Testing Laboratory for Mobile Applications and Malware Analysis

Andrii Shalaginov
andrii.shalaginov@ntnu.no

June 3, 2020

Abstract

Project will cover architecture and principles of Mobile Applications automated analysis which will be used in building Testing Laboratory. The need for development of such a laboratory is caused by increasing usage of mobile devices instead of general personal computers for business and leisure purposes. The target application domain also contain Mobile Malware and Mobile Software from third-party producers with possible embedded malicious actions besides general functionality. Therefore, security measurement for mobile applications in more broader context is required such that attack types estimation, mobile forensics and machine learning analysis for phishing detection. It could be highlighted the following features, which will be useful in Testing Laboratory:

- Mobile Malwares and Malicious Software analysis in created environment (mobile phones as zombies, pirate applications stores as malware distributions, mobile botnets, etc).
- Dynamic and Static test phases for appropriate data collection.
- Extensive configuration of testing environment.
- Machine Learning as a decision making engine for estimation likelihood of threats.

Keywords: Mobile Malware, Malicious Program Analysis, Machine Learning for Threats Detection, Automated Analysis, Testing Laboratory

Contents

1 Introduction **2**

2 Background **3**
2.1 Problem Area Definition . . . 3
2.2 User Applications in Android Operating System (OS) . . . 3
2.2.1 Application Installation Package . . . 4
2.2.2 Fundamentals of Android Application Design . . . 5

3 Testing Laboratory Framework Overview **7**
3.1 Requirement Specification . . . 7
3.2 Functional Requirements . . . 7
3.2.1 Application Samples Processing . . . 7
3.2.2 Emulator Set-Up & Customisation . . . 8
3.2.3 Test Cycle . . . 8
3.2.4 Analysis of Collected Data . . . 9
3.3 Non-Functional Requirements . . . 9
3.4 Functional Diagram . . . 10
3.5 Use Case Diagram . . . 10

4 Basic Blocks Design **12**
4.1 Ergonomics & Design . . . 13
4.2 Application Samples Processing . . . 14
4.3 Emulator Set-Up & Customisation . . . 15
4.4 Test Cycle . . . 16
4.5 Analysis of the Collected Data . . . 18

5 Results and Analysis **21**
5.1 Prerequisites for Malware Success . . . 21
5.2 Performed Analysis . . . 21

6 Existing Solutions & Further Work **23**

7 Conclusions **24**

References **25**

Acronyms **27**

A APPENDIX **28**
A.1 System Designing . . . 28
A.2 Results and Practical Aspects . . . 31

1 Introduction

The main idea of Testing Laboratory for mobile malware automated analysis is to provide user with flexible graphic interface and appropriate functionality for assessing information security of the application based on collected static and dynamic test data and made machine learning decisions and evaluations [24]. Such information could be used for building assessment model of Confidentiality, Integrity, Availability of user's sensitive data and impact on financial resources of a user.

The project includes Analysis, Design and Implementation stages of Testing Laboratory. Each stage will be described in the report below. The Laboratory is aimed on the user with basic or middle awareness level about Mobile Android Platform and possible malicious actions. The system has single role, which aimed on forensics analyst without any need of specific administration and maintainance.

This project report has structure as following. The Section 2 gives description of different aspects of Android Applications such that installation and execution processes. The Section 3 provides functional and non-functional requirements to the Testing Laboratory to be designed and implemented. The practical description of implementation and design details are presented in the Section 4. Practical usage results and evaluation are shown in the Section 5. Section 6 sketch the available solutions in the domain and view of authors on future research. Conclusions are presented in the Section 7.

2 Background

Mobile technologies were developed significantly in recent decade and have grown from simple phone functionality to fully operated smart computers. With growing functionality and opportunities to store much sensitive information, which could be stolen or altered by 3rd party without user acknowledgement. This section provides general overview of the Android Platform, ways to install applications, their structure and how the data is operated on it as well.

2.1 Problem Area Definition

Recently, the amount of mobile devices became equal to amount of shipped Personal Computer, which indicates increasing popularity and capability of smart-phones and tables. According to research [1], vendors produced 488 millions mobile devices vs 411 Personal Computers in 2011.

The problem, which this project intended to solve and to build corresponding theoretical and practical grounding, can be defined as following "Building testing Laboratory for automated analysis of malicious mobile software".

According to the report [2], most of the Anti-Virus protection programs for Personal Computers have generally more than 90% detection rate, which means that the security issues for PC were significantly studied during past few decades of modern OS existence. Despite this fact, the Anti-Virus software for mobile devices is not so sophisticated. Basically only about 25% of all studied existing protection software show detection rate more then 90% as it is stated in the research [3]. This is because the Mobile OS are relatively new and did not studied quite comprehensively in domain of all possible vulnerabilities, bugs and coding errors. Mostly, the mobile malicious software inspection requires manual processing by specialist and has many obstacles in case of automatic processing.

The vital part in security threats within mobile devices plays permission-based access control, which can be granted by explicitly requesting user about that. Through this procedure, the user can accidentally provide wrong and risky permission to the application that requests it. It is generally hard to determine whether an application is needed specific permission indeed or not without human interaction. Nevertheless, it is possible to make machine learning assumption about need for use particular permission based on semantic nature of a functionality and an application (e.g., a calculator probably will not use access to bank account information).

Existing mobile anti-virus solutions can not provide sufficient analysis due to restricted available computational resources and architectural aspects of the Mobile Platform. Therefore, there is a need for creating of Testing Laboratory aimed on versatile testing and analysis of applications.

2.2 User Applications in Android OS

Android Operation System has own flow of installing applications. They could be downloaded from trusted sources such that Android Market or used from third-party developers. The last fact provides a lot of possibilities for attackers to intrude and exploit applications before they will be installed on the victim mobile devices.

2.2.1 Application Installation Package

Each application installer on Android system is represented as an Android Application Package, container that stores all necessary information for application installation (APK) file that contains compressed components, necessary for installation process. Generally it has following structure [4, 5]:

- *lib* - libraries and compiled code for various processor architecture (recently: armeabi [6], armeabi-v7a, x86, mips).
- *res* - supplementary data such that media, preferences, etc.
- ***AndroidManifest.xml*** - information about application name, version, required permissions, layout, dependencies, etc.
- *classes.dex* - executable code for Dalvik Virtual Machine
- *resources.arsc* - binary file after compilation.
- *assets* - folder with manual, licence, help, etc.
- *META-INF* - folders that contain certificates and other security options in order to prove that application is original.

The Android Application Development process is depicted on the Figure 1. It can be figured out that and application might be considered as a legitimate and original in case if it is signed by original developer key, which is agreed officially with issuing market. Otherwise, unknown or tampered keys can be used during installation of the application, which will lead to unobstructed setup of any third-party program (majority of users allow this option in mobile device settings).

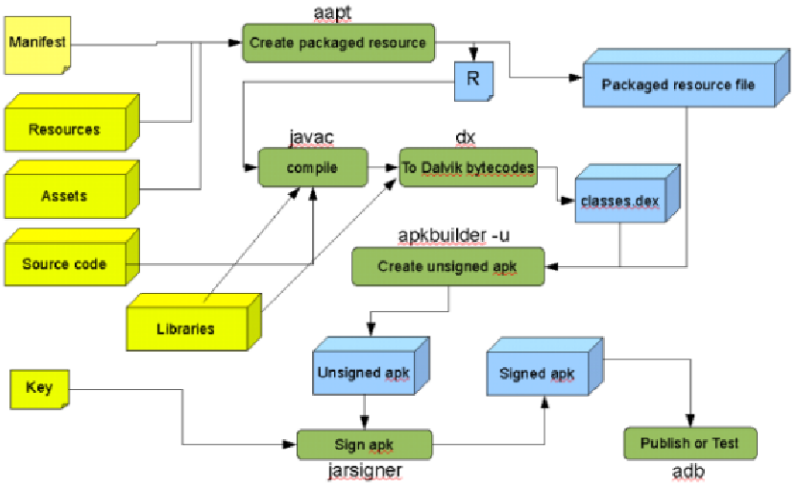


Figure 1: APK development flow [7]

One component that should be examined first in the APK package is AndroidManifest.xml file, which contains important information and may provide picture of application intentions and functionality. Basically, this file contains following main options [5]:

- <uses-permission> - application required permissions.
- <permission> - permissions to the applications that other applications could have relatively this application.

- <application> - application level description (used themes, layouts, etc), which may contain following sections:
  - <intent> - system for messaging about system notifications and navigation between various hardware states.
  - <activity> - component of the application that will interact with user by means of windows or other interfaces. There could be more than one activity and they provide responding to user/system activity.
  - <receiver> - possibility for the application to get broadcast events without even launching.
  - <service> - arbitrary process that will run in background and does not provide user interface.
  - <provider> - control access from other applications to the application's data.

2.2.2 Fundamentals of Android Application Design

Each application in Android OS is running within its own environment as it is depicted on the Figure 2. Applications can not access directly into each other environments, but can use private and shared storage on the non-volatile memory for this purpose [8].

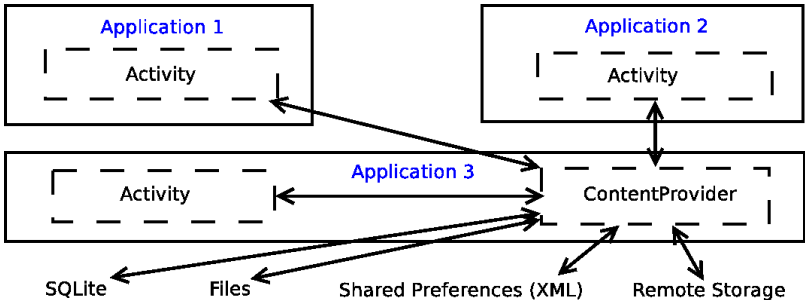


Figure 2: Building Blocks of Android Application [9]

As it was mentioned previously, AndroidManifest.xml file contains basic description of the application's components. Understanding of each component is quite important for future analysis of application's intentions and functionality. Also a general application has three main abstraction blocks in running phase as it can be seen on the Figure 3: activities, tasks and processes.

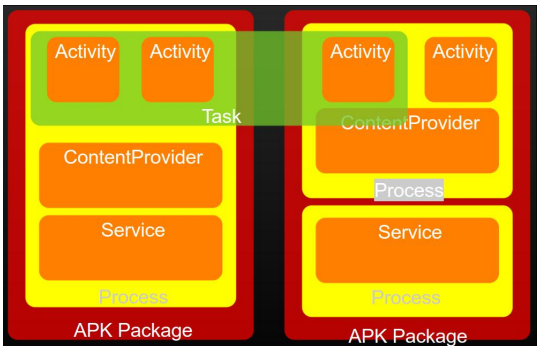


Figure 3: Inside running Android Application [10]

Application data are stored generally in a folder */data/data/<appName>/* and can be accessible not only for considering application, but also to the others. There are five methods of storing data in Android Architecture for android application [8]:

1. Shared preferences.

   Simple XML format that store application data.

2. Internal storage.

   May contain various data types stored in different file formats, mounted as a */data/data/<appName>/app_* and */data/data/<appName>/files* directories.

3. External storage.

   Files stored on Secure Digital Memory Card (SD) Card mounted as a */mnt/sdcard*.

4. SQLite.

   Provides connector to Data Base (DB) and store them into */data/data/<appName>/databases*.

5. Network.

   Remote storage such that DropBox.

Possibilities for accessing another application data are shown on the Figure 4 and depend only on provided permissions for the application during installation phase.

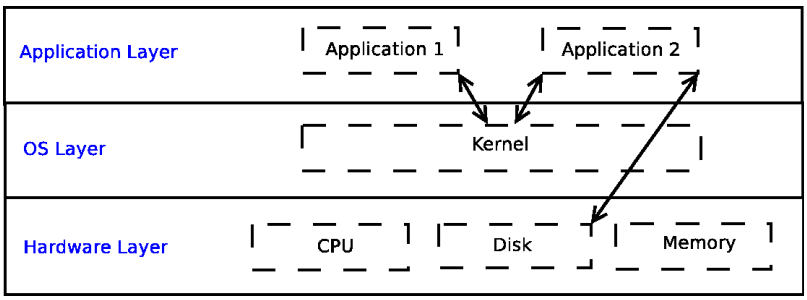


Figure 4: Basic layers in Android architecture [11]

3 Testing Laboratory Framework Overview

The purpose of the Testing Laboratory is to build environment for processing and testing mobile application for Android Operation System. It should provide comprehensive static and dynamic analysis for each application and retrieve all possible information for further analysis. Finally, the Laboratory will be targeted on detection and identification of malicious software according to predefined taxonomy of possible harmful actions.

3.1 Requirement Specification

The Laboratory shall agree with following requirements, which are specified during analysis and planning phase. The concept of the Testing Laboratory is based on the computational forensics process [**?**], which is depicted on the Figure 5. This process is targeted on black and grey boxes testing and analysing of the malware samples.

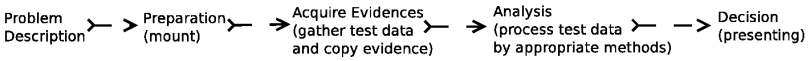


Figure 5: Digital Forensics process [12]

3.2 Functional Requirements

This Section includes Requirements to the Functionality, which considered Laboratory should satisfy. The Testing Laboratory is presented as a System and splited into Subsystems for getting more flexible structure.

