International Journal of Scientific Engineering and Applied Science (IJSEAS) – Volume-2, Issue-3,March 2016
ISSN: 2395-3470
www.ijseas.com

Techniques that Allow Hidden Activity Based Malware on Android Mobile Devices

Milan Oulehla[1], David Malanik[1]
[1] Faculty of Applied Informatics, Tomas Bata University in Zlín, Zlín, Czech Republic

Abstract
Currently, number of Android based mobile devices has been constantly increasing. In 2014, Google had over 1 billion active Android users [14]. Android has become the most popular operating system in the world. However, the Android operating system is not only popular with its users but also with malware programmers. The main issue concerning such widespread operating system is not the GUI and reliability but security. This paper tries to open a different perspective on the Android security issue. While the majority of already published articles describes techniques allowing malware detection, this article is focused on malware from the attacker's perspective and tries to shed light on the techniques allowing functioning of hidden Activity based malware on Android mobile devices. Specifically, the text describes a technique based on camouflage of an Activity that allows running of BroadcastReceiver which has been waiting in background and responds to events such as receiving an SMS, pushing the home button, Wi-Fi connection etc. This technique is important for malware aimed at devices with Android version 3.1 or higher. For safety reasons, these Android versions do not allow running of BroadcastReceiver without an Activity. The article describes how to avoid this protection.
Keywords: *Android; Activity; BroadcastReceiver; camouflage; malicious activity; malicious software; mobile devices; mobile malware; mobile virus; smart phones; tablet.*

1. Introduction

Popularity of mobile devices such as smartphones, tablets and wearable hardware is continuously growing in both private and commercial sector. This fact can be illustrated by Fig. 1 and Table 1.

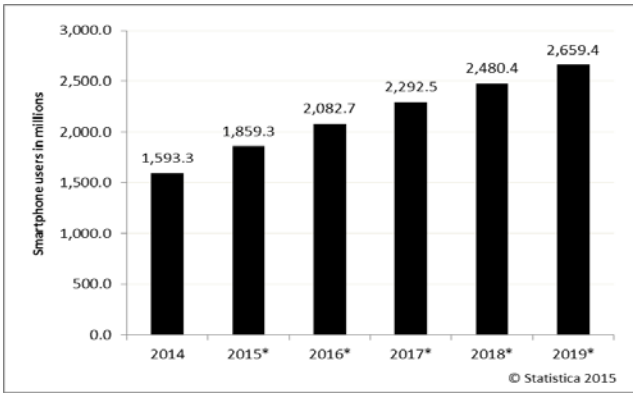

Fig. 1 Smartphone users in millions [16]

Table 1: Internet of Things Units Installed Base by Category [17]

Category	2013	2014	2015	2020
Automotive	96.0	189.6	372.3	3,511.1
Consumer	1,842.1	2,244.5	2,874.9	13,172.5
Generic Business	395.2	479.4	623.9	5,158.6
Vertical Business	698.7	836.5	1,009.4	3,164.4
Grand Total	**3,032.0**	**3,750.0**	**4,880.6**	**25,006.6**

Nevertheless, amount of mobile malware is increasingly growing too, as can be seen in Fig. 2.

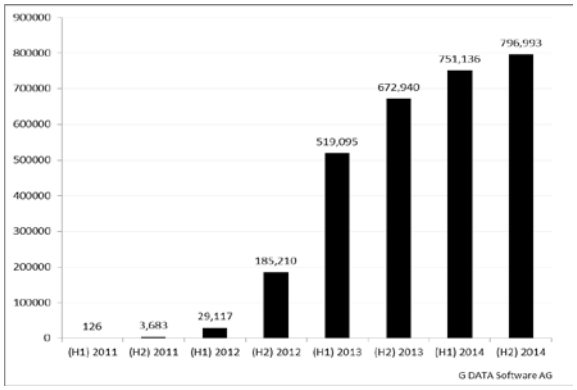

Fig. 2 Android malware samples / half yearly [18]

These reports show that the number of malware samples is alarming and in correlation with a huge number of mobile devices brings high probability of attacks in the near future. Thus, there are many potential targets and a lot of applicable malware.

