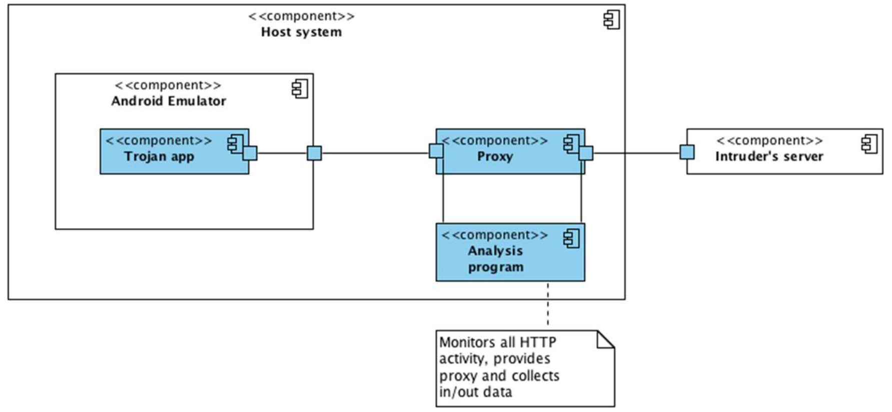


Fig. 4. A proxy server offers to monitor the traffic

After that, all the HTTP connections will go through a proxy, which may log us information is sent and received by the applications on the phone (the process is shown on fig. 4). As a result, we can identify intruder's domains, through which the trojan applications receive data and transmit the control commands.

Having the recorded network data, we also can match various versions of the malware, identifying the groups of trojans by comparing the remote hosts and interactions data.

4.4 Tracing the application's logic with a patched OS kernel

To understand the logic behavior of Android applications, it is decided to use the method of dynamic analysis with logging tools built into Android.

However, the standard "adb logcat" command creates a very sparse dump of the application's operations during the run and not intended for the analysis.

The TaintDroid project [4] was created to tell the user about private data leaks during the work of Android OS applications in the form of push notifications on the top of a phone screen. It patches Android kernel and library code and inserts the own code to reveal and process taint data.

They had to do a lot of work to pass some additional arguments between every library function call in order to collect the data (a cut from the patch for the file vm/InlineNative.cpp is shown on table 2).

Table 2. A fragment of the TaintDroid [4] patch

```
@@ -291,8 +310,13 @@ bool javaLangString_compareTo(u4 arg0, u4 arg1
 /*
  * public boolean equals(Object anObject)
  */
+#ifdef WITH_TAINT_TRACKING
+bool javaLangString_equals(u4 arg0, u4 arg1, u4 arg2, u4 arg3,
+    u4 arg0_taint, u4 arg1_taint, struct Taint* rtaint,
JValue* pResult)
+#else
 bool javaLangString_equals(u4 arg0, u4 arg1, u4 arg2, u4 arg3,
     JValue* pResult)
+#endif /*WITH_TAINT_TRACKING*/
```

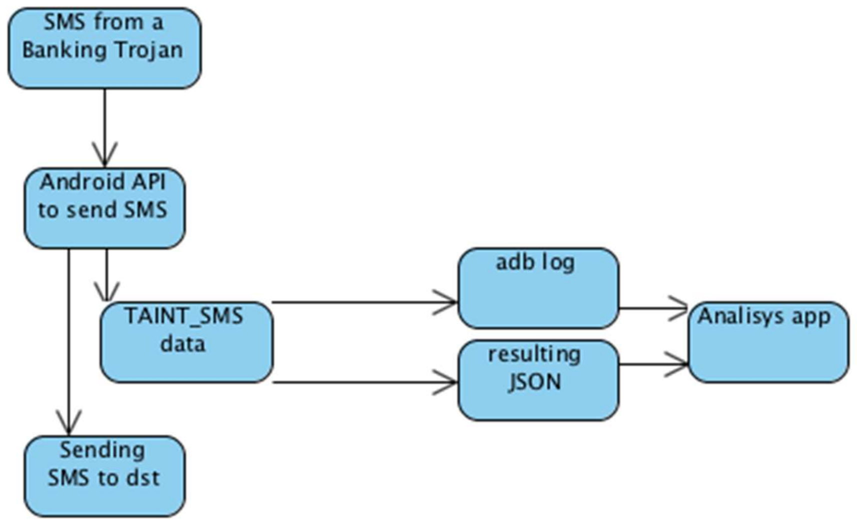


Fig. 5. An example of an SMS flow with using DroidBox

After doing it they have inserted their code to check the fact of opened files, network access, used encryption algorithms and data pass to encrypt/decrypt functions, sending and receiving SMS (a number and a text of the message), phone calls from applications, class loading into memory – just what is necessary for the analysis of malicious code.

