Journal of Mobile, Embedded and Distributed Systems, vol. VII, no. 1, 2015
ISSN 2067 – 4074

Risk Assessment Model for Mobile Malware

George STANESCU
Department of Economic Informatics and Cybernetics
The Bucharest University of Economic Studies
ROMANIA
georgestanescu90@gmail.com

Abstract: The mobile technology is considered to be the fastest-developing IT security area. Only in the last year security threats around mobile devices have reached new heights in terms of both quality and quantity. The speed of this development has made possible several types of security attacks that, until recently, were only possible on computers. In terms of the most targeted mobile operating systems, Android continues to be the most vulnerable, although new ways of strengthening its security model were introduced by Google. The aim of this article is to provide a model for assessing the risk of mobile infection with malware, starting from a statistical analysis of the permissions required by each application installed into the mobile system. The software implementation of this model will use the Android operating system and in order to do so, we will start by analyzing its permission-based security architecture. Furthermore, based on statistical data regarding the most dangerous permissions, we build the risk assessment model and, to prove its efficiency, we scan some of the most popular apps and interpret the results. To this end, we offer an overview of the strengths and weaknesses of this permission-based model and we also state a short conclusion regarding model's efficiency.

Key-Words: *Mobile Security, Malware, Risk Assessment*

1. Introduction

The huge popularity of the actual mobile platforms can be motivated by feature-rich devices, as well as by the large number of applications that have different practical uses in our everyday life. People use increasingly often portable devices, such as smartphones, to collect, store and manage personal data. This data can be highly privacy-sensitive and this raises the question in what measure the mobile operating systems are able to protect user data. The rapid growth in smartphone and tablet usage has led to the inevitable rise in targeting of these devices by cybercriminals. The exponential growth in Android devices, the largely unregulated Android third party application markets and the fact that Android has a rather open architecture produced a sharp rise in malware targeting this platform, attracting 98.05% of known malware in 2013 [1] and the specialists do not expect this trend to change in near future. According to the same source, the majority of malicious mobile applications are targeted primarily at stealing money and, secondly, at stealing personal data.

The concept of mobile malware is strongly related to the one of computer virus (or computer malware) from which it evolved as a result of the increasing number of mobile devices. In order to control how apps access sensitive devices and data stores, most Unix-based mobile systems including Android use a system of permissions. This model ensures security and privacy by mediating the access between sensitive data stored into the device (contacts, banking accounts, social accounts, emails etc.) and device functionalities (network access, GPS, external storage, Bluetooth etc.). Each app that needs to access an asset (phone feature or personal information) has to explicitly request it from the platform. This is done by defining a related permission in the Android Manifest xml file bundled with the app, and then, users have the possibility to see and explicitly grant the permission as a precondition to installing the app. The permissions that are being declared in the manifest file of an application can be interpreted as a first security indicator before applying other

antimalware solutions that are often computationally expensive.
The purpose of this research is to define a security model capable of assessing the risk of malware infection on mobile devices by analyzing the permissions that were granted to applications. The model will be built starting from a statistical analysis of the most used permissions both for benign applications and for malware. Once the model is built, we create an Android application capable of retrieving the permissions for the installed apps and making predictions regarding the risk of malware infection. In order to test the efficiency of the model, we analyze several apps that have been already detected as benign or malicious by trusted antimalware solutions. We then expose the conclusions of this research and offer an overview regarding the strengths and weaknesses of this permission-based risk model.