[F1]: The next sub-system must be implemented:

- Uploading and Static processing of the Application Samples
- Emulator Set-Up & Customisation
- Test Cycle in Dynamic virtual environment
- Analysis of Collected Data by means of machine learning and pattern recognition

[F2]: Each implemented sub-system must include:

- Basic overview and description
- Resources consuming estimation (execution time, memory, etc.)
- Required dependencies (shared libraries, paths, installed software)
- Input / Output specification

3.2.1 Application Samples Processing

The sub-system will be used for partial statical analysis of application packages. Additionally, it shall have Graphic User Interface implemented as a web-page and shall represent understandable control elements.

[A1] Sub-system Input:

- uploading of *.apk files in suitable manner through web script,
- might be a possibility to upload bunch of the applications.

[A2] Sub-system Output:

- web-page with extensive application description (original package name, version, minimal Android API version, require permission),
- status of application package processing (success|fail|wrong structure),
- all information shall be stored in DB in format that is appropriate for further analysis and usage. Application should be given and unique indentificator for further usage.

3.2.2 Emulator Set-Up & Customisation

Emulator Set-Up & Customisation must provide interface for user in order to install, select existing, create new and control virtual machine in installed Android Software Development Kit, bunch of tools for developing, testing and building applications (SDK) environment. The User-System interaction should be performed over web-pages. Additionally, there should be possibility to choose one of the few ***user profiles*** for virtual machine.

[A3] Sub-system Input:

- creating Android Virtual Device, also called Virtual Machine based on Android Operation System image (AVD) based on available and installed android Platform Targets,
- user input and its validation (text fields, checkboxes, etc),
- predefined path to Android SDK installation in order to be able to check configuration and existing virtual machine.

[A4] Sub-system Output:

- general information about installed Android SDK, available Platform Targets and created AVDs
- status of user query to the sub-system (success|fail).
- configuration overview of each particular virtual machines must be stored into DB and it must be given an unique virtual machine identifier.

3.2.3 Test Cycle

The sub-system must be able to perform test cycle aver selected application(s). Each transaction may contain single or multiple options to be tested on the targeted application (up to user what to test). The raw results for each tested application must be be stored.

[A5] Sub-system Input:

- choosing predefined virtual machine configuration,
- choosing one or multiple applications to be analysed from list of uploaded,
- choosing type and sequence of the test cycle (static, dynamic, particular areas to test, etc).

[A6] Sub-system Output:

- status of the performed test cycle transaction (success|fail),

- general information about collected data for each particular test cycle's option (duration, errors, logs, etc),
- raw information about tests is stored in appropriate format into DB from test cycle for each application for further analysis.

3.2.4 Analysis of Collected Data

Analysis of Collected Data must be presented as a collection of possible analysis approaches over previously collected data such that string search in memory dumps and files, permission analysis, stored application's data, etc.

[A7] Sub-system Input:

- selection among a set of possible approaches for particular uploaded and processed application application,
- selection one or multiple applications to be analysed.

[A8] Sub-system Output:

- detailed report about performed analysis and found artifacts,
- screenshots that were obtained,
- stored by the application data structure,
- resources usage,
- estimated threats from application and its classification,
- possibilities of correlation between found data pieces,
- the date from analysis should be stored in DB and have a possibilities to be exported from the system.

3.3 Non-Functional Requirements

The Laboratory should agree with additional requirements like resource consumption, licences, coding standard, design and operational guidelines.

[NF1]: Implemented system should preserve modularity among sub-systems.

[NF2]: No licence issues, Android SDK is available as an open source software.

[NF3]: Implemented source code must follow coding convention.

[NF4]: Exceptions inside the system should be documented and should prevent any possible memory leakage and exceeding reasonable execution time.

[NF5]: For proper operation, the environment should work on an open-source software.

[NF6]: Execution time for Application Samples Processing (for single application) and Emulator Set-Up & Customisation sub-systems should not exceed 1 minute.

[NF7]: Execution time for Test Cycle and Analysis of Collected Data sub-systems should not exceed 10 minutes for processing each selected application (including about 5 minutes for running Virtual Machine from the scratch).

[NF8]: Memory consumption of the system must be lower then 2GB.

[NF8]: Disk Storage consumption of the system for each processed application must

not exceed 100MB.

[NF9]: The system must be adaptable to changing environment - detect any deviations in environment configuration from minimal required and warn a user.

3.4 Functional Diagram

The Figure 6 represents functionality of the Teesting Laboratory. I consists of four basic blocks and finally provide a report for the analysis phase.

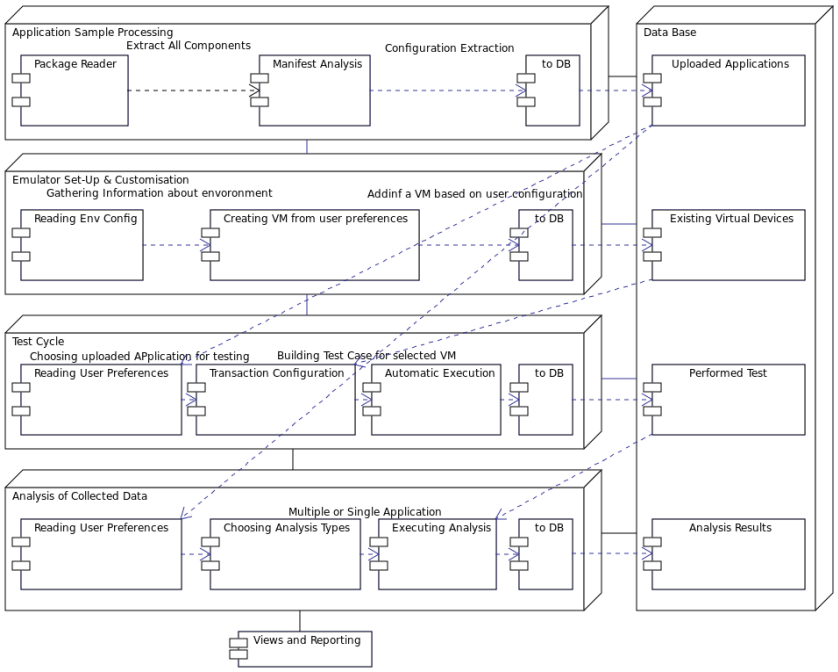


Figure 6: Functional Flow Diagram for the Testing Laboratory

3.5 Use Case Diagram

The Laboratory will have a single user, which is able to execute particular set of actions within each of our sub-systems. This is depicted on the Figure 7.

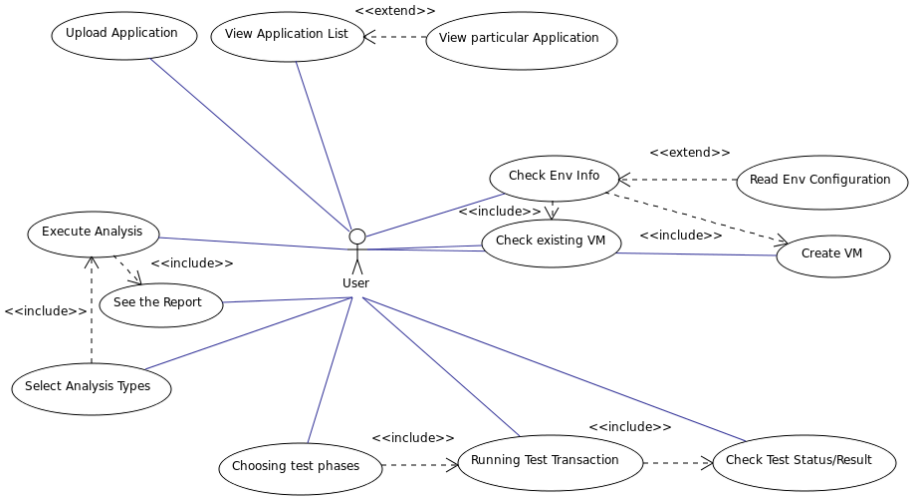


Figure 7: Use Case Diagram for the Testing Laboratory

4 Basic Blocks Design

The Testing Laboratory design consists of two parts: design of Graphic User Interface (GUI) and design of functionality and algorithm implementation. It was chosen to use Top-Down software design model for developing Testing Laboratory in order to be able to build up functionality blocks in future. The chosen design was successfully used in previous works and demonstrated that separation of functionality in system architecture allows to develop flexible and plugable system with independent sub-systems.

For GUI designing, a WEB platform was chosen that represents itself browser interface rather than the native windows-based GUI. From globalisation, usability, cross-platform usage and interoperability points of view, it is more applicable for future development and does not require any advanced software or hardware options on client side [13]. The Testing Laboratory is based on Model View Controller architecture [14], which allows to build flexible application for processing complex data, perform naive client-server communication, separate functionality and view of the system and design understandable interface. The system consists of the following logical parts:

1. **User.** Represents a human that sends requests to the system and retrieve desirable output and results.
2. **View.** Set of templates, which are independent of functionality and provides functionality for displaying information and admission of control commands. It is a static web-page that contain areas for representing changeable information in human-understandable manner. According to [15], the View includes following Widgets: restricted Window browser preview, Text Boxes, Buttons, Hyperlinks, Check and Radio Boxes, Data Tables and interactive Asynchronous Javascript and XML (AJAX) areas for autonomous retrieving information from the server side. Also for more convenient navigation, the View includes separated Tabs for each of 4 Blocks.
3. **Controller.** Physically is a part of View for sending sequences of control commands and user-defined data.
4. **Model.** Functionality Block that can change state based on User input and present some output based on stored information.

The main web page of the Testing Laboratory is presented on the Figure 8, which has four tabs for each sub-system respectively.

Core concept of functionality design includes building 4 sub-systems in a common shell for interaction with external programs and DB. Each sub-system has own class and objective-oriented design, can work autonomously and has own table in DB in order to store parameters and various results of processing. For successful operation, there should be installed Android SDK as a main external tool for performing tests and extracting necessary data. The functionality is implemented as php scripts that allows significant flexibility in performing various actions and data processing. As an important prerequisites for successful work, there might be the same owner (user,group) of Apache Web Server and Android SDK platform for excluding restriction in tools execution. The Testing Laboratory requires from user minimal knowledge about system performance and internal structure. All that is required for proper operation is to change paths and other variables in the first section of *config.inc.php* file.

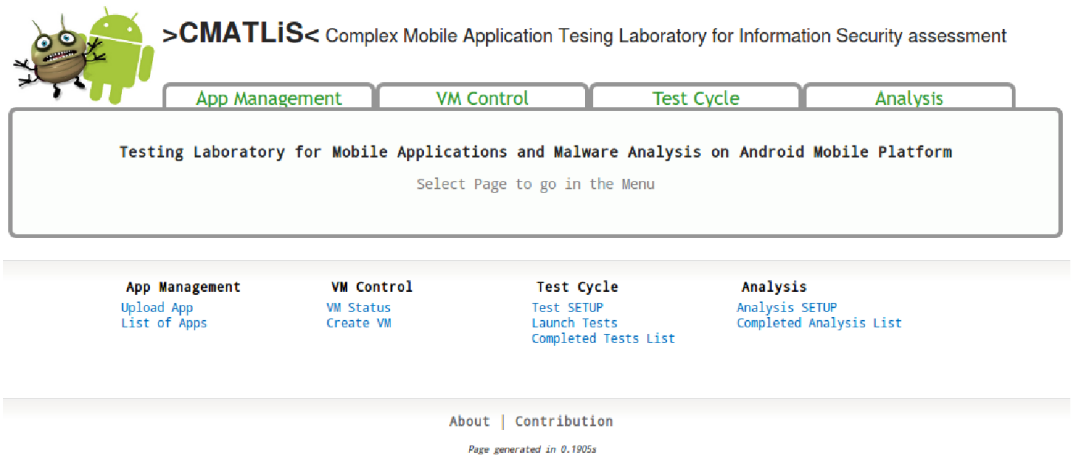


Figure 8: The main index page of the Testing Laboratory

The basic requirements for the Testing Laboratory could be as following: Linux Platform (kernel version =>3.0), Android SDK (revision =>21), Apache Web Server (version => 2.2), PHP (version => 5.3), MySQL (version => 5.1) and jQuery library (version => 1.8). Additionally, there must be installed Android SDK Platform Package (version => 1.5, but requires higher versions for some applications), SQLite3 (+support for PHP), tool *'tree'* for Linux.

According to [16], the software product can be measured by means of Lines of Code (LOC) or Kilo Lines of Code (KLOC). The Testing Laboratory was measured with help of Count Lines of Code program [17] and the results are presented on the Figure 9. In summary, the developed software product has 2386 effective LOC and 454 comment LOC. Based on the commentaries, there was generated Doxygen Documentation [18].

```
http://cloc.sourceforge.net v 1.53  T=0.5 s (36.0 files/s, 6630.0 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
PHP                              9            247            493           1240
HTML                             7             76             10            774
CSS                              1             52              8            301
Javascript                       1              9             34             71
-------------------------------------------------------------------------------
SUM:                            18            384            545           2386
-------------------------------------------------------------------------------
```

Figure 9: Output from the CLOC [17] program

4.1 Ergonomics & Design

Graphic User Interface was tested by several Computer Science students and, as result, chosen implementation methodology and ergonomic design has lived up itself. The Testing Laboratory has understandable interface, structure of content and reliable functionality. A user of the Laboratory, nevertheless, should have basic knowledge about Mobile Platform, Digital Forensics process and malware aspects for successful operation and testing of the application. Designed system is targeted on Forensics Specialists or Master Students of Information Security with forensics background knowledge.