Next, we found Droidbox [7] – a framework for the dynamic analysis of applications for Android, which is built on the top of TaintDroid's patches and consists of two parts:

- a Python script to start and produce results;
- a modified OS kernel, containing TaintDroid's patches and additional patches for improving the logging tools.

These patches work as follows: because of some added additional lines to the OS code to preserve TaintDroid data to the log (see fig. 5), information of applications' behavior now can be accessible with 'adb logcat' command and can be processed with some external software.

It should be stressed that some viruses perform checks the OS version of Android, the phone name, phone number, the IMEI, in order to determine the work in the emulator with standard predefined values and possible to stop working after such detection. So, we have to use one's own binary patches to the system image with the replacement of the required parameters with some new values without having to recompile the entire kernel.

4.5 Decompilation and source code analysis

If the log during the time of the analysis of the application shows us no activity (for example, the server that receives a command is not working now), we can try to decompile the application code to observe the logic.

As every Java application could be decompiled to a source code (especially if the application developer did not use any obfuscation algorithms after building the code), we can try to decompile the application and get some semblance of its source code. For such decompilation firstly we apply dex2jar program [8] (to convert from a.dex archive inside of an .apk file to a set of .class files containing compiled Java code), and Java decompiler program (a tool for process .class and obtain .java files from them).

Because of a difficult Java structure, currently there is not exist a good decompiler for all cases, so we propose to use some decompilers simultaneously (FernFlower [9], CPR [10], Jadx [11], Procyon [12]), and to select the best result after decompiling.

The results of decompiling to a source code can be used against intruders to prove the harmfulness of the application.

However, if the application file structure is just restored (though not their contents), then it can also be used for comparing the trojans from the various crime groups.

The process of restoring normal application code and understanding what it does – is a handwork, but not a complicated enough. The issue of combating obfuscation requires special studies. For example, this is a fragment of decompiled code that possible had been obfuscated before distributed, it is unreadable and currently the method of source code observing is not used for such files:

Table 3. A fragment of decompiled obfuscated code

```
var1_5 = var9_1[0]
var9_1 = Class.forName(Application.onCreate("\ub559\u4623
\u069b\u920c\u5140\u162b\u13b7\u927b\uff0c\ubf32\u1c91
\u4ae2"))
var12_9 = var9_1.getConstructor(new Class[]{var9_1, String.class})
var10_11 = Class.forName(Application.onCreate("\ub552
\u462c\u0689\u921f\u5101\u162b\u13bc\u927b\uff29\ubf34
\u1c93\u4af3\u4a06\u3b4a\u4b63\u895b\u8aa6\u19b8\u62da
\uecc1\u2174\u912e\uc452"))
var14_12 = var10_11.getMethod(Application.onCreate
("\ub554\u4627\u0699\u9229\u5107\u1630"), new
Class[]{String.class, Integer.TYPE})
```

4.6 The general method

The proposed general method is given on fig. 6. Suppose we have a phone with some possible trojans. We look at the applications, get them from the phone and select them to make the analysis by some indirect indications (like name, icon, size, permissions). And possible we try to check some applications in the antivirus databases.

If we found a suspicious application, we are going to make a comprehensive analysis. The app in binary classes form could be checked by a static analyzer for possible security vulnerabilities (currently the static analyzer has not been selected yet). After, the app goes to the decompilation, and if the result of it is good the code behavior can be observed from the sources.