2. Mobile Malware

Malware (malicious software) is any software used to disrupt computer operation, gather sensitive information, or gain access to private computer systems. It can appear in the form of code, scripts, active content, and other software. "Malware" is a general term used to refer to a variety of forms of hostile or intrusive software [2]. The most important malware categories are: computer viruses, ransomware, spyware, adware, scareware. The term of mobile malware refers to a malicious software that targets mobile devices having pretty the same disruptive impact over mobile terminal as in the case of traditional malware over computers.
At the time of its first occurrence, mobile malware had limited effects on mobile platforms. The first mobile malware named Cabir has been detected on June 14, 2004. It was an application for Symbian OS that acted as a worm for mobile phones which spread via Bluetooth. The creators wanted to build a proof-of-concept virus code for non-standard operating systems and applications. By the beginning of 2005, the main types of mobile malware had evolved, and were used by virus writers: worms that spread via smartphone protocols and services, vandal Trojans that install themselves to the system by exploiting Symbian design faults, Trojans designed for financial gain [3]. The first Android malware was recorded in September 2008, although it was not to cause damage. The project was developed by the security research group Blitz Force Massada from the University of Electronic Science and Technology of China (UESTC) and was a collection of more than 30 attacks executed by four different modules [4]. In November 2009, Retina-X Studios announced the world's first professional spy software for Android, Mobile Spy, which could silently monitor devices via web browser, calls, text messages, photos, videos, GPS locations, and even visited URLs from the device [4]. A major feature of Mobile Spy is that the user is not aware that he is being monitored by this application; it runs in a stealth mode, in the background, without a visible icon.
It has been 10 years since the first recorded mobile malware, but it is only within the past few years that it has become a true threat to end users [5]. There are a lot of reasons why cybercriminals tend to focus on mobile devices. One of the most important is that computers do not have a built-in billing system while mobile phones do and that increases the potential of earning opportunities for hackers and virus authors. The actual trend indicates that mobile devices are no longer just a playground for cybercriminals, but have become valuable tools for financial profit. The theft of personal information, the unintentional sending of short messages to premium-rate numbers without user's approval and the ability for infected smartphones to be remotely accessed for other malicious purposes are the most

ISSN 2067 – 4074

popular attacks. Over the last year, the number of mobile malware modifications designed for phishing, the theft of credit card information and money increased by a factor of 19.7 [6]. Some of the most sophisticated attacks exploit the fact that smartphones are rarely shut down making the malware far more reliable since all its resources are always available and ready for continuous processing.

Although most attacks occur on the Android platform, the major types of malware are found on all mobile platforms. Most times, the same type of attack can be used on systems with similar architectures. For example, in UNIX permission-based architecture similar vulnerabilities can be easily exploited. The security differences will be made by the way in which the operating systems react to the same type of attack and also by the way in which application markets detect the potentially dangerous apps. On the other hand the cybercriminals create applications that include hidden (obfuscated) malicious functionality in an attempt to avoid detection included in the vendor's application vetting process. Inevitably, Google's platform has become a much greater target for mobile malware writers than iOS because, unlike Apple, it does not employ a walled garden policy with regard to apps. It's also significant that Android has a large proportion of the mobile market—up to 79% in 2013 [5].

Infection Strategy

The initial introduction of a malware into a system is the essential step that must always succeed for the attack to be performed. There are various ways of infecting the mobile devices but the most important source of malware is the third party application markets. Third party developers creating applications for Android can submit their applications to official Android Market but also to alternative Android markets from where users can download and install them. While this provides a high level of availability of unique, specialized or general purpose applications, it also gives rise to serious security concerns. The official market benefits from advanced security mechanisms able to detect and eliminate most of the malware. In contrast, the markets provided by various suppliers may have an ineffective mechanism to verify the application. In some cases these vetting tools can even be absent, and, in other cases, the attackers that publish dangerous apps know the security mechanisms and even manage to get through any security checks. Other strategies for infecting or spreading malware are targeting wireless communication (through MMS, Bluetooth or email), Wi-Fi networks, OS vulnerabilities, SMS, Device-to-PC synchronization and even removable storage used by portable devices.

Types of mobile malware

Regardless the mobile platform they use, the mobile malware with a potential privacy impact can be fit into one of the following categories [7]:

- Tracking/Surveillance, which refers to monitoring user's activity (e.g. using device's sensors).
- Interception/Eavesdropping, which refers to illegal interception of communications and is applicable to all communication data types (including external communication data, e.g. call log).
- Profiling, which refers to user activity monitoring, but for advertising purposes.
- Phishing, which refers to tricking the user to disclose credentials.
- Personal Information Disclosure, which refers to the disclosure of all other types of personal information, which do not fall in the other four threat types (e.g. documents, multimedia files, etc.).

