Development of the Protection System Against Malicious Software

D. M. Mikhaylov, A. V. Zuykov, M. I. Froimson and A. S. Smirnov
Engineering Center of the National Research Nuclear University "MEPhI"(Moscow Engineering Physics Institute), Kashirskoye highway, 31, Moscow, Russia

Keywords: Mobile Security, Antivirus, Malicious Code, Identification of Potentially Dangerous Applications.

Abstract: Nowadays mobile phone manufacturers offer their customers not only devices for calling and sending SMS-messages, but also multi-functional smartphones with much broader capability. Mobile phones are used to access the Internet, for e-mailing, social networking, finding phone owner`s location and routing, etc. Due to its multiple functionalities, a smartphone can store a lot of data that may be breached by theft or data leakage. This article focuses on development of security software for mobile devices able to detect malicious software and infected files on the device as well as to protect the user from applications recording personal conversations, secretly transmitting data and activating programs. Antivirus software monitors secret camera activation warning the user. The paper also provides the comparison of the proposed security software with popular antiviruses: Kaspersky Mobile Security, Dr.Web, ESET, Norton Antivirus, etc.

1 INTRODUCTION

Mobile devices has become an important part of our everyday life as they allow not only making calls and sending messages but also access the Internet and perform a wide range of functions (Wang, 2013; Kim, 2013; Min, 2014; Amalan, 2013; Szakacs-Simon, 2013; Lee 2013). With the growing number of smart phone users, the problem of information security becomes rather significant. Intruders can not only infect the device and steal confidential information as well as money from the phone or mobile bank account but also bring it out of operation (Mikhaylov and Zhukov, 2013).

Android is the most popular operating system (OS) nowadays (Northcraft, 2014) and thus there is a sharp increase in mobile malicious software. The ease of modifying and the simplicity of the design of the operating system are the aspects that cause system vulnerabilities and are drawing malware developers towards Android smart phones. The issue of leaking private information because of vulnerabilities of Android applications is discussed in (Shahriar, 2014; Sbirlea, 2013; Taenam, 2013). (Mulliner et al., 2012) consider malicious injection of cellular signaling traffic from mobile devices. Stirparo, Fovino, Taddeo and Kounelis tell about stealing user's credentials and sensitive private information stored and processed by Android devices (Stirparo, 2013). Mobile phone can also suffer attacks performed by malware circulating via QR codes (Woo Bong, 2011), using security halls of GSM (Miller, 2011), etc. (Barrere and Cozzette, 2013); and this list can be continued.

Market analysis of the antivirus class products shows not only the failure of the modern mobile antivirus software to counterwork the one developed by attacker, but also brings into question effectiveness of transferring standard approaches to developing such software from PC to mobile platform.

The issue of development of new effective protection software is of great interest today (Al-Saleh, 2013; Bläsing, 2010). For example, (Pieterse and Olivier, 2013) evaluate the current state of mobile security applications and propose simple steps for Android mobile device users to protect their phones. In (Fu-Hau Hsu, 2012) the mechanism preventing antivirus from being terminated without the consciousness of the antivirus software users is proposed. (Chia-Mei Chen and Ya-Hui Ou, 2011) tell about secure mechanism for mobile web browsing detecting client-side malicious web sites. In (Kasama, 2012) a malware detection method based on investigation of behavioral difference in multiple executions of suspicious software is presented. (Dube et al., (2013) propose malware target recognition of unknown threats.

Mikhaylov D., Zuykov A., Froimson M. and Smirnov A..
Development of the Protection System Against Malicious Software.
DOI: 10.5220/0005060201610170
In *Proceedings of the International Conference on Knowledge Management and Information Sharing* (KMIS-2014), pages 161-170
ISBN: 978-989-758-050-5

Almost all existing antivirus products for mobile platforms use the procedure of scanning files to detect malicious code in their applications according to databases of known viruses. Such protection called signature analysis has a major drawback which is that the slightest change malicious application code makes it undetectable to antivirus until malicious code is listed in the signature database (Dube, 2013). As a consequence, antiviruses with protection of this type can only detect widespread malicious applications and obviously weak to new threats that may be simply a modification of the old ones. To provide protection from threats such antivirus solutions should be frequently updated (not less than once a day). Moreover, signature bases reach a large size that is critical for mobile devices that are not designed to store large amounts of data.