4.2 Application Samples Processing

This subsystem is designed for collecting, extracting entire structure of applications and gathering statistic information from applications to be analysed. It has own table for storing extensive description and obtained information for each processing uploaded application as it is shown in the Appendix A (Figure A.1.2). The uniqueness of application is determined by *'md5sum'* of the installation package and therefore recorded in the DB table's as unique index. If the application is to be deleted, then the data will be erase as well.

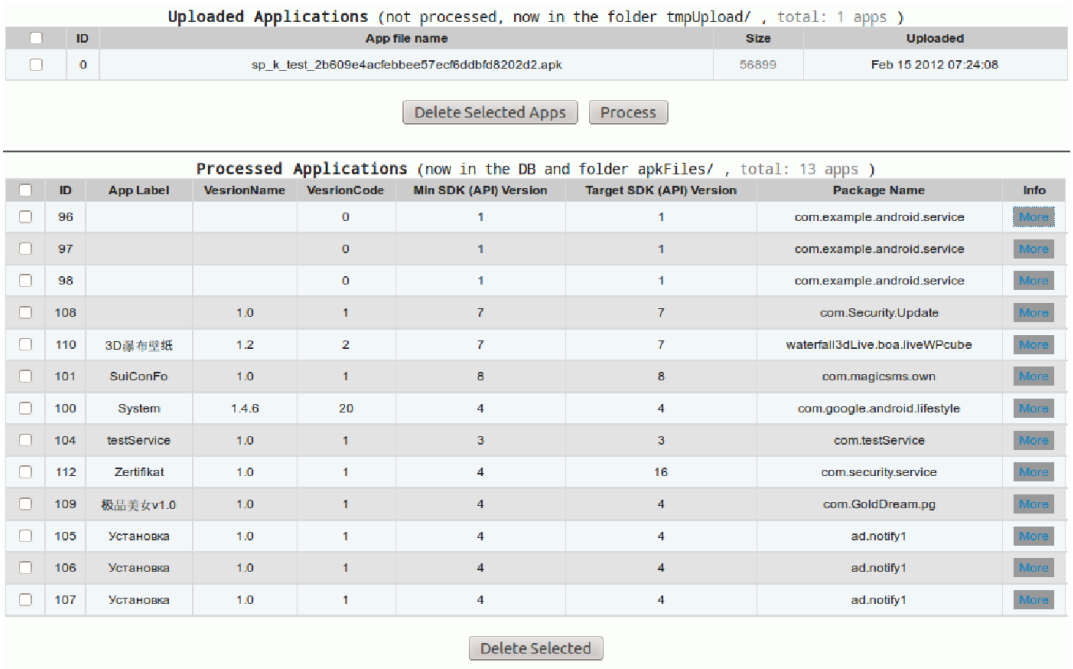
There are two possibilities to upload application in the Testing Laboratory:

- single App through user interface as it is shown in the Appendix A (Figure A.1.1),
- user can put the bunch of applications (APK files) into directory, defined by the variable *'tmp_folder'* in the *config.inc.php* file.

There can be distinguished two states of application in this subsystem, as it is shown on the Figure 10:

- **Uploaded**. Application(s) have just uploaded in the temporary folder on the server and are displayed in the corresponding section,
- **Processed**. Uploaded applications were analysed by statistical examiner function, information about them is stored in the DB and they are move to the permanent storage,

On the processing stage, the application is verified to be completed Android installation package, corresponding settings are extracted and is checked if it was not already uploaded.



Uploaded Applications (not processed, now in the folder tmpUpload/ , total: 1 apps)

	ID	App file name	Size	Uploaded
☐	0	sp_k_test_2b609e4acfebbee57ecf6ddbfd8202d2.apk	56899	Feb 15 2012 07:24:08

Delete Selected Apps | Process

Processed Applications (now in the DB and folder apkFiles/ , total: 13 apps)

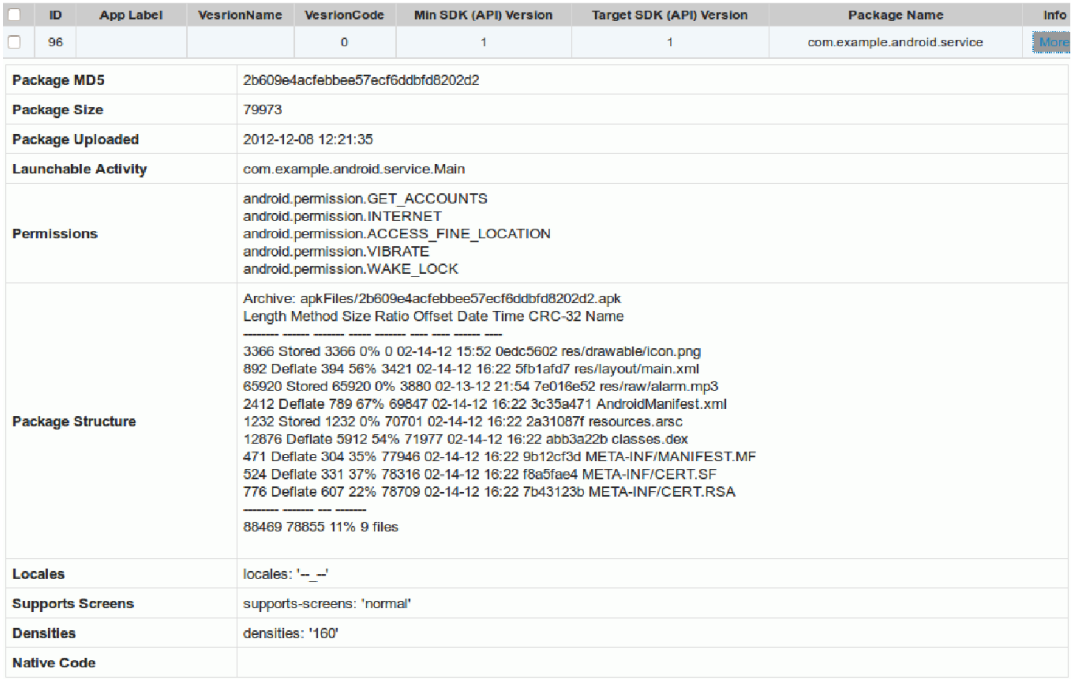
	ID	App Label	VesrionName	VesrionCode	Min SDK (API) Version	Target SDK (API) Version	Package Name	Info
☐	96			0	1	1	com.example.android.service	More
☐	97			0	1	1	com.example.android.service	More
☐	98			0	1	1	com.example.android.service	More
☐	108		1.0	1	7	7	com.Security.Update	More
☐	110	3D瀑布壁纸	1.2	2	7	7	waterfall3dLive.boa.liveWPcube	More
☐	101	SuiConFo	1.0	1	8	8	com.magicsms.own	More
☐	100	System	1.4.6	20	4	4	com.google.android.lifestyle	More
☐	104	testService	1.0	1	3	3	com.testService	More
☐	112	Zertifikat	1.0	1	4	16	com.security.service	More
☐	109	极品美女v1.0	1.0	1	4	4	com.GoldDream.pg	More
☐	105	Установка	1.0	1	4	4	ad.notify1	More
☐	106	Установка	1.0	1	4	4	ad.notify1	More
☐	107	Установка	1.0	1	4	4	ad.notify1	More

Delete Selected

Figure 10: List of uploaded application (which could be processed) and already processed applications stored in the DB

On the Figure 11, the complete and comprehensive list of extracted parameters of each

particular application are presented. It can give tentative picture of application package structure and functionality to be executed.



ID	App Label	VesrionName	VesrionCode	Min SDK (API) Version	Target SDK (API) Version	Package Name	Info
96			0	1	1	com.example.android.service	More

Package MD5	2b609e4acfebbee57ecf6ddbfd8202d2
Package Size	79973
Package Uploaded	2012-12-08 12:21:35
Launchable Activity	com.example.android.service.Main
Permissions	android.permission.GET_ACCOUNTS android.permission.INTERNET android.permission.ACCESS_FINE_LOCATION android.permission.VIBRATE android.permission.WAKE_LOCK
Package Structure	Archive: apkFiles/2b609e4acfebbee57ecf6ddbfd8202d2.apk Length Method Size Ratio Offset Date Time CRC-32 Name 3366 Stored 3366 0% 0 02-14-12 15:52 0edc5602 res/drawable/icon.png 892 Deflate 394 56% 3421 02-14-12 16:22 5fb1afd7 res/layout/main.xml 65920 Stored 65920 0% 3880 02-13-12 21:54 7e016e52 res/raw/alarm.mp3 2412 Deflate 789 67% 69647 02-14-12 16:22 3c35a471 AndroidManifest.xml 1232 Stored 1232 0% 70701 02-14-12 16:22 2a31087f resources.arsc 12876 Deflate 5912 54% 71977 02-14-12 16:22 abb3a22b classes.dex 471 Deflate 304 35% 77946 02-14-12 16:22 9b12cf3d META-INF/MANIFEST.MF 524 Deflate 331 37% 78316 02-14-12 16:22 f8a5fae4 META-INF/CERT.SF 776 Deflate 607 22% 78709 02-14-12 16:22 7b43123b META-INF/CERT.RSA 88469 78855 11% 9 files
Locales	locales: '--_--'
Supports Screens	supports-screens: 'normal'
Densities	densities: '160'
Native Code	

Figure 11: Detailed description of application structure and related aspects

4.3 Emulator Set-Up & Customisation

The Testing Laboratory is targeted on using Virtualization environment for dynamic application testing and corresponding data gathering. This subsystem provides user with possibility to control (view, delete and create) Virtual Machine for Android, also called AVD. The Android SDK [5] provides Target Platforms (or Operating System images), based on which user can create own AVD.

Before using the Testing Laboratory and after changing configuration of Android SDK, a user should recheck all available Target Platforms and AVDs. The process is time consuming, so this action is going to be executed on user's demand. Corresponding information are stored into two DBs:

1. for available Platforms as it is depicted in the Appendix A (Figure A.1.3),
2. and for available AVDs as it is depicted in the Appendix A (Figure A.1.4).

Mentioned information is taken from the DB due to performance reasons and is presented on corresponding pages in the Testing Laboratory. For purpose on concentrating all possible actions within the Laboratory, controlling Virtual Machine (VM) functionality was implemented. A user can add own AVD based on selected Android Target Platforms as it is depicted on the Figure 12. Corresponding information and status of created devices will be presented.

Additionally, a user can see all available configuration and detailed description of AVDs and Platforms on the relevant page, show on the Figure 13. It provides extensive information such that application programming interface, quite important for Android

Figure 12: User Interface for creating an AVD based on chosen Android Platform Target

Devices (API), application binary interface, depends on emulated physical devices (ABI) and used skin preferences.

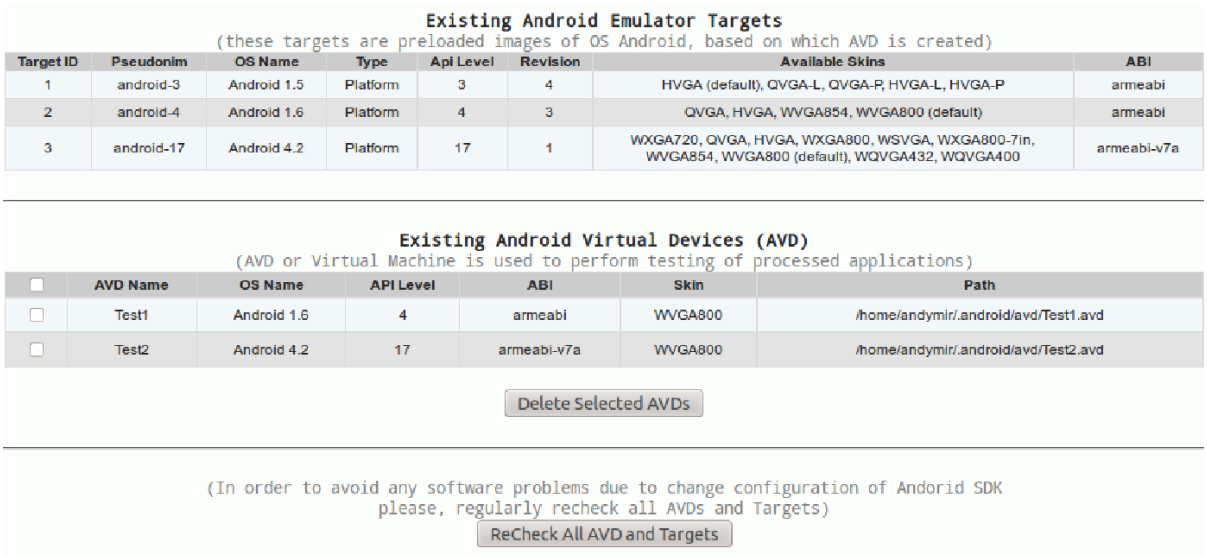


Figure 13: Status of existing Android Target Platforms and created AVDs with detailed description

4.4 Test Cycle

This subsystem provides comprehensive possibilities for a user to test (bunch of) processed uploaded application based on different requirements and available AVDs. A user is capable to configure manually two separated domains of testing as it is shown on the Figure 14:

1. **Emulator** [5]. Responsible for customising launch process of selected AVDs (disable audio, use host graphic acceleration, no boot animation, no window - console mode, specific user profile, intercepting traffic).
2. **Android Debugging Bridge, a tool that is used to control and communicate with running Android Emulator (ADB)** [5]. Responsible for customising dynamic testing process of the application.