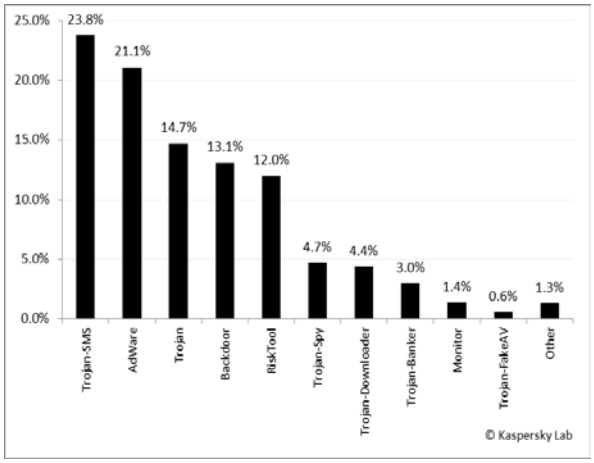

Fig. 3 Distribution of mobile threats by type [19]

Many types of malware are focused on Android mobile devices (see Fig. 3); however, the basic functionality of many malware kinds is similar. They try to mask themselves; camouflaging is necessary for any successful malware. Moreover, there is another alarming trend: a massive increase in the number of Trojans aimed directly at mobile banking applications. This trend is represented in Fig. 4 below.

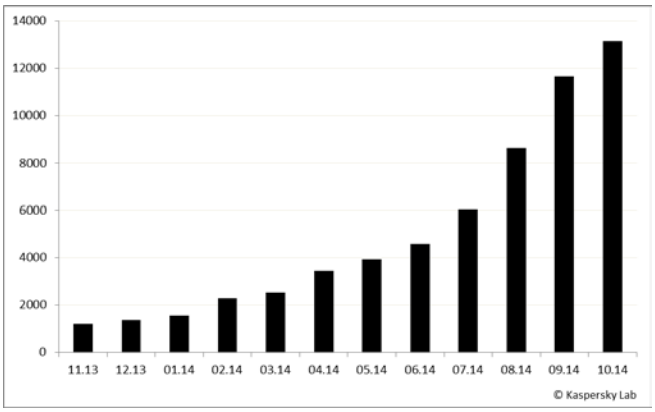

Fig. 4 Number of mobile banking Trojans in the Kaspersky Lab collection [19]

The research paper tries to contribute to the security improvement in the field of malware hiding.

2. The theoretical background of malware for the Android platform

In order to understand the development of malware for the Android platform better, it is necessary to explain some essential terms used in this field. The first of them is the Activity class. ”An Activity represents a single screen with a user interface. For example, an email app might have one activity showing a list of new emails, another activity to compose an email and another for reading emails. Although the activities work together to form a cohesive user experience in the email app, each one is independent of the others. As such, a different app can start any of these activities (if the email app allows it). For example, a camera app can start the activity in the email app that composes a new email, in order for the user to share a picture” [1]. In short, Activity is a GUI and application logic of one screen. It has various methods such as onCreate [2], onStart [3], onResume [4] etc. These methods can be created by Java language which is a native language for the Android operating system. Graphical user interface of the Activity is developed by XML.

”A broadcast receiver is a component that responds to system-wide broadcast announcements. Many broadcasts originate from the system - for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured. Apps can also initiate broadcasts - for example, to let other apps know that some data has been downloaded to the device and is available for them to use. Although broadcast receivers don't display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs” [1] Briefly, the BroadcastReceiver is a class that does not have a user interface and which is able to run silently in the background and which processes broadcasts from the system or from other applications. For this reason, the BroadcastReceiver is particularly suitable for performing malicious actions. There are two techniques how BroadcastReceiver can be misused. The first approach is for Android 2.3.3 and lower because applications in these Android versions do not have to contain an Activity and therefore independent BroadcastReceiver can handle the events silently in the background. The second technique is for Android 3.1 and higher. In these versions of Android, each application with BroadcastReceiver must also have an Activity. (On condition that BroadcastReceiver

requires some permission.) The last thing for a good understanding of the Android malware issues is the mechanism of the ideal malware for mobile devices. Malware writers are trying to reach this mechanism.

The ideal malware starts automatically after the operating system startup and then runs concealed without any GUI in the background. For this reason, users are not able to notice that the malicious software is performing some action on their smartphones or tablets. Malware running in the background responds to various events such as:

- The operating system receives incoming SMS (virus can read this message from the SMS inbox).
- The mobile device was connected to a Wi-Fi network (virus can silently send some stolen data) etc…

If the malware detects the operating system shut-down request, it terminates itself without any error messages. The ideal malware lifecycle is shown in Fig. 5.