3. Android Security Model

Android, at its core, relies on one of the security features provided by Linux kernel: running each application as a separate process, having its own set of data structures and preventing other processes from interfering with its execution. At the application layer, Android uses fine-grained permissions to grant or revoke the interaction between applications (components) and other applications (components) or critical assets. User approval is required before an application can get access to critical operations. Applications explicitly request the permissions they need in order to execute successfully. By default, no application has permission to perform any operations that might adversely impact other applications, the user's data, or the system. Examples of such operations include sending SMS messages, reading contact information, and accessing the Web. On the other hand, playing music files or viewing pictures do not fall under such operations, and, thus, an application does not need to explicitly request permissions for these. Application-level permissions provide a means to get access to restricted content and APIs.

Each Android application runs in a separate sandbox called Dalvik Virtual Machine (VM). Although this sandbox does not enforce security, it is optimized for running on embedded devices efficiently, with a small footprint. It is possible to break out of this sandbox and, thus, it cannot be relied on to enforce security. Android permission checks are not implemented inside the Dalvik VM but, rather, inside the Linux kernel code and enforced at runtime [8].

Similar to other UNIX security approaches, the access to low-level facilities is provided through user ID and group ID enforcement, whereas additional fine-grained security features are provided through the permissions declared in the Android Manifest file.

When installing new application, Android assigns it a unique user id (UID) and a group id (GID). Each installed application has a set of data structures and files that are associated with its UID and GID. Permissions to access these structures and files are allowed only to the application itself (through its ID) or to the super-user (root). However, other applications do not have elevated super-user privileges and, thus, cannot access other applications' files.

Android's Manifest Permissions

Through the sandbox mechanism, the Linux kernel prevents applications from accessing other applications' data or user information, or from performing critical operations such as making phone calls, accessing the Internet, or receiving SMS messages. If an application needs to perform the aforementioned operations, read user's personal data (e.g., contacts), or communicate with other applications, the application needs to specifically request these permissions (this security model is called MAC). Applications declare these permissions in their configuration file (Android Manifest.xml), as shown in the figure below:

```
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.android.app.detector" >
    <uses-permission android:name="android.permission.GET_TASKS" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.READ_SMS" />
    .....
</manifest>
```

Figure 1. Declaring permission for an Android application

When an application is installed, Android prompts the user to either allow or reject requested permissions. The user cannot allow certain permissions and reject the others. The application requests a set of permissions, and the users either approve or deny all of them (see Figure 1). Once the user has approved these permissions, Android kernel will grant access to the requested operations or allow interaction with different components. Once the user has approved permissions, he cannot

ISSN 2067 – 4074

revoke them. The only way to remove the permissions is to uninstall the application. This is because Android does not have the means to grant permissions at runtime. One of the arguments Google decided not in implement this feature was that this will lead to less user-friendly applications. Android permissions are also displayed to the end-user when downloading applications from the official application market. If the user just downloads the installer file (having the extension .apk), a warning about security implications will only be displaced during runtime. When an application tries to perform an operation for which it hasn't the suitable permission declared, Android will typically throw a Security Exception back to the application.

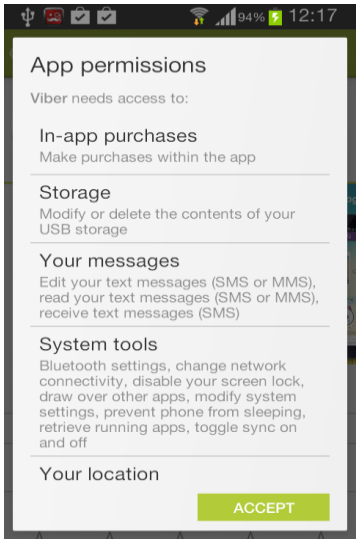

Figure 2. Dangerous permission groups being shown before installing an Android application

Manifest permission system file contains all the predefined permissions used by Android system, but applications may also define and enforce its custom permissions. To enforce custom permissions, developers must first declare them in AndroidManifest.xml using one or more <permission> tags.

To allow certain low-level permissions, the system needs to map the permission string to the group that can access the functionality. For example, if an application requests access to read SMS (READ_SMS), after user's approval at the time of the installation, Android will add the application to the corresponding group. An application needs to be a member of this group in order to have access to the "read SMS" feature.