During Execution all user data are wiped from Emulator and it does not use snapshots.

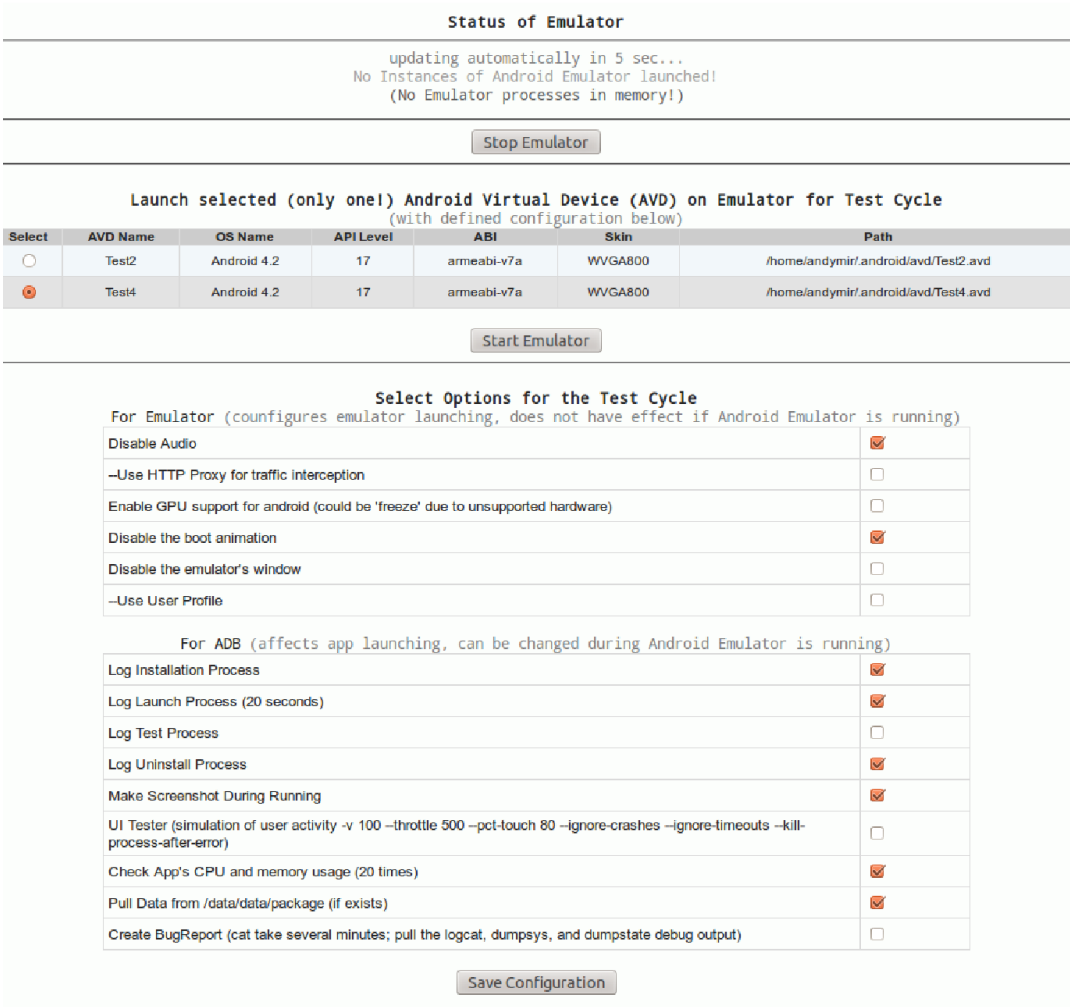
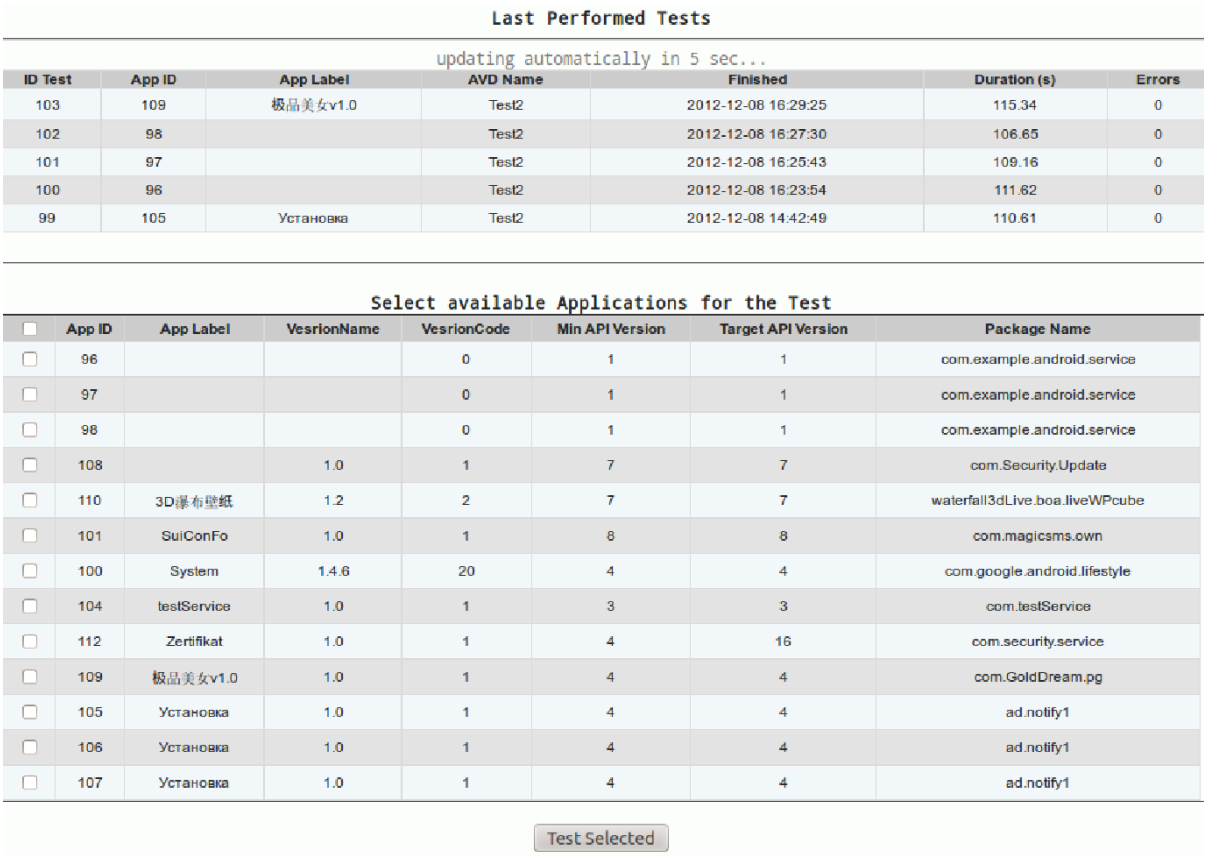


Figure 14: User Interface for configuring and launching Emulator

It ensures that all applications are executed in native and equally clear environment during Test Phase. This phase is mostly Dynamic and provides following functionality for obtaining necessary data: logging all system messages during installation/running/testing, simulating user by means of User Interface testing with defined number of possible events, collecting application's resources usage, taking screenshots, extracting complete bugreport - all available info from the device info and pulling application's data from the */data/data* folder). Further development allows to store network traffic, volatile memory, etc.

During the Test Cycle execution process, information about successfully finished tests and running emulator is automatically updating as on the screen each 5 seconds as it is shown on the Figure 15. As result, it does not require from use excessive actions for obtaining info.

After finishing testing transaction for each particular application, the extensive data are stored in subsystem's table in the DB as it it show in the Appendix A (Figure A.1.5). Additionally, the folder *'Test_<data>_<time>'* is created for storing pulled and gathered data for particular application, which makes it available for further analysis. Performed tests are presented in user-friendly format and has all necessary complete information about a test as it is depicted in the Appendix A (Figures A.1.6, A.1.7).



Last Performed Tests

updating automatically in 5 sec...

ID Test	App ID	App Label	AVD Name	Finished	Duration (s)	Errors
103	109	极品美女v1.0	Test2	2012-12-08 16:29:25	115.34	0
102	98		Test2	2012-12-08 16:27:30	106.65	0
101	97		Test2	2012-12-08 16:25:43	109.16	0
100	96		Test2	2012-12-08 16:23:54	111.62	0
99	105	Установка	Test2	2012-12-08 14:42:49	110.61	0

Select available Applications for the Test

	App ID	App Label	VesrionName	VesrionCode	Min API Version	Target API Version	Package Name
☐	96			0	1	1	com.example.android.service
☐	97			0	1	1	com.example.android.service
☐	98			0	1	1	com.example.android.service
☐	108		1.0	1	7	7	com.Security.Update
☐	110	3D瀑布壁纸	1.2	2	7	7	waterfall3dLive.boa.liveWPcube
☐	101	SuiConFo	1.0	1	8	8	com.magicsms.own
☐	100	System	1.4.6	20	4	4	com.google.android.lifestyle
☐	104	testService	1.0	1	3	3	com.testService
☐	112	Zertifikat	1.0	1	4	16	com.security.service
☐	109	极品美女v1.0	1.0	1	4	4	com.GoldDream.pg
☐	105	Установка	1.0	1	4	4	ad.notify1
☐	106	Установка	1.0	1	4	4	ad.notify1
☐	107	Установка	1.0	1	4	4	ad.notify1

Test Selected

Figure 15: User Interface for launching adjusted Test Cycle for selected processed applications. Last Performed Tests is updating automatically.

4.5 Analysis of the Collected Data

First, basic notes about possibility of intelligence analysis of collected data. The detection of the application that has malicious and illegitimate functionality and intentions is a complex process that requires comprehensive methodology. In perspective of mobile forensics, such a methodology should include following important application's data areas:

1. Required permissions for application
2. Live data patterns in memory
3. Passing data through network interfaces
4. Particular application data on the non-volatile storage
5. Joint storage for applications data
6. Live behaviour (usage of available resources with user interaction or without)

As it was studied in [19, 20, 21] that the majority or threats and key to malicious actions executions inside an applications are stick to permissions. Unfortunately, users mostly approve all required permissions without ensuring what is really necessary and what is not. This leads to quite simple execution of malware and malicious application from unsigned third-party sources [8].

The Analysis subsystem of Testing Laboratory stores all successfully performed analysis data in corresponding table of DB, the structure of which is shown in the Appendix

A (Figure A.1.8). There could be applied various possible configurations for the same test data as it is depicted on the Figure 16.

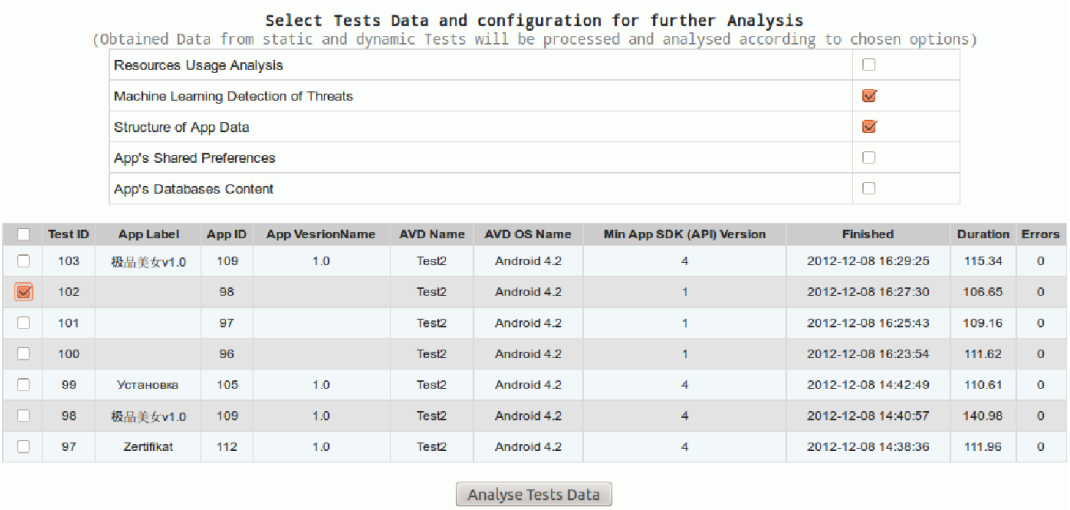


Figure 16: User Interface for analysis configuration and selecting apps

After execution a user can check all performed analysis, which is presented in the Appendix A (Figure A.1.9) and see information for particular analysis for particular test, which is shown on the Figure 17. In case if analysis was configured to use machine learning, the results will also contain basic description of possible treats as well.