Consider the most common protection methods for Android OS:

- Static analysis allows identifying malicious applications by analyzing the code before installation on the presence of matching signatures with values from the database. Well-developed obfuscation techniques require creating a new signature for modified malware codes. This leads to the necessity of frequent updates and signature database storage (can reach large size) which is not suitable for mobile devices with a limited internal memory. Note that it is often difficult to distinguish a legitimate application that uses access to any data from malicious one by this method. (Hsiang-Yuan 2013; Batyuk, 2011; Yan, 2013)
- Dynamic analysis is generally not used due to lack of necessary methods for its operation. The antivirus software is functioning on a mobile device sending software to be checked on the developing company server where it is tested; the results are sent back. Such methods often require a lot of time and continuous access to the Internet that is not always possible. (Yan, 2013)
- Monitoring – large number of projects related to Android OS protection from viruses aimed at adding to Android OS capabilities for tracking user applications and if needed – prohibiting access to them. An example of such protection is the TaintDroid project (Enck, 2010). The drawback of all these solutions is the necessity of OS modification. Moreover, because of the fragmentation of Android devices it is not enough just to insert changes to operating system`s code once. It is necessary to support the work of the code for a large number of different devices on different versions of Android.
- Virtualization allows running on the same physical device several Android operating systems simultaneously. This approach allows a very strict separation of application as well as their isolation but greatly reduces the device speed and requires OS modification.

Therefore, antivirus software based on a fundamentally new approach to protecting against malicious applications for mobile platforms is proposed and implemented within the framework of the project. It is based on wrapping potentially dangerous system calls by drive code that allows monitoring and, if necessary, blocking dangerous calls. It also provides protection against unauthorized calls commitment by creating an intermediate pseudo device filtering commands between RILd (radio interface layer demon) and modem. The proposed security software uses the camera and recorder features to detect unauthorized access aimed at obtaining information about the user's environment.

2 SECURITY SOFTWARE DEVELOPMENT

Before describing the proposed antivirus software, consider Android operating system (Android operating system architecture 2011). Each application in the Android system operates under its own Linux account (on the core level) and runs its own virtual machine Dalvik as a separate process. Thus, direct interaction between the processes becomes impossible as well as invasion of one process into the memory of the other and access to the data of another process. Interaction between running applications and data sharing is very limited and regulated by the application that provides data during the installation. Android system in principle suppresses the possibility of creating "classic" computer virus that injects its code into the data area of other processes, set itself up in the system modules and also distributes itself over data networks (because there is no opportunity to start the installation of malicious application on the remote device).

The designed antivirus is based on the so-called proactive protection, the main task of which is to detect antivirus security threats in real-time by

monitoring applications activity for potentially dangerous actions (data transmission, locating, sending SMS, making outgoing calls, reading user data and accessing integrated phone capabilities – camera, microphone) and prevent them by blocking dangerous actions and alerting the user.

The main mechanism of the antivirus is to incorporate its drive code into the controlled application. Potentially dangerous methods calls are covered by shell methods, which request antivirus for policies in connection to the actions and accordingly implement or not the target method and transmits or not information about it to antivirus to log the action and notify the user.

Control is established as follows. .apk file (Android application package) is the package file format used to distribute and install application software and middleware onto Google's Android operating system (Lawrence, 2012). First, .apk file target application is dearchived (it is a zip-archive).

We get a set of files, among which two are used: classes.dex – contains executable code of the application – and AndroidManifest.xml – contains information about the application components, required permissions, etc. Byte code of classes.dex file is parsed into classes, their methods, members, etc., a shell method class and a helper class for getting application context are added to it (AndroidManifest.xml file also accordingly modified). Then classes disassembled code is searched for all potentially dangerous methods calls, and those are replaced with shell method calls with the same parameters and return types. Figure 1 provides general conditioning of .apk file.

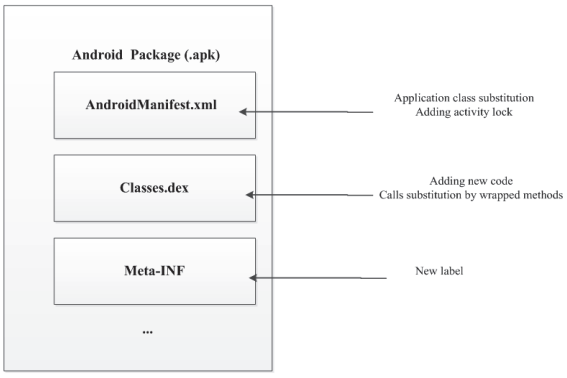

Figure 1: General apk-file conditioning.

Then .apk file is reassembled and signed by a newly generated signature unique to each application. Unique signature for each application is needed because of more possibilities of communication of applications with the same signature. Then it is installed once more. The source .apk file is saved for possible rollback.

Control code incorporation is implemented in C++ language as a native library. To ensure maximum performance modifications are made directly in dex format without conversion to smali-code and vice versa.

Table 1 shows the general structure of the dex-file.

To integrate the drive code first it is necessary to

Table 1: Dex-file general structure.