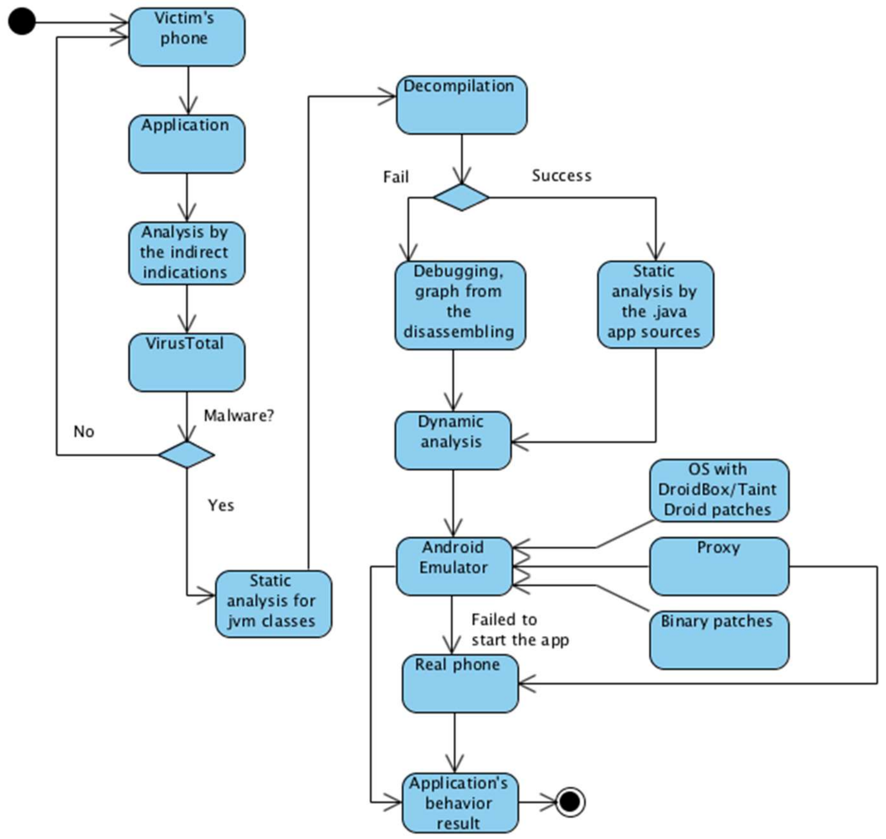

Victim's phone
Application
Analysis by the indirect indications
VirusTotal
Malware?
No
Yes
Static analysis for jvm classes
Decompilation
Fail
Success
Debugging, graph from the disassembling
Static analysis by the .java app sources
Dynamic analysis
OS with DroidBox/Taint Droid patches
Proxy
Binary patches
Android Emulator
Failed to start the app
Real phone
Application's behavior result

Fig 6. Proposed process of analysis.

In addition, sources could be statically analyzed by the static analyzer, which works with .java sources. If the decompilation has failed, the app goes to the debugger and the behavior graph, which is received after disassembling is observing.

For analysis the dynamic behavior we use the emulator with a modified OS kernel for logging, it is patched to change the IMEI, phone and so on, and proxy to detect a network activity. Some applications cannot be run from the emulator for various reasons, they could be run in the real phone with using the proxy monitoring.

After finishing the process, we will have some data about the application's behavior. The completeness of the resulting data depends on the application but in the almost all cases of analysis that we did, the information about the application is sufficed to threat is it a malware or not.

4.7 Developing an application based on proposed methods of the analysis

Our students have developed [13] a small aggregated single-window application that performs semi-automatically the functions described here, without having to manually run all the utilities, copying files, and so on. It can be used by a specialist of digital forensic science. The application communicates with the Android Platform Tools for managing connected phone, the emulator and examines the logs.

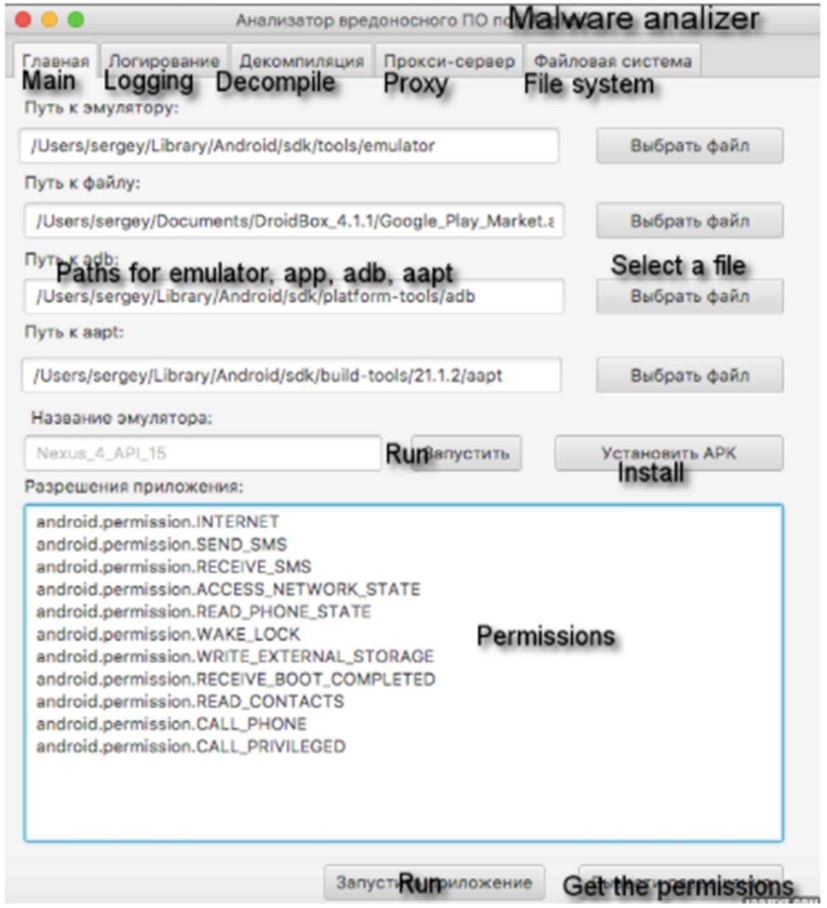
The app allows (see fig. 7):