The potential risk implied by using a certain permission is indicated by a permission level attached to that permission. This indicator warns about the procedure the system should follow when determining whether or not to grant the permission to an application requesting it [9]. There are four permission levels defined in Android:

- normal – it is the default one and is granted automatically by the system, but the user can still review it during the install time;
- dangerous – it should be explicitly granted by user because it allows the application to perform certain operations that use data in a way that can impact the user in a negative manner.
- signature – it is granted only if the application are signed with the same certificate as the application that declared the permission.
- signatureOrSystem – it is granted only to applications that are in the Android system image or that are signed with the same certificates as those in the system image.

As the permissions in the first protection level are always granted and those in the last two categories are granted by the system, we will focus on the analysis of dangerous permissions as they often cause user privacy or financial losses. A good example of granting dangerous permissions is the usage of ACCESS_FINE_LOCATION and ACCESS_COARSE_LOCATION that lets an app access the user's location that might be a privacy concern if the app does not need such a capability. Also, granting an app READ_SMS and SEND_SMS permissions lets an application send and receive SMS which might be a privacy issue [10]. Moreover, dangerous

permissions that let apps read logs, calendar, user accounts or phone state can be easily used by malware (botnets and Trojans) to produce harmful results on a command of their remote owners. In this category we can include the following permissions:

- READ_PHONE_STATE
- PROCESS_OUTGOING_CALLS
- READ_CALENDAR
- READ_CONTACTS
- READ_USER_DICTIONARY
- USE_CREDENTIALS

.

4. Permission-Based Risk Assessment Model

Permissions are the core of Android application security. That is why awareness and understanding of the permission model is essential both for developers and users of an Android device. Most smartphone users do not possess the technical knowledge to make the distinction between benign and potentially dangerous apps, and they often accept the permissions required by an application without even reading their descriptions. To give users a better insight of the impact that granting permissions could have on the security of mobile device and to critical information it manages, we constructed a risk analysis model based on the permissions granted to each application. The proposed model starts from a series of statistical observations on the permissions used for benign applications, as well as for malicious software. An important aspect to mention is that all the permissions requested by a malicious applications are not necessarily required to do their processing but we won't take this aspect into account because it would be too difficult to decide which permission are really used by an application and which don't. Also, when building the risk model, we take advantage of the fact that when a Trojan is attached to a legitimate application, the resulting application requires the permissions of the original application in addition to the permissions it requires for its own purposes [11]. This translates to a significant deviation in the amount of permissions requested by malware. According to statistics from [11] and [12] we built a ranking of the first 100 common used permissions for each of the two cases. The original two datasets were gathered by analyzing a large number of applications from four different application markets, consisting of multiple features, ranging from developer identity to requested permissions. They cover the whole spectrum of applications could even act suspiciously, break rules or contain malware and were included with no regard for popularity or usefulness. Few of the results are shown in table below and are expressed as percentages:

Permissions	Benign Usage (%)	Malicious Usage (%)
INTERNET	85	96
READ_PHONE_STATE	34	84
ACCESS_NETWORK_STATE	56	75
READ_HISTORY_BOOKMARKS	1	58
WRITE_HISTORY_BOOKMARKS	1	57
ACCESS_WIFI_STATE	10	54
WRITE_EXTERNAL_STORAGE	36	53
VIBRATE	20	53
RECEIVE_BOOT_COMPLETED	9	47
.....	.....	.....
WRITE_GSERVICES	0.01	0.95
UPDATE_DEVICE_STATS	0.00	0.95
SUBSCRIBED_FEEDS_WRITE	0.00	0.95
SIGNAL_PERSISTENT_PROCESSES	0.00	0.95

Figure 3. Usage percentages for Android permissions in benign and malicious apps

As we can notice, the permission for requesting full network access is the most used in both cases. However, most of the permissions have different percentages for the two cases, fact that indicates that some permissions are preferred to be used in mobile attacks (READ PHONE STATE, ACCESS WIFI STATE etc.) and some are hard to be exploited(CAMERA, GET TASKS).

An interesting fact is that permissions with the highest percentage in any of the two cases fall into the category of dangerous permissions (their permission level is dangerous).