Name	Format	Description
header	header_item	Heading
string_ids	string_id_item[]	List of line identifiers. These identifiers contain information on all lines of the file used both for the internal designation (e.g., type descriptors) and constants used in the code.
type_ids	type_id_item[]	Types identifier list (classes, arrays and primitive types). Those types are referenced in the file, either specified therein or not.
proto_ids	proto_id_item[]	Methods counterparts referenced in the file identifier list.
field_ids	field_id_item[]	Fields referenced in the file, either specified therein or not, identifier list.
method_ids	method_id_item[]	Methods referenced in the file, either specified therein or not, identifier list.
class_defs	class_def_item[]	Class definition list. Classes must be sorted so that for each class all parent-classes and implemented interfaces are defined above.
data	ubyte[]	The data area that includes all supporting information for the above tables.
link_data	ubyte[]	The data used in the statically linked files.

merge classes.dex files of the target application and controlling module. Each section N of the two files listed in the table is combined according to the sorting requirements and without allowing duplicates.

Drive code requires access to the application context, so it is necessary that the application class created when staring application contained the code for reference. This class can be specified by the application in AndroidManifest.xml: <applicationandroid:name="com.mypackage.MyApplicationClass">. If the target application does not specify the class and the base implementation, for integration it is enough to add the attribute to the manifest. However, if the application class is specified, in order to preserve the functionality of the application it cannot be replaced by the class of your own. Inheritance link between classes can be a solution. Thus, the resulting class will contain both the original application code, and the code for getting context of controlling module. Listing below shows a method of establishing inheritance link.

```
   void DexFileBig::setInheritanceForWAppClass(ClassDef *classDefParent, ClassDef
*classDefChild,DexFileWriter *outClassDefs, DexFileWriter *outClassDatas) {

    classDefChild->superclass_idx = classDefParent->class_idx;

    CountedPtr<ClassData::EncodedMethod>         methodOur         =         this-
>getNonStaticMethodByName(classDefChild, "onCreate");
    ClassDef *lastOverridingClass;
    CountedPtr<ClassData::EncodedMethod> methodTheir = findLastMethodOverride(
            classDefParent,         "onCreate",         "Landroid/app/Application;",
&lastOverridingClass);
    CountedPtr<ClassData::EncodedMethod> initOur = this->getStaticMethodByName(
            classDefChild, "<init>");
    CountedPtr<ClassData::EncodedMethod> initTheir = this->getStaticMethodByName(
            classDefParent, "<init>");

    if(methodOur == NULL || initOur == NULL || initTheir == NULL) {
        LOGET("DexFileBig::setInheritanceForWAppClass "
                "methodOur:%d, initOur:%d, initTheir:%d",
                methodOur.get(), initOur.get(), initTheir.get());
        return;
    }

    if(methodTheir != NULL) {
        this->replaceIdInCallSuperMethod(methodOur.get(),   methodTheir->method idx,
0x106f);
        this->setNeededAccessFlagsOnWAppMethod(methodTheir.get());
        if (lastOverridingClass != classDefParent) {
            setNeededAccessFlagsOnWApp(lastOverridingClass);
            outClassDefs->pushMove(lastOverridingClass->offset);
            lastOverridingClass->dump(outClassDefs);
            outClassDefs->pop();
            outClassDatas->pushMove(lastOverridingClass->class_data->offset);
            lastOverridingClass->class data->dump(outClassDatas);
            outClassDatas->pop();
        }
    } else {
        LOGWT("DexFileBig::setInheritanceForWAppClass methodTheir not found");
    }

    this->replaceIdInCallSuperMethod(initOur.get(), initTheir->method_idx, 0x1070);

    this->setNeededAccessFlagsOnWApp(classDefParent);
    this->setNeededAccessFlagsOnWAppMethod(initTheir.get());
   }
```

```
   publicstaticvoid requestLocationUpdates(LocationManager lm, String provider,
long minTime, float minDistance, LocationListener listener) {
    Log.d(TAG, "Watcher watches requesting location updates");
    Context awa = WApplication.getAppContext();
    int[] permissions;
    if (awa != null)
        permissions = Politics.getPermissionFromDb(awa, awa.getPackageName(),
"act location");
    else
        permissions = newint[] { Politics.NONE, Politics.NONE };
    int recPermission = Math.min(permissions[0], permissions[1]);
    switch (recPermission) {
    case Politics.NOTIFY:
        sendNotification(awa, ACTION LOCATION, MSG LOCATION, recPermission, null,
null, null, null);
    case Politics.NONE:
    case Politics.IGNORE:
        lm.requestLocationUpdates(provider, minTime, minDistance, listener);
        Log.d(TAG, "Location updates requested");
        break;
    case Politics.BLOCK:
        showAlertWindow(awa, ACTION_LOCATION, ACTION_DESCR_LOCATION, null, null);
    case Politics.BLOCKONLY:
        sendNotification(awa, ACTION LOCATION, MSG LOCATION, recPermission, null,
null, null,null);
        thrownew SecurityException();
    }
   }
```