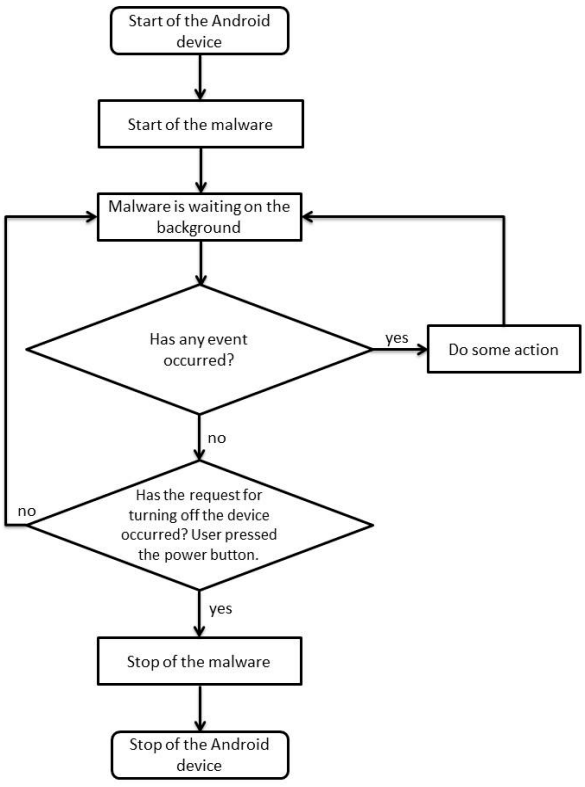

Fig. 5 The lifecycle of an ideal malware

3. Malware strategies using BroadcastReceiver

As mentioned above, malware using BroadcastReceiver is a quite popular solution for malware developers. The creation of BroadcastReceiver based malware is described in this section. According to the FBI report [9], there is still quite significant market share of devices running Android 2.3.3 and lower.

"Industry reporting indicates 44 per cent of Android users are still using versions 2.3.3 through 2.3.7-known as Gingerbread-which were released in 2011 and have a number of security vulnerabilities that were fixed in the later versions" [9]. In 2014, the Gingerbread version had a 13.6 per cent market share [8]. First, techniques for creating malware for Android 2.3 will be shown. Afterwards, this malware will be used for characterization of some significant risks of the operating system Android 2.3.

3.1 Malware for Android 2.3.3 and lower

The process of malware creation for Android 2.3.3 and lower is composed of a sequence of following steps: First, an Android project, which is not different from a standard application project, has been built up. Second, an instance of the BroadcastReceiver class has been created and then its method onReceive has been implemented.

3.1.1 – Malware code

For implementation of the onReceive method, for instance, this kind of Java code can be used:

```
@Override
public void onReceive(Context context, Intent intent)
{
  try
  {
    Bundle bundle = intent.getExtras();
    String sMSMessageString = "";

    Object[] pdusObj = (Object[]) bundle.get("pdus");
    SmsMessage smsMessage =
SmsMessage.createFromPdu((byte[]) pdusObj[0]);

    sMSMessageString += "From: " +
smsMessage.getOriginatingAddress();

    Calendar calendar = Calendar.getInstance();
```

```
    calendar.setTime(new
Date(smsMessage.getTimestampMillis()));

    SimpleDateFormat sdf = new SimpleDateFormat("'\nDate:
'MM-dd-yyyy'\nTime:
    'HH:mm");
    sMSMessageString += sdf.format(calendar.getTime());

    sMSMessageString += "\nText: " +
smsMessage.getMessageBody();

    Toast.makeText(context, "*** Captured SMS ***\n" +
sMSMessageString,
    Toast.LENGTH_SHORT).show();
  }
  catch (Exception e)
  {
    Log.d("MMM", e.toString());
  }
}
```

Code explanation: When an event of the incoming SMS occurs, OnReceive implementation reads the sender's number, sending date and time and text from this message. This is just an example how the information from the stolen SMS is harmlessly displayed on a mobile device. However, in a real situation, a more probable scenario would be used. Information from the stolen SMS will be stored in a file in persistent memory of the mobile device and it will be encrypted (after the encryption, users will not be able to read this file and they will not be able to find out that their messages are stolen). If malware detects available Wi-Fi connection, it sends the encrypted file with the messages to the attacker using the SCP protocol.