Also, one alarming thing is the excessive use of a permission for creating malware, while the same permission is rarely used

ISSN 2067 – 4074

for benign applications. For example, the READ HISTORY BOOKMARKS and WRITE HISTORY BOOKMARKS permissions are used only by one percent of the benign apps, while they are widely used for producing malware with a huge percentage (58 and 57 respectively).
The next step in building the privacy model is to use the percentages as weight values and combine statistical analysis of both malicious and benign apps. In order to do so, we perform a normalization process by the most used permission in each case (its value will be set to 100%). The normalization for benign usages of a permission i (NBU_i) is computed using the following formula, where BU_i is the benign usage for that permission and BU_{max} is the largest weight of all benign usages:

$$NBU_i = \frac{BU_i}{BU_{max}} \cdot 100 \quad \textbf{(1)}$$

Using a similar approach the normalized malicious usage (NMU_i) is computed for each permission that is being analyzed:

$$NMU_i = \frac{MU_i}{MU_{max}} \cdot 100 \quad \textbf{(2)}$$

The results of the normalization process are shown in the next figure:

Permissions	Normalized BU (%)	Normalized MU (%)
INTERNET	100	100
READ_PHONE_STATE	40	87.5
ACCESS_NETWORK_STATE	65.88	78.13
READ_HISTORY_BOOKMARKS	1.18	60.42
WRITE_HISTORY_BOOKMARKS	1.18	59.38
ACCESS_WIFI_STATE	11.76	56.25
WRITE_EXTERNAL_STORAGE	42.35	55.21
VIBRATE	23.53	55.21
RECEIVE_BOOT_COMPLETED	10.59	48.96
.....	.....	.....
WRITE_GSERVICES	0.01	0.99
UPDATE_DEVICE_STATS	0	0.99
SUBSCRIBED_FEEDS_WRITE	0	0.99
SIGNAL_PERSISTENT_PROCESSES	0	0.99

Figure 4. Normalized usage percentages for Android permissions in benign and malicious apps

Once the normalization is done, the next step is to build a permission vector having the size equal to the number of permissions from the model (100 in this case). Any element of the vector may have the value of 0 or 1. For each application that is being analyzed we extract the related permissions (excluding custom permission because they weren't introduced into the model) and populate permission vector. The elements that match application's permissions will all have the value of 1 and any other element of the permission vector will be 0. Then, using the permission vector (PV), the normalized malicious (NMU) and benign (NBU) percentages we compute two indicators: malware score (MS) and normality score (NS) using the following formulas (k=100 analyzed permissions):

$$MS = \sum_{i=1}^{k}(PV_i \cdot NMU_i) \quad \textbf{(3)}$$

$$NS = \sum_{i=1}^{k}(PV_i \cdot NBU_i) \quad \textbf{(4)}$$

Once we obtain the two values, we can define the Permission Trust Index (PRI) as:

$$PTI = \frac{NS}{MS} \quad (5)$$

To interpret the result of this indicator we introduce several risk levels: non-existing, negligible, moderate, significant and maximum. According to the value obtained for PTI, an application is indicated as extremely dangerous if PTI is less than 0.5 or very close to this value. Also, an application can be considered as trusted (clean) if PTI is closed to 1 or over this value. A PTI value between 0.5 and 0.7 indicates an application with a significant risk level, while a value between 0.7 and 0.85 shows a moderate risk level. A negligible risk level corresponds to a trust index greater than 0.85 but less than 0.95.

5. Model Implementation in Android

To test this risk model, we developed an application (called Malware Detector) able to analyze the applications installed on any Android device and extract their permissions (only the default permissions defined by Android). The normalized values regarding the malicious and benign usages of the 100 permissions described

in the previous chapter were stored in the application. For each application installed in the Android system, the Malware Detector app will determine the permission vector and will compute the two scores (malware score and normality score). After obtaining the Permission Trust Index we can decide the risk level for each analyzed application and we can show an application list ordered by the value of this indicator.