Potentially dangerous methods call control is implemented by shell method. Listing below shows an example of such a method for location request

To integrate these control shells in the application code, all potentially dangerous methods calls are found in data-section dex-file and replaced with calls to the relevant shells. As there are no chances to interact with user data or other applications without particular API call and the limited number of such methods, we can wrap all these calls with control code. Thus, we will have control of all methods that could be potentially dangerous. The proposed algorithm bypasses the need to modify Android OS and download new virus base while providing much of the security the user desire.

Potentially dangerous activities are grouped as follows according to a threat to user: INTERNET (data transmission), TRACKING (using camera, microphone), LOCATION (positioning), MONEY (outgoing calls and SMS), USERDATA (access to personal user data) CONFIG (system settings change). These groups are made according to the user friendly sense separation, which the standard Android Package Installer allowance confirm dialog during applications installation lacks.

For each of the above permission groups one of the policies is set:

- allow – the action is performed in normal mode;
- notify – the action is performed, but the user is notified and the action is recorded in the log;
- notify and block – the action is not performed and a window with information about the blocked action is shown to the user;
- block – the action is not performed and a notification is not shown.

In case of potentially dangerous method call of a controlled application, processing is performed according to the algorithm shown in Figure 2.

An important aspect of the described proactive approach of the antivirus software is to prevent unauthorized camera activation by malicious software.

Android security policy does not allow the camera to run without preview image, which should protect against unauthorized access to this feature, but there is no restriction on size or position of the preview that still allows an attacker to conduct covert photography. Developed antivirus software monitors the calls to camera and warns the user when the camera is run without a corresponding display on the screen. Analysis shows that existing antivirus software lack these features.

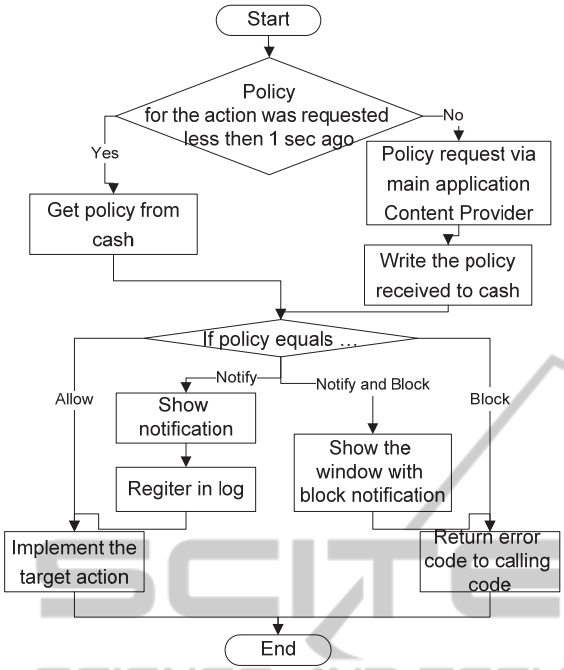

Figure 2: Algorithm of processing a potentially dangerous method call of a controlled application.

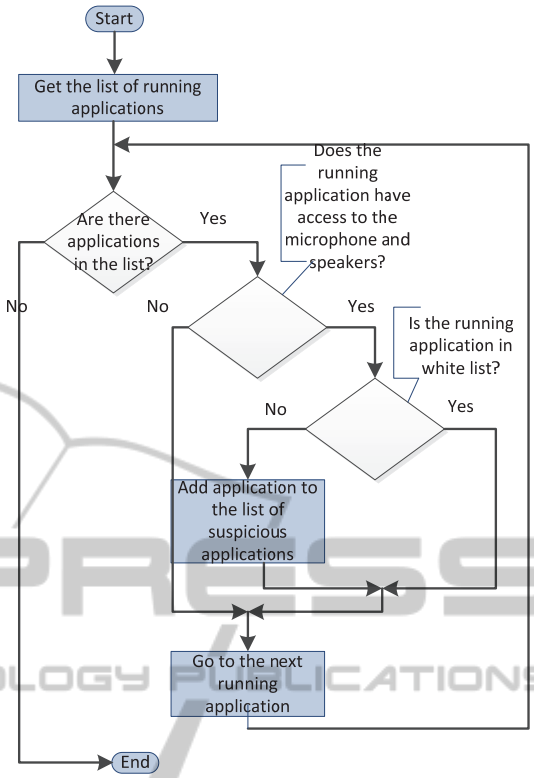

Figure 3: Flowchart of making the list of running applications that have access rights to the microphone and speakers of the mobile device.