- getting the permissions and copy the application from an Android phone. y malicious applications among given applications from given phones;
- make requests to the VirusTotal to check the application in the antivirus databases;
- start the Android emulator with the patched kernel and system image, make the dynamic analysis of the application, display the collected logs;
- decompile the application by the described decompilers and navigate through the decompiled files;
- run the integrated proxy server and monitor application's networking activity.
- Later in this application, we plan to generate reports in the standard form for digital forensic science.

5. Results and conclusions

- The article described the basic methods of analysis the behavior of malicious applications for Android. The general analysis process is proposed. Currently, the results are actually being used to make proofs of committing computer crimes (it is a research work by the contract).
- An extension of this work will be a higher automation of the analysis as well as analysis of the applications that have sophisticated protection, some

strong research work for call graph building from the disassembled and obfuscated code, providing patches for modern Android kernels.



Путь к эмулятору: /Users/sergey/Library/Android/sdk/tools/emulator

Путь к файлу: /Users/sergey/Documents/DroidBox_4.1.1/Google_Play_Market.a

Путь к adb: /Users/sergey/Library/Android/sdk/platform-tools/adb

Путь к aapt: /Users/sergey/Library/Android/sdk/build-tools/21.1.2/aapt

Название эмулятора: Nexus_4_API_15

Разрешения приложения:

```
android.permission.INTERNET
android.permission.SEND_SMS
android.permission.RECEIVE_SMS
android.permission.ACCESS_NETWORK_STATE
android.permission.READ_PHONE_STATE
android.permission.WAKE_LOCK
android.permission.WRITE_EXTERNAL_STORAGE
android.permission.RECEIVE_BOOT_COMPLETED
android.permission.READ_CONTACTS
android.permission.CALL_PHONE
android.permission.CALL_PRIVILEGED
```

Fig. 7. The helping application has been developed for simplify the analysis

References

[1]. Emarketer.com: 2 Billion Consumers Worldwide to Get Smart(phones) by 2016. Available under the link: https://www.emarketer.com/Article/2-Billion-Consumers-Worldwide-Smartphones-by-2016/1011694. 03.03.2018.

[2]. Vedomosti.ru. Hackers have stolen from Android owners 349 million rubles for four quarters. Available under the link: https://www.vedomosti.ru/technology/articles/2016/10/13/660728-hakeri-ukrali-android. 03.03.2018 (in Russian)

[3]. Arp D. et al. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. NDSS, 2014, vol. 14, pp. 23-26