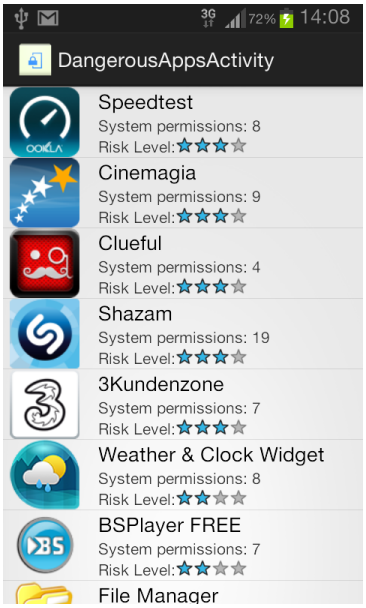

Figure 5. The list of installed apps ordered by their risk level derived from Permission Trust Index

To convince the users about the security risk they expose by using apps with a low trust index, when selecting an app from the list we also provide a short description of the permissions used by that app. Moreover, starting from the results presented in the previous chapter we have grouped the analyzed permissions in several levels of risk (in a similar manner to that used for the installed applications: non-existing, negligible, moderate, significant and maximum). The permissions having a malicious usage (MU) greater than 70% have a maximum risk level (such as INTERNET, READ PHONE STATE and ACCESS NETWORK STATE) permissions at a rate between 20% and 70% fall into the significant risk category (READ/WRITE HISTORY BOOKMARKS, ACCESS WIFI STATE etc.), those with the rate between 4% and 20% are moderate risk permissions, while those with the rate lower than 4% are of negligible risk. Permissions that are not taken into account by this risk model are not considered security threats and, therefore, their risk level is considered to be nonexistent. Based on this classification, the permissions associated with each analyzed applications will be grouped by their degree of risk, so the user can view the permissions in the natural order of the impact they have on system security.

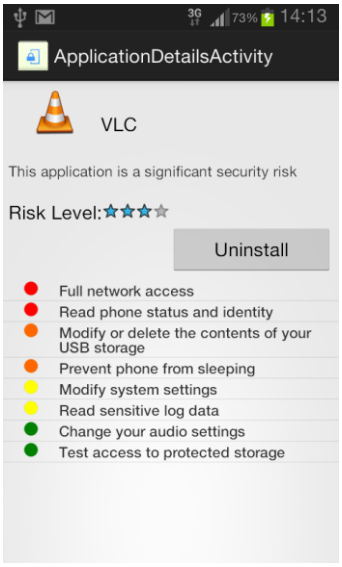

Figure 6. The permissions used by an application ordered by their risk level

In order to prove the efficiency of this permission-based risk model, we analyze two apps considered to be benign and two apps known as being malicious according to reliable sources. Malware Detector app will determine the Permission Trust Index for WinZip and Google Drive as benign apps and GoldDream and Pjapps.A as malware. Pjapps.A is a Trojan that runs on Android mobile devices and attempts to send sensitive information to a remote server and could also send SMS messages to other phones and allow limited remote control of the mobile device [13]. GoldDream is a Trojan that disguises itself on certain marketplaces as game software

ISSN 2067 – 4074

and in order to compromise the device installs and executes new packages, makes arbitrary phone calls, sends arbitrary SMS messages and uninstalls packages [14].

The confidence score obtained for Google Drive was 0.72, which indicates a moderate security risk. Although this application is known to be safe, the relative low value for the trust index is because this application uses some permissions that are also exploited by malicious applications. On the other hand, WinZip has a trust level of 0.82. Although, it uses two permissions with maximum security risk (INTERNET and ACCESS NETWORK STATE) and two permission with significant security risk (WRITE EXTERNAL STORAGE and MOUNT UNMOUNT FILESYSTEMS), the other five permissions are negligible security risks.

Malware Detector computed a trust score of 0.44 for Pjapps.A app, indicating a severe security risk. This result is justified by the fact that the first eight permission in the malicious usage ranking are also being used in this application. It is a bit more difficult to detect the GoldDream malware because the trust index is 0.54 in this case. Although the result is close to the 0.5 limit for severe security risk, the model rather indicates a significant risk level. However, if we analyze the permissions that are being used we can conclude that we have to deal with a very dangerous application (all three most dangerous permissions are used, eight of them have significant risk level, three have moderate risk and only one has a negligible risk).

5. Conclusions

In this paper, we proposed a risk assessment model for mobile malware using a permission-based approach. In order to decide the risk level for each application we introduced the Permission Trust Index that was statistically determined based on the normalized percentages of the most used permissions for both malicious and benign applications. In addition, we introduced this model in an Android application in order to prove its efficiency. After we ran the model for two benign and two malicious apps we concluded that its security goals were achieved, although it also has some limitations. Moreover, a more detailed view was built, in order to inform the user about each permission an application is using and we also grouped the analyzed permissions to five security level (maximum, significant, moderate, negligible and non-existent) according to the same statistical researches.