Protection against malicious applications that record telephone conversations and conversations nearby the user was developed on the basis of working features of such applications. Features of the application described above were obtained from an analysis of opportunities for recording telephone calls to user-level applications. During the analysis it was found that each application installed by the user on the mobile device must contain the access right to particular OS functions. In addition, simultaneous access to the microphone and the speakers of the mobile device for recording incoming data can be got by only one application (if other applications try to call for, they will get the device-is-busy-by-another-application exception).

On the basis of the features of malicious applications that record telephone conversations and conversations nearby the user, the algorithms implementing the following were developed:

- creating a list of running applications that may to record conversations (Figure 3);
- mobile device microphone and speakers availability check (Figure 4).

Checking access rights to mobile device microphone and speaker is based on the presence of android.permission.RECORD_AUDIO permission in the configuration file of running application (Figure 4).

The mobile device microphone and speaker availability check is performed by an attempt to record a telephone conversation or nearby conversation. Success of the attempt to record indicates the absence of applications calling to the microphone and speaker.

In the case of exception proving the microphone and speaker being used, a list of running applications that may record conversations is created. This list of applications is shown to the user, who can terminate any application from the given list or add it to the white list. Applications from the white list will be marked as trusted.

Protection against malicious applications that perform program calls and hidden data transmission is developed on the basis of features, revealed during analysis of program calls and hidden data transmission capabilities. During the analysis it was found that the program for rooted devices developed

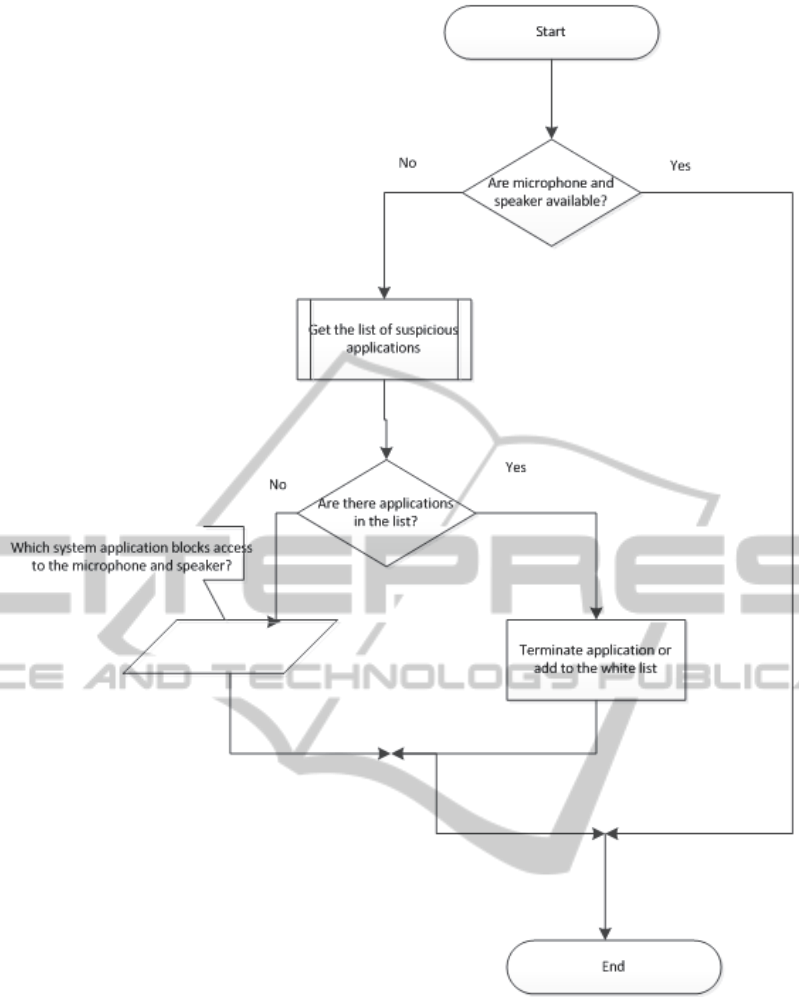

Figure 4: Flowchart of microphone and speaker availability check.

by an attacker must interact directly with the radio interface layer (RIL), sending AT-commands for the call and the data transfer.

According to the described features of malicious applications that perform program calls and hidden data transmission, to ensure protection it is necessary to filter AT-commands received by baseband-processor. In order to filter AT-commands received by baseband-processor, a layer responsible for the interaction between baseband-processor and RIL demon was developed.

Creation of the interaction layer between baseband-processor and RIL demon means renaming pseudo-terminals /dev/smd0 в /dev/smd0Real, and then creating the process that will create a proxy /dev/smd0 and two streams for working with pseudo-terminals and proxy (Figure 5). Initial /dev/smd0 is a driver for the baseband. All interactions between the phone and base stations pass through this driver.