3.1.2 – Edit of AndroidManifest.xml code

The next step which is necessary to be done is adding a receiver tag to the AndroidManifest.xml file:

```
<receiver android:name=".MalwareReciever"
        android:exported="true"
        android:permission="android.permission.BROADCAST
_SMS">
        <intent-filter>
                <action
        android:name="android.provider.Telephony.SMS_REC
        EIVED"/>
        </intent-filter>
</receiver>
```

Code explanation: The system is instructed that the malware application wants to use BroadcastReceiver called MalwareReceiver. After that, the system is told that BroadcastReceiver needs BROADCAST_SMS permission. Finally, the system is asked to inform BroadcastReceiver every time SMS_RECEIVED event occurs (a new SMS was received).
Next, uses-permission tag to the AndroidManifest.xml file should be added:

```
<uses-permission
android:name="android.permission.RECEIVE_SMS" />
```

From the project directory the MainActivity.java file has been deleted. In the end, the activity tag from the AndroidManifest.xml file has been deleted as well.

3.1.3 – Adjustment of the application list

Fully operational malware can be created using the steps described above. Nevertheless, the user is still able to notice this malware on the list of installed applications (Settings -> Applications -> Manage applications -> List of installed applications). That is the reason why it is necessary to carry out its camouflage. It typically consists of two steps. Note: there are two lists of installed applications. The first one is available in the Android settings (Settings -> Applications -> Manage applications -> List of installed applications). The second one is an icon list available by clicking on "application button" (it is usually a software button). They will be called "a list of installed applications" and "an icon list of installed applications".

In the first step it is necessary to change the application name displayed on the list of installed applications. There is a tag application parameter: android: label="@string/app_name" in the AndroidManifest.xml file. In this case, the parameter takes value from the file strings.xml. Here, the value has been changed from <string name="app_name">Our Malware Name</string> to an unintelligible value for a common user, for example: <string name="app_name">Google Service Setup</string>. The Android operating system does not check the app_name value. Therefore, the malware can have a name similar to some operating system components. Such fake names are useful since most users usually try to find information only via an

internet search engine. If users receive some results, they stay calm because they think it is all right to have this program on their mobile devices. For a common user, the difference between real "Google Services Framework" and the invented "Google Services Setup" is marginal. If the user tries to search "Google Services Setup" term in the Google search engine, he receives approximately 16,200 results (this number dates back to the time of the experimental malware creation).

In the second step, the malware icon has been changed to a system icon. Moreover, the malware creator does not have to create his own imitation of the system icons because it is simply possible to copy them from the directory .../sdk/samples/... The result of camouflage can be seen in Fig. 6.

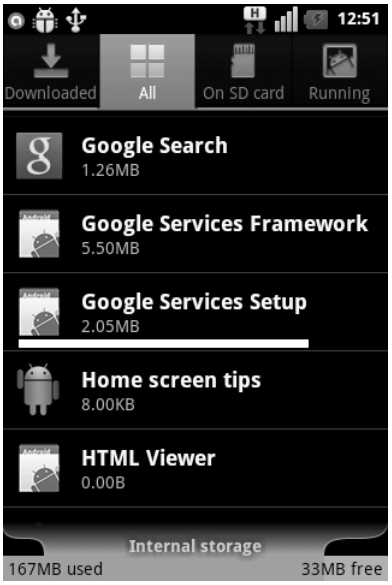

Fig. 6 The camouflage test result of malware on Android 2.3

3.1.4 – Experiment summary

The created testing malware revealed dangerous features of Android 2.3. Here are some of them:

- As it was written above, the operating system Android does not check the value of the app_name parameter. This fact allows giving the malware a name similar to a part of the operating system (this feature is also in the higher Android versions, including 4.4.2).
- Malware does not have to contain an Activity and it can handle events silently in the background.
- If the malware does not contain an Activity, the icon of the malware does not appear on the icon list of applications. But it appears on the list of installed applications. As it was mentioned above, that is the reason why the malware has to be concealed.
- If the malware does not contain an Activity but only BroadcastReceiver, it is not necessary to launch the malware automatically after the startup of the operating system. If a system event occurs (such as incoming SMS or connection to the Wi-Fi), the operating system automatically activates the BroadcastReceiver of the malware so it can respond to that event (e.g. stealing the SMS).
- Thus, it is not even necessary to ask for permission <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/> allowing to launch an application when booting of the operating system is completed. The situation can be seen in Fig. 7 and Fig. 8. After the Android OS startup, list of running applications is empty as can be seen in Fig. 7. In spite of that, the incoming message has been caught by the malware. See Fig. 8.