An important thing worthy to be mentioned about this model is that this risk assessment method is based on a simple analysis of application's permissions. Therefore, it can be used rather as a first indicator for signaling the security impact an application could have on a mobile device. Its main advantage is that it does not require considerable resources in order to assess the trust level of the installed applications. Therefore, it would be preferable that besides adopting this model, to also use some other tools able to perform an in-depth analysis of the applications (by searching APIs which are called in application, detecting malware signatures, etc.).

Potential limitations

The validity of this risk assessment method largely depends on the accuracy of the statistics regarding the permissions used by two application categories (malicious and benign). That is why, it is very important to collect application from multiple markets, not only the official market. According to [11], the percentages we used in our model were the result of the analysis of four Android markets (Google Play, Amazon Appstore, F-Droid and SlideMe) and several malicious repositories: Contagio Mobile Dump, Symantec's Threat Explorer database and F-Secures Threat Description database. However, the

number of clean applications dominates the one of malicious ones and the accuracy in detecting a malicious application may be decreased compared to a clean app.
One other potential inconvenience of the model is the relatively high false positive detection rate. For some of the known malware we noticed that the trust index was over the limit we proposed, as it is the case of GoldDream virus. Consequently, a solution to improve the model is to analyze the occurrences of two, three or more permissions, to identify a pattern of the most popular permission groups used in mobile attacks. This could slow the algorithm that computes the confidence score, but would eliminate the confusion for applications whose confidence indicator are close to a maximum risk threshold.
Another possible extension to this project is to use the same methods we have used for our analysis on the third-party permissions, although Google Play does not list any third-party permissions from applications not created by Google, which would make the permission collecting process more difficult to achieve.

Acknowledgement

Parts of this paper were presented at The 7th International Conference on Security for Information Technology and Communications (SECITC 2014), Bucharest, Romania, 12-13 June 2014.

References

[1] Kaspersky Security Bulletin, Kaspersky Lab, Global Research and Analysis Team, 2013;
[2] Malware page on Wikipedia, http://en.wikipedia.org/wiki/Malware, accessed on May 07, 2014;
[3] A. Gostev, Mobile Malware Evolution: An Overview, Part 1, http://www.securelist.com/en/analysis?pubid=200119916, accessed on May 07, 2014;
[4] C.A. Castillo, Android Malware Past, Present and Future, Mobile Security Working Group, McAfee, 2011;
[5] V. Svajcer, Sophos Mobile Security Threat Report, 2014;
[6] V. Chebyshev, R. Unuchek, Mobile Malware Evolution: 2013 - Methods and techniques, https://www.securelist.com/en/analysis/204792326/Mobile_Malware_Evolution_2013 accessed on May 07, 2014;
[7] A. Mylonas, A. Theoharidou, D. Gritzalis, Assessing privacy risks in Android: A user-centric approach, Dept. of Informatics, Athens University of Economics & Business;
[8] A. Dubey, A. Misra, Android Security – Attacks and Defenses, Taylor & Francis Group, LLC, 2013;
[9] Android Developers page – permission topic, http://developer.android.com/guide/topics/manifest/permission-element.html, accessed on May 07, 2014;
[10] P. O. Rai, Android Application Security Essentials, Packt Publishing, 2013;
[11] T. Boksasp, E. Utnes, Android Apps and Permissions: Security and Privacy Risks, Norwegian University of Science and Technology, 2012;
[12] C. Y. Huang, Y. T. Tsai and C. H. Hsu, Performance Evaluation on Permission-Based Detection for Android Malware, National Taiwan Ocean University;
[13] Microsoft Malware Protection Center, TrojanSpy:AndroidOS/Pjapps.A definition, http://www.microsoft.com/security/portal/threat/encyclopedia/entry.aspx?Name=TrojanSpy:AndroidOS/Pjapps.A#tab=1, accessed on May 07, 2014;
[14] Android GoldDream definition provided by Symantec, http://www.symantec.com/security_response/writeup.jsp?docid=2011-070608-4139-99, accessed on May 07, 2014