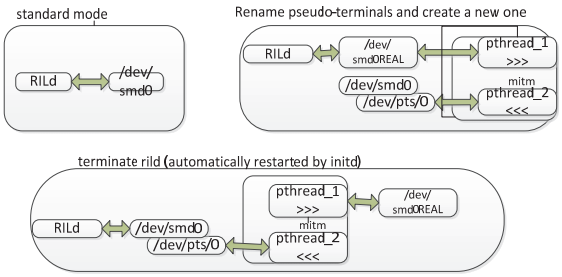

Figure 5: Stages of a filtering mechanism introduction for Android.

To implement AT-commands transferred through RIL to baseband-processor filtering mechanism, streams were set up as follows:

- pthread_1 stream connects to the pseudo-terminals and records AT-command got from pthread_2 stream to it;
- pthread_2 stream (filter stream) connects to the created proxy new /dev/smd0 and transfer AT-commands that do not follow filtering rule. So all suspicious AT- commands will be filtered on this stream.

Filtering is conducted according to the AT-commands responsible for outgoing calls and data transfer identified during the analysis:

ATD – dial an outgoing call;

ATA – answer an incoming call;

AT+CMGS – send a data message;

AT+CMSS – send a saved message;

AT+CBST – establish connection for data transmission through the Internet.

The algorithm that provides the foundation for filtering outgoing calls is shown in Figure 6.

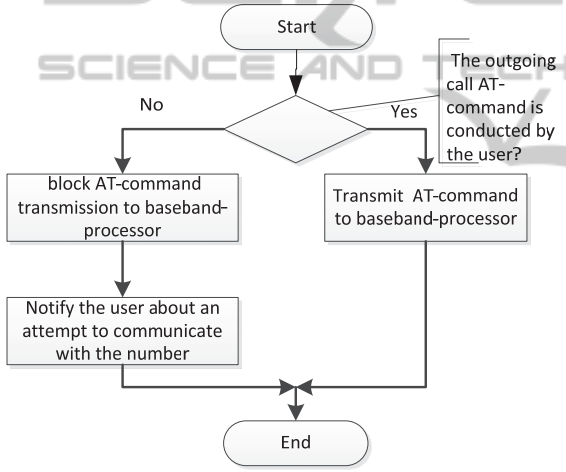

Figure 6: Flowchart of outgoing calls filtering.

The algorithm shown in the figure 4.6 is implemented in pthread_2 stream. When an AT-command comes from the created proxy /dev/smd0, the stream checks it with ATD. Getting ATD order means that an outgoing call takes place, but nothing proves if it is initiated by the software. To identify the software call the stream conducts a flow outgoing call database query. If the phone number being called is the same as in the last number in outgoing calls database record, the pthread_2 stream transmits AT-command to the pthread_1 stream, which in turn transmits it to baseband-processor, and the outgoing call is being performed. Otherwise, the pthread_2 stream drops AT-command and notifies the user about an attempt to communicate with the number specified in the command.

3 TESTING

The proposed antivirus security software has been tested following the steps:

- antivirus installation on the infected device;
- virus database update;
- full scan;
- infected files delete. Cache delete;
- attempt to download an infected object from the Internet.

In order to evaluate the performance of antivirus, devices with different security software (namely Kaspersky Mobile Security, Dr.Web, ESET, Norton Antivirus, F-Secure Mobile Security, McAfee Mobile Security, AVG Mobilation, Avast!, Avira Anroid Security, Trend Micro Mobile Security, Lookout Premium and the proposed one) were infected by viruses such as Android.SmsSend.186.origin and remote control spy programs such as Mobile Spy (Trojan Android.SmsSend.186.origin 2012, Mobile Spy 2013).

The testing results are presented in Table 2.

Table 2: Popular antiviruses' performance testing on devices infected by Android.SmsSend.186.origin.

	Proposed security software	**Kaspersky Mobile Security, Dr.Web, ESET, Norton Antivirus**
Effectiveness	Detect all tested malware	No one detects all malware
Database update	Only software update; database update is not required	Daily database update is required taking lots of memory
Full scan	All infected files are found	All infected files are found only by Kaspersky Mobile Security
Infected files delete	Dialog window with options: delete, ignore	Dialog window with options: delete, ignore
Attempt to download an infected object	Prohibited	Prohibited only by Kaspersky Mobile Security and Dr.Web

Based on the testing, it was concluded that proposed antivirus has the same operation quality as Kaspersky Mobile Security and Dr.Web, which are widespread in Russia. In addition to direct protection against viruses and malware, the developed software can provide such additional functions as: hide personal contacts and information, block unwanted

calls and SMS, unauthorized camera and microphone activation prevention.