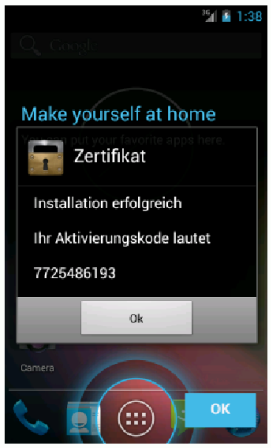
Finally, information from performed analysis can provide estimation of likelihood that considering application represents threats to Confidentiality, Integrity and Availability of user's sensitive data, available for spending money and the mobile machine generally.

	Analysis ID	Test ID	App ID	App Label	Finished	THREATS	Info
☐	295	97	112	Zertifikat	2012-12-09 18:03:52	detected	More

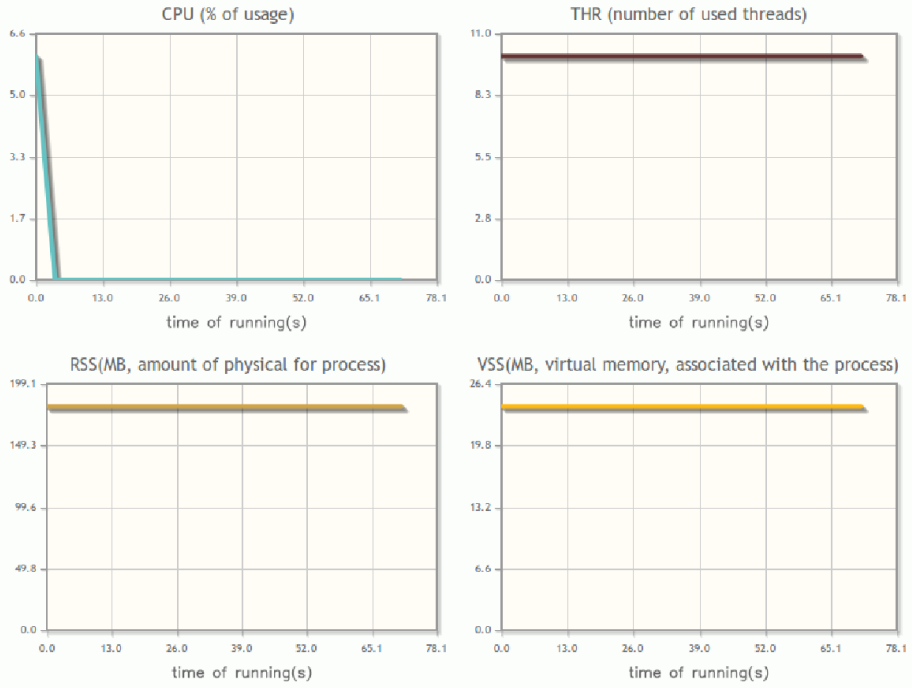
Possible Threats

```
Array
(
    [permissions] => high impact from permissions
    [total] => 2
)
```

ScreenShot



Resources Usage



Data Folder (In /data/data) Structure

```
testData/Test_2012_12_08_14_36_59/data
`-- shared_prefs
    `-- SecurityService.xml

1 directory, 1 file
```

Data Bases extracted content (file,table,content)

Unavailable!

Shared Preferences xml content (file,content)

Unavailable!

Requested Permissions

```
android.permission.RECEIVE_SMS
android.permission.SEND_SMS
```

Figure 17: Extensive information for selected application, which is available after Analysis phase.

5 Results and Analysis

For providing comprehensive evaluation of implemented Testing Laboratory, there were gathered 533 samples of different mobile malware from various hack sites such that *rst-center.com* and *xakep.ru*. Additionally it was found quite big collection of various sample from the source [22]. Among all collected applications, the Testing Laboratory detected only 388 unique samples, other have different names, but content of the APK file is the same. Per application testing time could vary from 2 to 5 minutes and significantly depends on chosen test configuration. The DB structure is presented in In the Appendix A (Figure A.2.10).

Further, for re-validation of collected DB, Clam AntiVirus [25] was selected to check all samples as it shown on the Figure 18.

```
----------- SCAN SUMMARY -----------
Known viruses: 1337368
Engine version: 0.97.6
Scanned directories: 1
Scanned files: 533
Infected files: 377
Data scanned: 629.14 MB
Data read: 788.59 MB (ratio 0.80:1)
Time: 75.910 sec (1 m 15 s)
```

Figure 18: The results of testing collected mobile malware by Clamscan AntiVirus

ClamAV is a misuse (or signature-based) malware detection program as well as the majority of contemporary similar systems [26]. They are capable of providing fast, precise (based on signature) static analyse of heuristic of the file. Unfortunately, antivirus software targeted on mobile devices mostly uses file name comparison and key words detection in content of file. This is, obviously, not enough and can not provide whole picture of application's intentions.

5.1 Prerequisites for Malware Success

Recently, the performed study [27] shows that most appropriate technique for software exploitation is providing a malware under cover of well-known applications, which most of users tend to utilise in everyday life. Feeling of safety because of famous brand dulls the perception of real threats and user can agree with set of dangerous permissions without rechecking it.

5.2 Performed Analysis

All tested and analysed samples of collected applications have requested inappropriate set of permissions. Most of the users do not care about such permissions and answer affirmatively in most case of installing third-party applications. This is very imprudently as i can be figured out from the Figure 19.

The Table 1 presents information about amount of applications that stores various kind of data. It, basically, means that there could be stored various stolen user's sensitive data, cryptographic keys or addresses of web sites. In the Appendix A (Figures A.2.11 and A.2.12), it could be notices that significant amount of applications requires permissions

android.permission.SEND_SMS	253
android.permission.READ_PHONE_STATE	151
android.permission.READ_SMS	126
android.permission.INSTALL_PACKAGES	120
android.permission.RECEIVE_SMS	117
android.permission.VIBRATE	104
android.permission.DELETE_PACKAGES	94
android.permission.WRITE_SMS	89
android.permission.SET_WALLPAPER	80
com.android.browser.permission.READ_HISTORY_BOOKMARKS	78
com.android.browser.permission.WRITE_HISTORY_BOOKMARKS	70
com.android.launcher.permission.READ_SETTINGS	66
android.permission.WRITE_APN_SETTINGS	62
android.permission.WRITE_SETTINGS	58
android.permission.ACCESS_COARSE_LOCATION	50
android.permission.RECEIVE_MMS	48
android.permission.GET_TASKS	48
android.permission.WRITE_EXTERNAL_STORAGE	46
android.permission.ACCESS_WIFI_STATE	46
android.permission.WRITE_CONTACTS	45

Figure 19: TOP20 used permissions in malware set (from 388 samples)

with high level of risk, but it could be hard to figure out it among other permissions with low level risks.

Store any kind of data	258
Store DBs	100
Store shared preferences data	133

Table 1: Amount of application that stores various data on the filesystem (out of 388 applications)

As it could be seen from the extensive results of analysis collected data in the Appendix A (Figures A.2.13 - A.2.25), mobile malware tend to mask as a wide range of applications (due to possibility to rebuild benign application and add malicious functionality). As the mobile device is quite often used as an attractive game platform, adversaries use popular games to hide malicious functionality. Form analysed resources usage, it could be seen that some of the malwares uses significant amount of processor time, which can indicate existence of backdoor part. There could be made variety of decision based on obtained analysis results.

6 Existing Solutions & Further Work

There exist numerous described methods and approaches for Android Applications testing [28, 29, 30, 31], which are aimed on black/grey box validation of functionality, compatibility, usage, performance, requirements and business model, etc. Such methods are not sufficient for Performing Tests of Information Security of mobile applications and assessment of related aspects. They could be successfully used within a companies and by developers to test own applications.

However, there exist several solutions for Information Security testing of Android applications. As it is described in published notes here [32], the physical mobile phone could be used for the testing of security of mobility solutions. Main goal of such methodology is find out how successfully the antivirus solution can detect all possible infected items on the phone. The minor thing of such method is that it require physical devices and therefore the amount targeted platforms is limited.

There also exists open-source scanning solution called Android Security Evaluation Framework (ASEF) [33] for evaluation of security of mobile phones and AVDs as well. It allows to test applications by means of installing, launching, gesturing and unintalling. At the end of scanning, ASEF provides user with data regarding network communication and possible vulnerabilities in applications. Unfortunately, this script provides simple signature based evaluation of application based on defined by author known vulnerabilities from CVE (on December 2012 it has 55 defined vulnerable applications). Further more, it performs no analysis and intended for using by advanced user through command line interface.

As prospective work, it could be highlighted needs for researches in order to provide more comprehensive theoretical background for processing and analysis of vast amount of data after test cycle. Additionally, this data could be used as a learning material for implementing decision support system in future.

7 Conclusions

The project theoretical part reveal core concepts of building and using Android applications on Mobile Devices. As practical part it was successfully implemented Testing Laboratory for testing security assessment of mobile applications. The system provides user with understandable WEB interface and possibility to adjust configuration of the Laboratory. Implemented functionality allows execution of application on wide range of possible platforms and configurations for emulation of agile and dynamic live environment.

As the main goal of the Testing Laboratory, it was chosen to build anomaly detection platform with smart and flexible autonomous execution of single of multiple applications in sequence that does not require user interaction. Finally, it provides basic evaluation and assessment of information security threats, by which user can be targeted in case of usage such application.

For practical part it was also collected and tested more then 500 sample of mobile malware. That give an opportunity to look closer on existing threats and evaluate performance of designed system. Finally, the Testing Laboratory reveal aspects of malware, which were not so obvious before testing.

References

[1] Damon Poeter, Report: Smartphone Shipments Overtake PCs in 2011, http://www.pcmag.com/article2/0,2817,2399846,00.asp?kc=PCRSS05039TX1K0000762, accessed date: 15.11.2012. 3

[2] Whole Product Dynamic: Real World Protection Test Report, Anti-Virus Comparative - October 2012, AV-Comparatives, November 2012. 3

[3] Test Report: Anti-Malware solutions for Android, The Independent IT-Security Institute AV-TEST GmbH, 2012. 3

[4] D. Morrill, Inside the Android Application Framework, https://sites.google.com/site/io/inside-the-android-application-framework , access date: 20.11.2012. 4

[5] Android Developers, http://developer.android.com/, access date: 14.11.2012. 4, 15, 16

[6] R. Grisenthwaite, ARMv8Technology Preview, ARM TechCon, 2011. 4

[7] How to build Android application package (.apk) from the command line using the SDK tools + continuously integrated using CruiseControl, http://asantoso.wordpress.com/2009/09/15/how-to-build-android-application-package-apk-from-the-command-line-using-the-sdk access date: 20.11.2012. 4

[8] A. Hoog, Android Forensics: Investigation, Analysis and Mobile Security for Google Android, Elsevier, 2011. 5, 6, 18

[9] The Building Blocks of Android Application, http://www.mobisoftinfotech.com/blog/android/the-building-blocks-of-android-application/, access date:20.11.2012. 5

[10] D. Morrill, Inside the Android Application Framework, Google I/O, 2008. 5

[11] T. Isohara, K. Takemori, A. Kubota, Kernel-based Behavior Analysis for Android Malware Detection, Kernel-based Behavior Analysis for Android Malware Detection, 2011. 6

[12] J. Haggerty, Digital Forensics, http://www.cse.salford.ac.uk/profiles/haggerty/forensics.php, access date: 10.11.2012. 7

[13] A. Akopyants, WEB vs GUI, Computerra, http://www.computerra.ru/hitech/34344/, access date: 03.12.2012. 12

[14] T. Reenskaug, Model View Controller, http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html, access date: 01.12.2012. 12

[15] ISO 9241, Design of Human-Computer Interation, International Organization for Standardization, 2006. 12

[16] Source Code Size Metrics, http://www.msquaredtechnologies.com/m2rsm/docs/rsm_metrics_narration.htm , access date: 20.11.2012. 13

[17] Count Lines of Code, http://cloc.sourceforge.net/, access date: 20.11.2012. 13

[18] Doxygen, http://www.stack.nl/~dimitri/doxygen/, access date: 01.12.2012. 13

[19] Android permissions explained, security tips, and avoiding mal-

ware, phandroid, http://androidforums.com/android-applications/36936-android-permissions-explained-security-tips-avoiding-malware.html, access date: 05.12.2012. 18

[20] M. Pettitt, Android Apps with "Dangerous" Permissions, MP's Security Blog, http://security.bleurgh.net/android-apps-with-dangerous-permissions, access date: 03.12.2012. 18

[21] Android Permission Map, http://android-permissions.org/, access date: 29.11.2012. 18

[22] Mobile malware dump July 2011, http://contagiodump.blogspot.no/2011/03/take-sample-leave-sample-mobile-malware.html, access date: 16.12.2012. 21

[23] Shalaginov, Andrii. "Advancing neuro-fuzzy algorithm for automated classification in largescale forensic and cybercrime investigations: adaptive machine learning for big data forensic." (2018).