[4]. Enck W. et al. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. ACM Transactions on Computer Systems (TOCS), 2014, vol. 32, №. 2, pp. 5.
[5]. VirusTotal – Free Online Virus, Malware and URL Scanner. Available under the link: https://www.virustotal.com. 03.03.2018
[6]. Debugging Dalvik programs with IDA. Hex-Rays. Available under the link: https://www.hex-rays.com/products/ida/support/tutorials/debugging_dalvik.pdf. 03.03.2018
[7]. pjlantz/droidbox: Dynamic analysis of Android apps. Available under the link: https://github.com/pjlantz/droidbox. 03.03.2018
[8]. dex2jar. Tools to work with android .dex and java .class files. Available under the link: https://sourceforge.net/projects/dex2jar. 03.03.2018
[9]. Fernflower is the first actually working analytical decompiler for Java. Available under the link: https://github.com/JetBrains/intellij-community/tree/master/plugins/java-decompiler/engine. 03.03.2018
[10]. CFR – another java decompiler. Available under the link: http://www.benf.org/other/cfr/. 03.03.2018
[11]. jadx – Dex to Java decompiler. Command line and GUI tools for produce Java source code from Android Dex and Apk files. Available under the link: https://github.com/skylot/jadx. 03.03.2018
[12]. Procyon/Java Decompiler. Available under the link: https://bitbucket.org/mstrobel/procyon/wiki/Java%20Decompiler. 03.03.2018
[13]. Abalmasov A.V., Staroletov S.M. Development of a malware analysis system for the Android platform. Bachelor's work. AltaiSTU, 2016. Available under the link: http://new.elib.altstu.ru/diploma/download_vkr/id/70003. 03.03.2018 (in Russian).

Методы анализа вредоносного программного обеспечения под ОС Android

С.М. Старолетов <serg_soft@mail.ru>
Алтайский государственный технический университет им. И.И. Ползунова, 656038 Барнаул, проспект Ленина, 46

Аннотация. В статье рассматривается проблема анализа приложений под ОС Андроид с целью выявления вредоносного поведения. Представлены методы анализа такого рода приложений, которые применялись при проведении реальных программно- и научно-технических экспертиз (статические, динамические проверки, отладка, декомпиляция, логирование). Описаны процесс анализа, существующие приложения, а также собственное программное обеспечение.

Ключевые слова: анализ поведения программ; вредоносное программное обеспечение; утечки данных; Java; программно-техническая экспертиза

DOI: 10.15514/ISPRAS-2018-30(1)-4

Для цитирования: Старолетов С.М. Методы анализа вредоносного программного обеспечения под ОС Android. Труды ИСП РАН, том 30, вып. 1, 2018 г., стр. 55-68. DOI: 10.15514/ISPRAS-2018-30(1)-4

Список литературы

[1]. Emarketer.com: 2 Billion Consumers Worldwide to Get Smart(phones) by 2016. Available under the link: https://www.emarketer.com/Article/2-Billion-Consumers-Worldwide-Smartphones-by-2016/1011694. 03.03.2018.
[2]. Vedomosti.ru. Хакеры украли у владельцев Android 349 млн рублей за четыре квартала. Доступно по ссылке: https://www.vedomosti.ru/technology/articles/2016/10/13/660728-hakeri-ukrali-android. 03.03.2018
[3]. Arp D. et al. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. NDSS, 2014, vol. 14, pp. 23-26
[4]. Enck W. et al. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. ACM Transactions on Computer Systems (TOCS), 2014, vol. 32, №. 2, pp. 5.
[5]. VirusTotal - Free Online Virus, Malware and URL Scanner. Available under the link: https://www.virustotal.com. 03.03.2018
[6]. Debugging Dalvik programs with IDA. Hex-Rays. Available under the link: https://www.hex-rays.com/products/ida/support/tutorials/debugging_dalvik.pdf. 03.03.2018
[7]. pjlantz/droidbox: Dynamic analysis of Android apps. Available under the link: https://github.com/pjlantz/droidbox. 03.03.2018
[8]. dex2jar. Tools to work with android .dex and java .class files. Available under the link: https://sourceforge.net/projects/dex2jar. 03.03.2018
[9]. Fernflower is the first actually working analytical decompiler for Java. Available under the link: https://github.com/JetBrains/intellij-community/tree/master/plugins/java-decompiler/engine. 03.03.2018
[10]. CFR – another java decompiler. Available under the link: http://www.benf.org/other/cfr/. 03.03.2018
[11]. jadx – Dex to Java decompiler. Command line and GUI tools for produce Java source code from Android Dex and Apk files. Available under the link: https://github.com/skylot/jadx. 03.03.2018
[12]. Procyon/Java Decompiler. Available under the link: https://bitbucket.org/mstrobel/procyon/wiki/Java%20Decompiler. 03.03.2018
[13]. Абалмасов А.В., Старолетов С.М. Разработка системы анализа вредоносного ПО на платформе Android. Бакалаврская работа. АлтГТУ, 2016. Доступно по ссылке: http://new.elib.altstu.ru/diploma/download_vkr/id/70003. 03.03.2018