4 CONCLUSIONS

The proposed antivirus can detect hazardous applications and programs that lead to malfunction, incorrect system work, device speeding down, data loss or corruption, system alert. It is able to protect the user from applications recording conversations, both personal and telephone, and does not allow spy applications to secretly transmit data and activate programs. Antivirus software monitors camera activation and warns the user when the camera is on without displaying on the screen corresponding information. This feature is not available in the existing well-known antivirus software.

The work is underway to improve the effectiveness and speed of the proposed antivirus, diversify its functions and carry out additional tests and simulations confirming security software efficiency.

REFERENCES

Wang Guolu; Qiu Kaijin; Xu hai; Chen Yao. 2013. The design and implementation of a gravity sensor-based mobile phone for the blind. *4th IEEE International Conference on Software Engineering and Service Science* (ICSESS). Pages: 570 – 574.

Dohee Kim; Eunji Lee; Sungyong Ahn; Hyokyung Bahn. 2013. Improving the storage performance of smartphones through journaling in non-volatile memory. *IEEE Transactions on Consumer Electronics*, (Volume: 59, Issue: 3). Pages: 556 – 561.

Xing, Min; Xiang, Siyuan; Cai, Lin. 2014. A Real-Time Adaptive Algorithm for Video Streaming over Multiple Wireless Access Networks. *IEEE Journal on Selected Areas in Communications* (Volume: 32, Issue: 4). Pages: 795 – 805.

Amalan, D.; Jurangpathy, B.Z.; Kodituwakku, S.D.; Luckshan, M.A.; Weerakkody, P.K.; De Silva, K.P.D.H. 2013. Voice active mobile browser for windows phone 7 (A browser that helps visually impaired people). *IEEE Global Humanitarian Technology Conference: South Asia Satellite* (GHTC-SAS). Pages: 144 – 148.

Szakacs-Simon, P.; Moraru, S.A.; Perniu, L. 2013. Android application developed to extend health monitoring device range and real-time patient tracking. *IEEE 9th International Conference on Computational Cybernetics* (ICCC). Pages: 171 – 175.

Jaehyun Lee; Junhyung Ahn; Yoonji Lee; Chulyun Kim. 2013. Implementation of Semantic Directory Service on Image Gallery Application of Mobile Devices. *International Conference on Information Science and Applications* (ICISA). Pages: 1 – 2.

Mikhaylov Dmitry, Zhukov Igor, Starikovskiy Andrey, Kharkov Sergey, Tolstaya Anastasia, Zuykov Alexander. 2013. Review of Malicious Mobile Applications, Phone Bugs and other Cyber Threats to Mobile Devices. *Proceedings of 2013 5th IEEE International Conference on Broadband Network & Multimedia Technology* (5th IEEE IC-BNMT 2013), November 17-19th 2013 Guilin, China. Pages 302-305.

Zhukov Igor, Mikhaylov Dmitry, Starikovskiy Andrey, Dmitry Kuznetsov, Tolstaya Anastasia, Zuykov Alexander. 2013. Security Software Green Head for Mobile Devices Providing Comprehensive Protection from Malware and Illegal Activities of Cyber Criminals. *International Journal of Computer Network and Information Security* (IJCNIS) Vol. 5, No. 5, April 2013. Pages 1-8.

Patrick Northcraft. 2014. *Android: The Most Popular OS in the World*. AndroidHeadlines.com, February 15, 2014. URL: http://www.androidheadlines.com/2014/02/android-popular-os-world.html.

Shahriar, Hossain; North, Sarah; Mawangi, Edward. Testing of Memory Leak in Android Applications. 2014. *IEEE 15th International Symposium on High-Assurance Systems Engineering* (HASE). Pages: 176 – 183.

Sbirlea, D.; Burke, M.G.; Guarnieri, S.; Pistoia, M.; Sarkar, V. 2013. Automatic detection of inter-application permission leaks in Android applications. *IBM Journal of Research and Development* (Volume: 57, Issue: 6), Nov.-Dec. 2013. Pages: 10:1 - 10:12.

Taenam Cho; Jae-Hyeong Kim; Hyeok-Ju Cho; Seung-Hyun Seo; Seungjoo Kim. 2013. Vulnerabilities of android data sharing and malicious application to leaking private information. *Fifth International Conference on Ubiquitous and Future Networks* (ICUFN). Pages: 37 – 42.

Mulliner, C.; Liebergeld, S.; Lange, M.; Seifert, J.-P. 2012. Taming Mr Hayes: Mitigating signaling based attacks on smartphones. *42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (DSN). Pages: 1 – 12.