[24] Shalaginov, Andrii. "Soft computing and hybrid intelligence for decision support in forensics science." 2016 IEEE Conference on Intelligence and Security Informatics (ISI). IEEE, 2016. 2

[25] Clam AntiVirus, http://www.clamav.net/lang/en/, access date: 06.12.2012. 21

[26] C. Clifton, M. Bishop, Malicious Code Vulnerability Analysis, 2003. 21

[27] 2011 Mobile Threats Report, Juniper Networks, 2012. 21

[28] Automated Testing of Android Applications, Intel, http://habrahabr.ru/company/intel/blog/152122/, access date: 20.11.2012. 23

[29] Structured Approach to Testing - Android in an Agile Environment, 3i Infotech, Siliconindia conference, 2012. 23

[30] AndroidTestCase, http://developer.android.com/reference/android/test/AndroidTestCase.html, access date: 27.11.2012. 23

[31] Robotium Framework, http://code.google.com/p/robotium/, access date: 01.12.2012. 23

[32] J. Wu, Mobility Security Product Test and Certificate for Android Platform, PC Security Labs, 2011. 23

[33] ASEF - Android Security Evaluation Framework, http://code.google.com/p/asef/, access date: 09.11.2012. 23

Acronyms

ABI application binary interface, depends on emulated physical devices

ADB Android Debugging Bridge, a tool that is used to control and communicate with running Android Emulator

AJAX Asynchronous Javascript and XML

API application programming interface, quite important for Android Devices

APK Android Application Package, container that stores all necessary information for application installation

ASEF Android Security Evaluation Framework

AVD Android Virtual Device, also called Virtual Machine based on Android Operation System image

DB Data Base

GUI Graphic User Interface

KLOC Kilo Lines of Code

LOC Lines of Code

OS Operating System

SD Secure Digital Memory Card

SDK Software Development Kit, bunch of tools for developing, testing and building applications

VM Virtual Machine

A APPENDIX

A.1 System Designing

Figure A.1.1: The User Interface for uploading Android Applications

Field	Type	Null	Key	Default	Extra
id_app	int(10) unsigned	NO	PRI	*NULL*	auto_increment
app_label	varchar(100)	NO	MUL	*NULL*	
md5_name	varchar(32)	NO	UNI	*NULL*	
version	varchar(100)	NO		*NULL*	
versionCode	mediumint(8) unsigned	NO		*NULL*	
sdkVersion	tinyint(4) unsigned	NO	MUL	*NULL*	
targetSdkVersion	tinyint(4) unsigned	NO		*NULL*	
package_name	varchar(200)	NO		*NULL*	
package_structure	mediumtext	NO		*NULL*	
launchable_activity	varchar(200)	NO		*NULL*	
permissions	mediumtext	NO		*NULL*	
filesize	int(10) unsigned	NO		*NULL*	
uploaded	timestamp	NO		CURRENT_TIMESTAMP	
native_code	varchar(200)	NO		*NULL*	
locales	text	NO		*NULL*	
supports_screens	text	NO		*NULL*	
densities	text	NO		*NULL*	

Figure A.1.2: The table 'processedApps' in the DB aimed to store information about processed uploaded applications

Field	Type	Null	Key	Default	Extra
id_target	int(10) unsigned	NO	PRI	*NULL*	auto_increment
pseudonim	varchar(100)	NO		*NULL*	
name	varchar(100)	NO		*NULL*	
type	varchar(100)	NO		*NULL*	
api_level	tinyint(3) unsigned	NO		*NULL*	
revision	tinyint(3) unsigned	NO		*NULL*	
skins	varchar(200)	NO		*NULL*	
ABIs	varchar(200)	NO		*NULL*	

Figure A.1.3: The table 'androidVMtargets' in the DB aimed to store info about available Android Platform Targets, if recheck was performed

Field	Type	Null	Key	Default	Extra
avd_name	varchar(200)	NO	PRI	*NULL*	
path	varchar(200)	NO		*NULL*	
target	varchar(100)	NO		*NULL*	
api_level	tinyint(3) unsigned	NO		*NULL*	
ABI	varchar(200)	NO		*NULL*	
skins	varchar(200)	NO		*NULL*	

Figure A.1.4: The table 'androidAVDs' in the DB aimed to store info about available AVDs

Field	Type	Null	Key	Default	Extra
id_test	int(10) unsigned	NO	PRI	*NULL*	auto_increment
avd_name	varchar(200)	NO	MUL	*NULL*	
id_app	int(10) unsigned	NO	MUL	*NULL*	
test_configuration	text	NO		*NULL*	
log_install	text	NO		*NULL*	
log_launch	text	NO		*NULL*	
log_test	text	NO		*NULL*	
log_uninstall	text	NO		*NULL*	
screenshot	tinyint(1)	NO		0	
pull_data	tinyint(1)	NO		*NULL*	
read_cpu_usage	text	NO		*NULL*	
time	timestamp	NO		CURRENT_TIMESTAMP	
duration	float unsigned	NO		*NULL*	
test_app_process_echo	text	NO		*NULL*	
bugreport	tinyint(1)	NO		*NULL*	
folder_name	varchar(200)	NO		*NULL*	
errors	smallint(1)	NO		0	

Figure A.1.5: The table 'performedTests' in the DB aimed to store Test Cycle result

Performed Tests

Test ID	App Label	App ID	App VesrionName	AVD Name	AVD OS Name	Min App SDK (API) Version	Finished	Duration	Errors	Info
103	极品美女v1.0	109	1.0	Test2	Android 4.2	4	2012-12-08 16:29:25	115.34	0	More
102		98		Test2	Android 4.2	1	2012-12-08 16:27:30	106.65	0	More
101		97		Test2	Android 4.2	1	2012-12-08 16:25:43	109.16	0	More
100		96		Test2	Android 4.2	1	2012-12-08 16:23:54	111.62	0	More
99	Установка	105	1.0	Test2	Android 4.2	4	2012-12-08 14:42:49	110.61	0	More
98	极品美女v1.0	109	1.0	Test2	Android 4.2	4	2012-12-08 14:40:57	140.98	0	More
97	Zertifikat	112	1.0	Test2	Android 4.2	4	2012-12-08 14:38:36	111.96	0	More

Figure A.1.6: Detailed list of performed Tests with possibility to get more info for selected application

☐	99	Установка	105	1.0	Test2	Android 4.2	4	2012-12-08 14:42:49	110.61	0	More

Routine Output

```
--------------------------------------------------------------------------------
Initial Uninstall!
Failure


--------------------------------------------------------------------------------
Install!
        pkg: /data/local/tmp/dc0ef32f1870708b870bc4c8fcf8d9ba.apk
Success


--------------------------------------------------------------------------------
Start App!


--------------------------------------------------------------------------------
Screening!


--------------------------------------------------------------------------------
Stop!


--------------------------------------------------------------------------------
Pull Data!


--------------------------------------------------------------------------------
Final Uninstall!
Success
```

Figure A.1.7: Part of detailed description of particular performed analysis

Field	Type	Null	Key	Default	Extra
id_analysis	int(10) unsigned	NO	PRI	*NULL*	auto_increment
id_test	int(11)	NO	MUL	*NULL*	
id_app	int(11)	NO	MUL	*NULL*	
app_label	varchar(200)	NO	MUL	*NULL*	
folder_name	varchar(200)	NO		*NULL*	
screenshot	tinyint(1)	NO		0	
resources_usage	text	NO		*NULL*	
data_structure_analysis	text	NO		*NULL*	
databases_analysis	text	NO		*NULL*	
ml_threats_analysis	text	NO		*NULL*	
shared_prefs_analysis	text	NO		*NULL*	
analysis_finished	timestamp	NO		CURRENT_TIMESTAMP	

Figure A.1.8: The table 'performedAnalysis' in the DB aimed to store Analysis results

Performed Analysis

(There are presented information, extracted after analysis of collected test data for each application)

☐	Analysis ID	Test ID	App ID	App Label	Finished	THREATS	Info
☐	291	99	105	Установка	2012-12-09 11:24:09	detected	More
☐	290	103	109	极品美女v1.0	2012-12-09 11:24:08	detected	More
☐	289	97	112	Zertifikat	2012-12-08 21:21:58	not detected	More
☐	270	100	96		2012-12-08 20:57:19	not detected	More
☐	269	101	97		2012-12-08 20:57:19	not detected	More

Figure A.1.9: List of Performed Analysis with detailed inline description

A.2 Results and Practical Aspects

	Table	Rows	Type	Collation	Size	Overhead
☐	androidAVDs	4	InnoDB	utf8_general_ci	16.0 KiB	-
☐	androidVMtargets	3	InnoDB	utf8_general_ci	16.0 KiB	-
☐	config	3	InnoDB	utf8_general_ci	16.0 KiB	-
☐	performedAnalysis	360	InnoDB	utf8_general_ci	1.6 MiB	-
☐	performedTests	360	InnoDB	utf8_general_ci	41.5 MiB	-
☐	processedApps	388	InnoDB	latin1_swedish_ci	5.6 MiB	-
	6 tables	**1,118**	**InnoDB**	**latin1_swedish_ci**	**48.8 MiB**	**0 B**

Figure A.2.10: Information the MySQL DB structure and amount of stored data

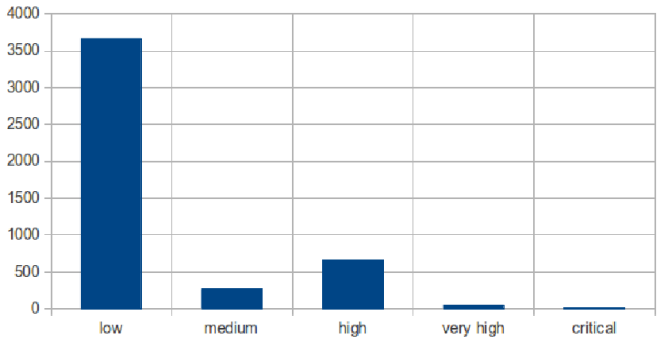


Figure A.2.11: Distribution of all possible permissions from 388 samples, each application can have a single or multiple permissions

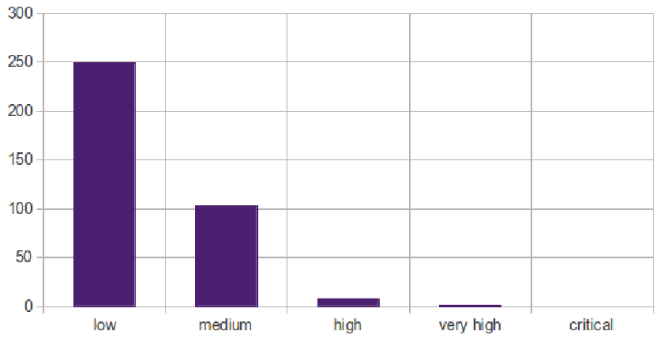


Figure A.2.12: Distribution of average level of requested permissions from 388 samples, major voting calculation among application's permissions

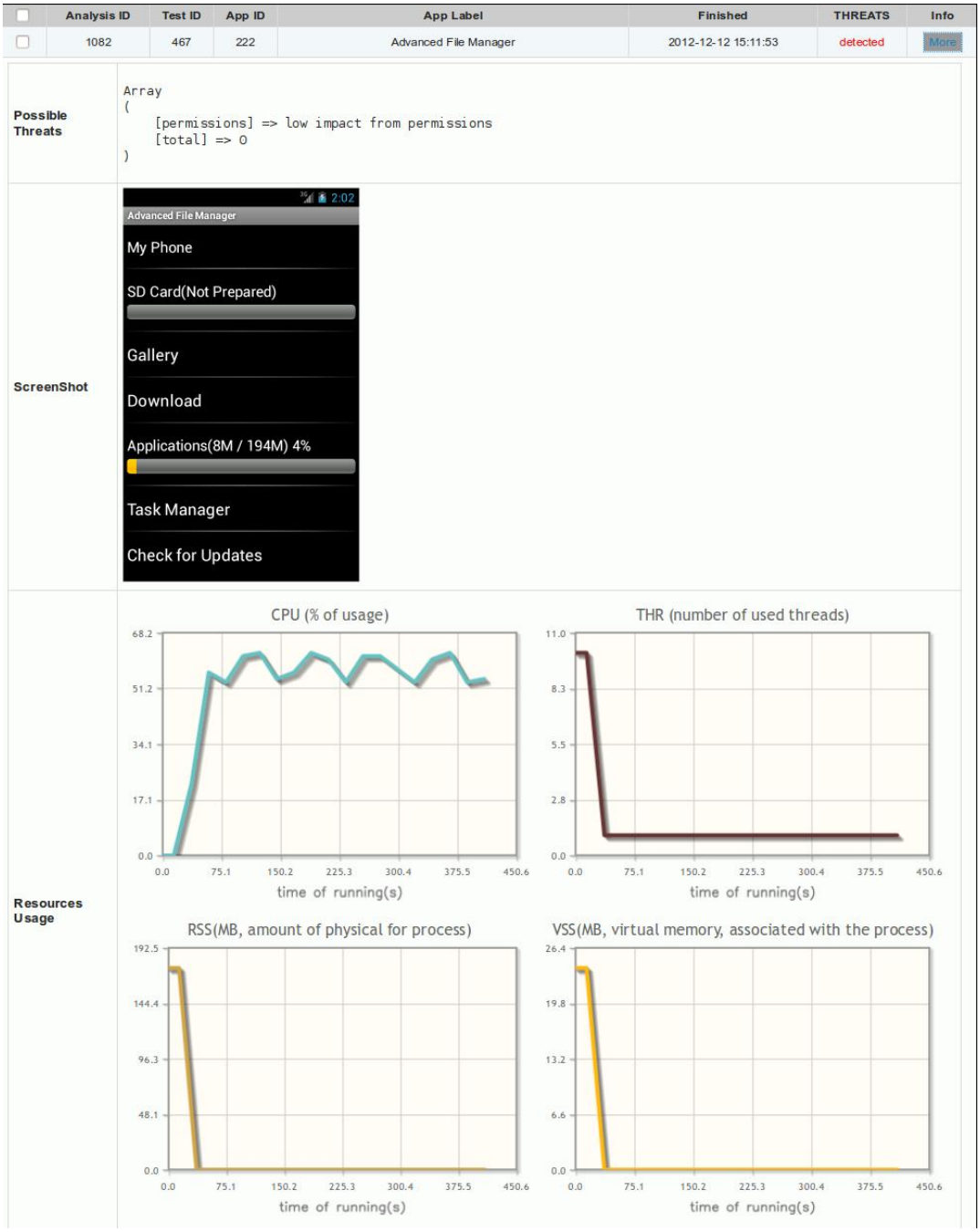


Figure A.2.13: Application 'Advanced File Manager', which has not suspicious interface, but consumes much processor time because of forking and has no memory consumption from main process as result. In fact it migrates to a new process and downloads executable code from the remote server

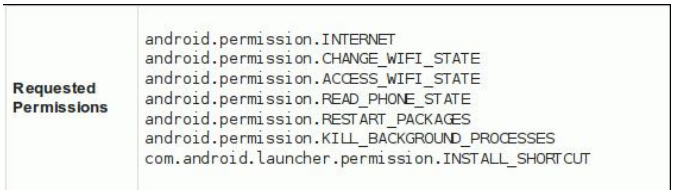


Figure A.2.14: Application 'Advanced File Manager' requires quite strange permissions for file manager

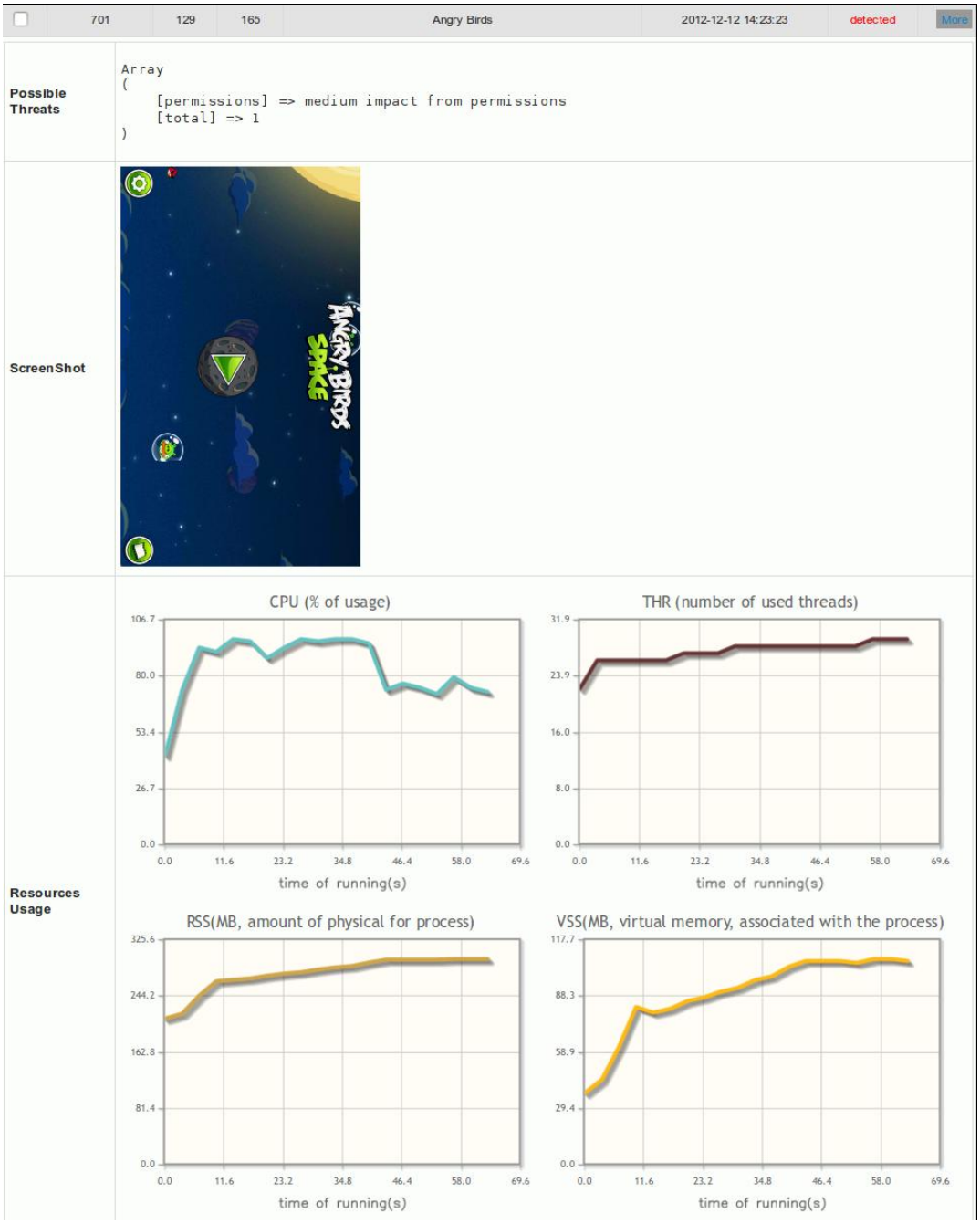


Figure A.2.15: The game 'Angry Bird' is one of the most popular. It has tipical welcome screen and consumes much resources as an interactive game

Figure A.2.16: Permissions for the game 'Angry Bird' looks at least strange, because usually game does not need access to reading log files

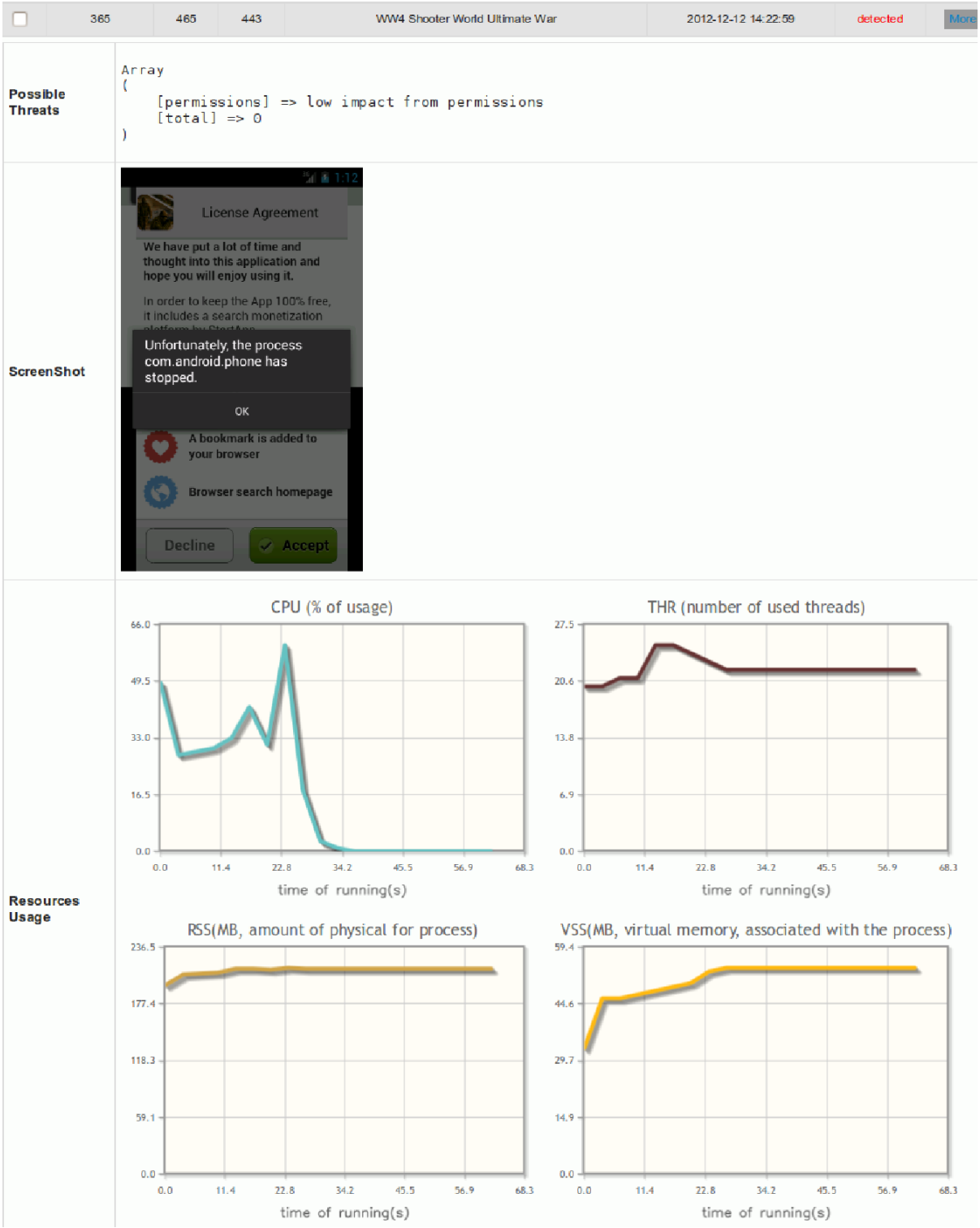


Figure A.2.17: The game 'WW4 Shooter World Ultimate War' has typical name for shooter and licence agreement

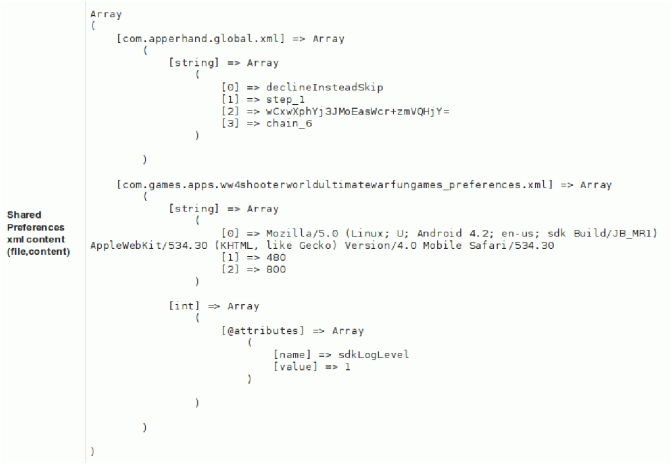


Figure A.2.18: It is not typical for the shooter 'WW4 Shooter World Ultimate War' to store User agent and some kind of hash/key in the DB

Figure A.2.19: Undoubtedly, the game 'WW4 Shooter World Ultimate War' must not have access to calling from the phone

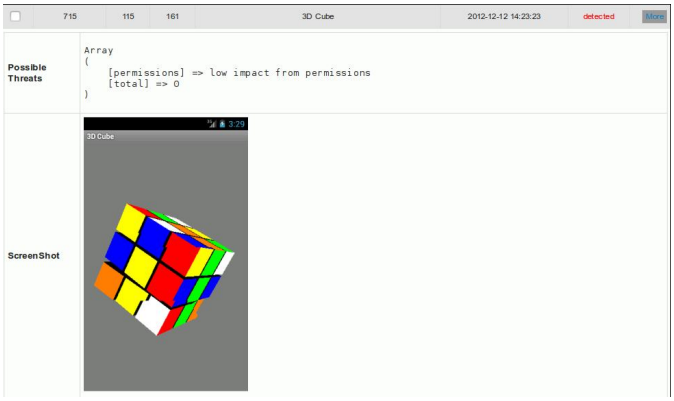


Figure A.2.20: Malware can have quite friendly appearance such that this screensaver with Cubic Rubik

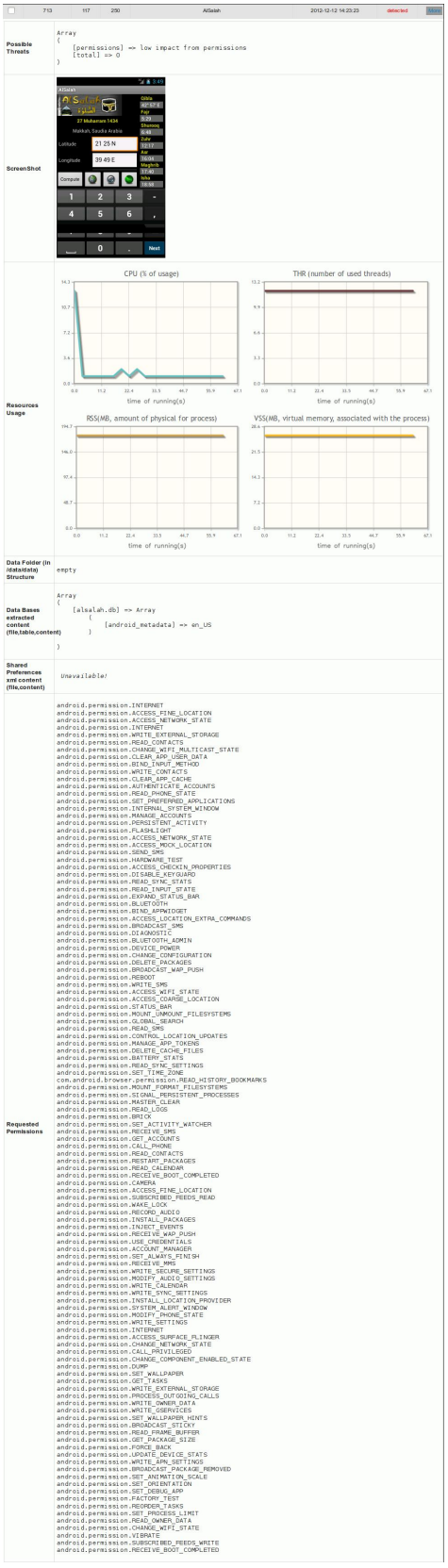


Figure A.2.21: It is easy to deduce from the UI that this application is intended to calculate some kind of GPS coordinates from entered parameter. As it could be noticed, the required list of permissions is quite long and, obviously, this application should not have access to rebooting device, making calls and sending SMSs