Stirparo, Pasquale; Fovino, Igor Nai; Taddeo, Marco; Kounelis, Ioannis. 2013. In-memory credentials robbery on android phones. *World Congress on Internet Security* (WorldCIS). Pages: 88 – 93.

Woo Bong Cheon; Keon il Heo; Won Gyu Lim; Won Hyung Park; Tai Myoung Chung. 2011. The New Vulnerability of Service Set Identifier (SSID) Using QR Code in Android Phone. *International Conference on Information Science and Applications* (ICISA). Pages: 1 – 6.

Miller, C. 2011. Mobile Attacks and Defense. IEEE Security & Privacy, (Volume: 9, Issue: 4), July-Aug. 2011. Pages: 68 – 70.

Barrere, M.; Hurel, G.; Badonnel, R.; Festor, O. 2013. A probabilistic cost-efficient approach for mobile

security assessment. *9th International Conference on Network and Service Management* (CNSM). Pages: 235 – 242.

Cozzette, A.; Lingel, K.; Matsumoto, S.; Ortlieb, O.; Alexander, J.; Betser, J.; Florer, L.; Kuenning, G.; Nilles, J.; Reiher, P. 2013. Improving the security of Android inter-component communication. *IFIP/IEEE International Symposium on Integrated Network Management* (IM 2013). Pages: 808 – 811.

Al-Saleh, M.I.; Espinoza, A.M.; Crandall, J.R. 2013. Antivirus performance characterization: system-wide view. *IET Information Security*, (Volume: 7, Issue: 2). Pages: 126 – 133.

Bläsing, T.; Batyuk, L.; Schmidt, A.-D.; Camtepe, S.A.; Albayrak, S. 2010. An Android Application Sandbox system for suspicious software detection. 5th *International Conference on Malicious and Unwanted Software (MALWARE)*. Pages: 55 – 62.

Pieterse, H.; Olivier, M.S. 2013. Security steps for smartphone users. *Information Security for South Africa*. Pages: 1 – 6.

Fu-Hau Hsu; Min-Hao Wu; Chang-Kuo Tso; Chi-Hsien Hsu; Chieh-Wen Chen. 2012. Antivirus Software Shield Against Antivirus Terminators. *IEEE Transactions on Information Forensics and Security*, (Volume: 7, Issue: 5). Pages: 1439 – 1447.

Chia-Mei Chen, Ya-Hui Ou. 2011. Secure mechanism for mobile web browsing. IEEE 17th International Conference on Parallel and Distributed Systems (ICPADS). Pages: 924 – 928.

Kasama, T.; Yoshioka, K.; Inoue, D.; Matsumoto, T. 2012. Malware Detection Method by Catching Their Random Behavior in Multiple Executions. *IEEE/IPSJ 12th International Symposium on Applications and the Internet* (SAINT). Pages: 262 – 266.

Dube, T.E.; Raines, R.A.; Grimaila, M.R.; Bauer, K.W.; Rogers, S.K. 2013. Malware Target Recognition of Unknown Threats. *IEEE Systems Journal*, (Volume: 7, Issue: 3). Pages: 467 – 477.

Hsiang-Yuan Hsueh; Kun-Fu Huang; Wei-Ming Wu; Chih-Lin Li. 2013. Evaluating the risk of Android application: Design and implementation of static analysis system. *6th International Conference on Advanced Infocomm Technology (ICAIT)*. Pages: 236 – 237.

Batyuk, L.; Herpich, M.; Camtepe, S.A.; Raddatz, K.; Schmidt, A.-D.; Albayrak, S. 2011. Using static analysis for automatic assessment and mitigation of unwanted and malicious activities within Android applications. *6th International Conference on Malicious and Unwanted Software* (MALWARE). Pages: 66 – 72.

Yan Ma; Sharbaf, M.S. 2013. Investigation of Static and Dynamic Android Anti-virus Strategies. Tenth International Conference on Information Technology: New Generations (ITNG). Pages: 398 – 403.

William Enck, Peter Gilbert, Byung-Gon Chun. 2010. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. Pages: 1 – 15.

Android operating system architecture. 2011. Android-shark. URL: http://android-shark.ru/arhitektura-operatsionnoy-sistemyi-android.

Michael Lawrence. 2012. *Tutorial How to Install Games on Galaxy Mini.* Tutorial For Android. URL: http://tutorialfor-android.blogspot.ru/2012/05/tutorial-how-to-install-games-on-galaxy.html.

Trojan Android.SmsSend.186.origin. 2012. IT-sector. URL: http://it-sektor.ru/android.smssend.186.origin-troyan-otpravlyauschiyi-platnye-sms-soobscheniya.html.

Mobile Spy. 2013. Top 10 Spy Software. URL: http://top10spysoftware.com/review/mobilespy.