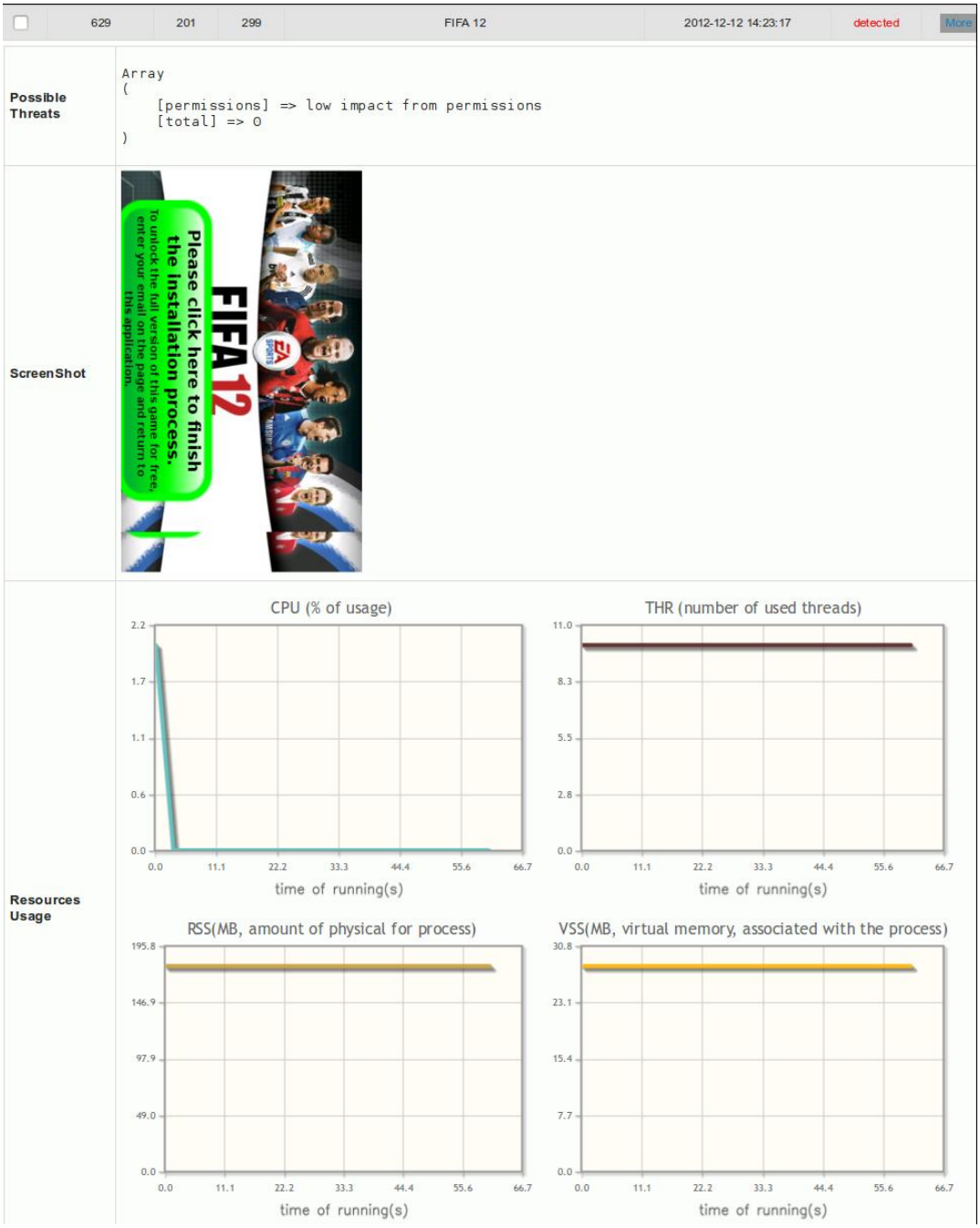


Figure A.2.22: Well-known football game 'FIFA 2012' has typical interface and welcome screen for such type of games.

Requested Permissions	android.permission.INTERNET

Figure A.2.23: The required permissions for the game 'FIFA 2012' has only one option - to access the Internet. So, we can understand that this applications is not for playing football

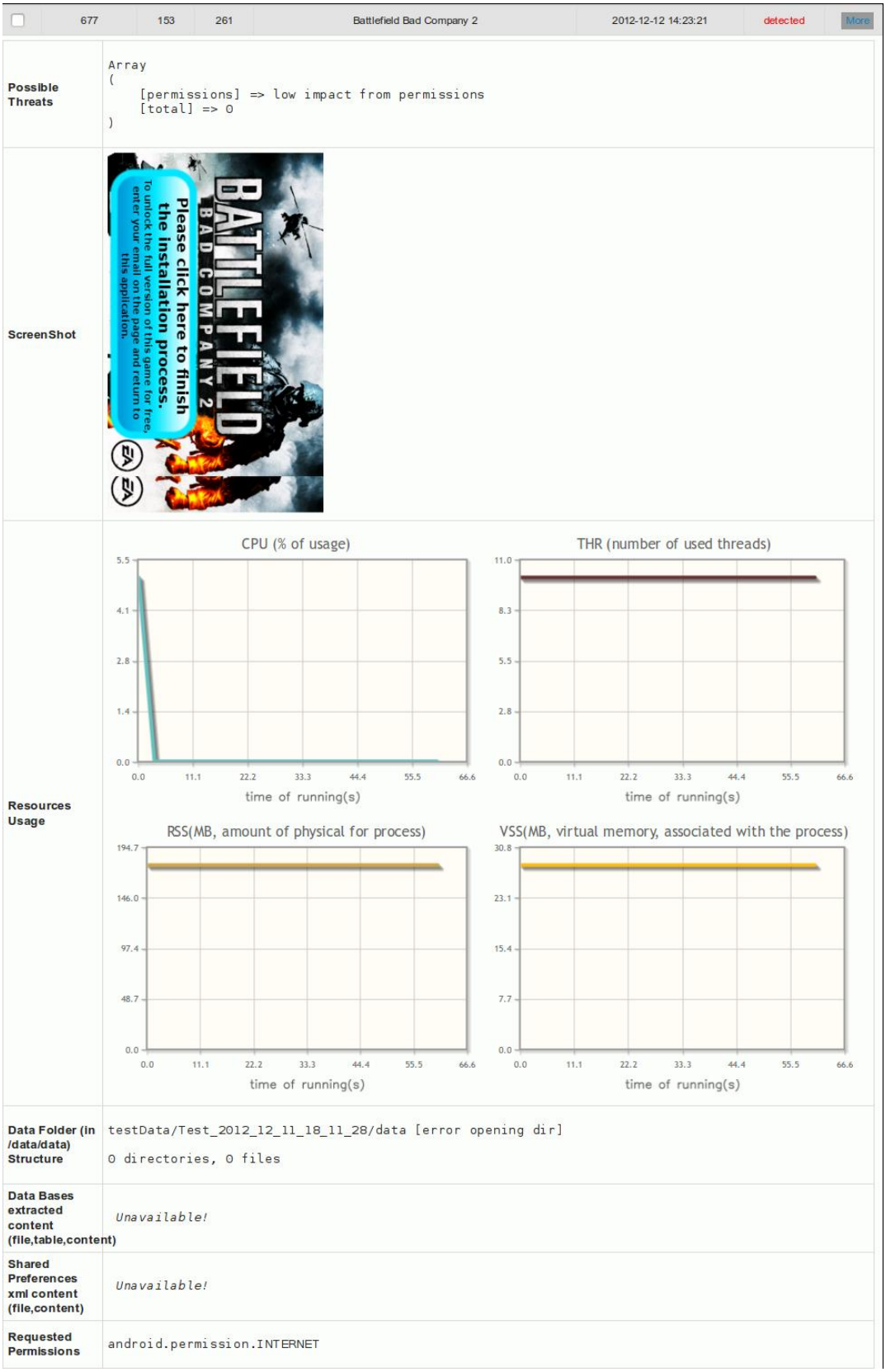


Figure A.2.24: Famous game 'Battle Field Bad Company', which has familiar welcome screen and only requires access to the Internet - obviously not a game. It is possible to figure out similarities with analysis data if the considered earlier game 'FIFA 2012' on the Figures A.2.22 and A.2.23

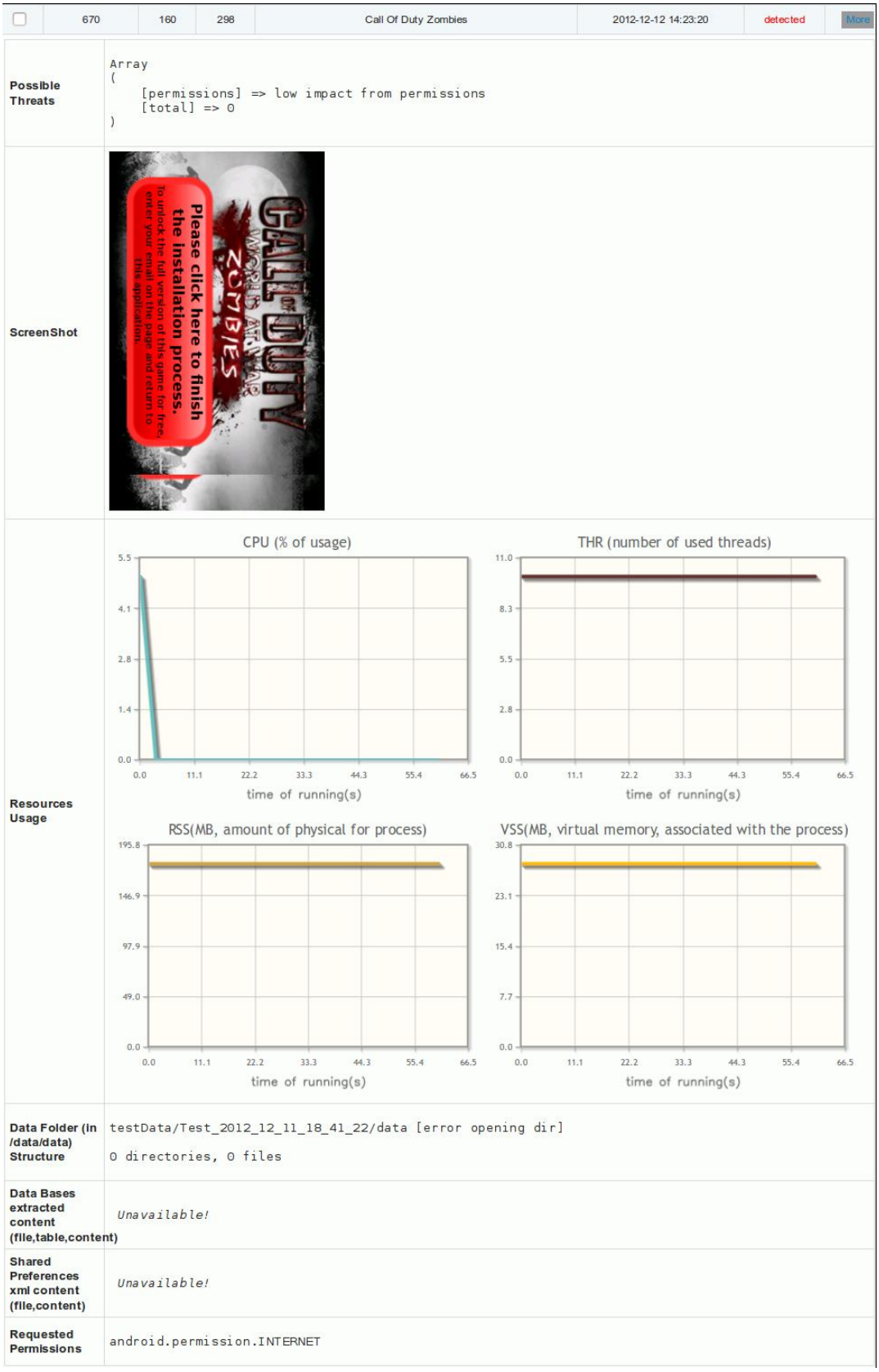


Figure A.2.25: Popular shooter game 'Call of Duty' has identical analysis data with mentioned previously 'FIFA 2012' on the Figure A.2.22 and 'Battle Field Bad Company' on the Figure A.2.24. It indicates that this application is not a game and has malicious intentions