These tests were performed on LG P500 Optimus One with Android 2.3.3.

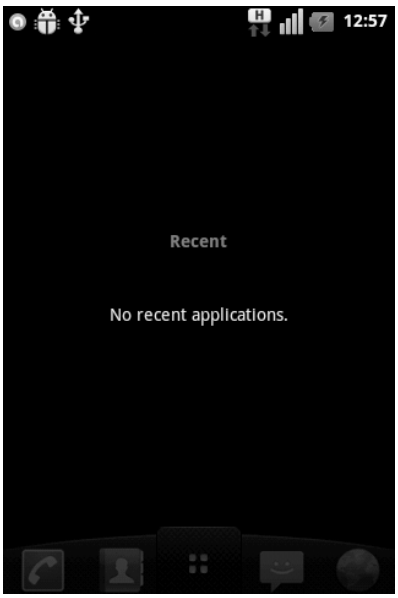

Fig. 7 After the startup of Android 2.3, the list of running applications is empty

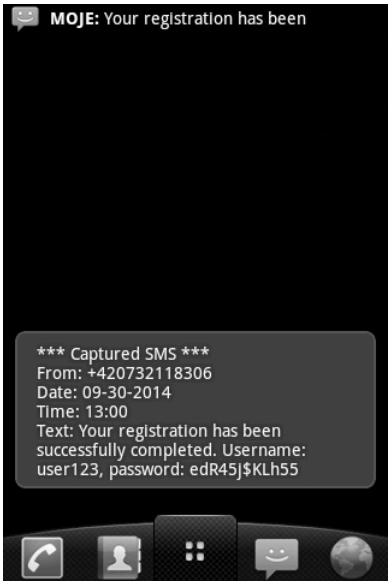

Fig. 8 The incoming message has been caught by the malware

Fig. 9 Added Activity on Android 3.1 and higher

3.2 Malware for Android 3.1 and higher

This is the main part of the research carried out in this section because it is concerned with modern smartphones and tablets with Android operating system. Google Inc. company [10] created a security improvement for these latest versions of Android that no longer allow applications or malware to silently execute their tasks only via BroadcastReceiver. Nowadays, each application with BroadcastReceiver must also have an Activity (if BroadcastReceiver requires some permission). If the malware, created as it was described above, is installed on devices running Android 3.1 or higher, the installation is successful but the application does not work. In order for the malware to work again, a number of modifications has to be made as shown below. Thus, the malware we created for Android 2.3.3 and lower has been edited further to be able to bypass the Google's security improvement. First, a class has been added to the malware that is a descendant of an Activity class and then the AndroidManifest.xml file has been edited by adding the activity tag. The result of this modification is shown in Fig. 9.

3.2.1 – Process of the Activity camouflage

As can be seen in Fig. 9, Activity is now visible and that is why the malware can be easily detected by users. If the malware is supposed to work the same way as on older Android versions, it is necessary to camouflage the Activity which is now mandatory. It is essential to make the Activity transparent. It can be done by editing the style in .../res/values/ styles.xml file where these item tags will be added:

```
<item
name="android:windowIsTranslucent">true</item>
<item
name="android:windowBackground">@android:color/
transparent</item>
<item
name="android:windowContentOverlay">@null</item
>
<item name="android:windowNoTitle">true</item>
<item
name="android:windowFullscreen">true</item>
<item name="android:windowIsFloating">true</item>
<item
name="android:backgroundDimEnabled">false</item
>
```

Then all UI elements, such as TextView from the XML layout file, have been removed. If we did not remove all the UI elements, Activity would not be

completely transparent, as shown by Fig. 10 (the highlighted part). In the XML layout file, it is important to keep only a valid tag describing the layout, for instance RelativeLayout.

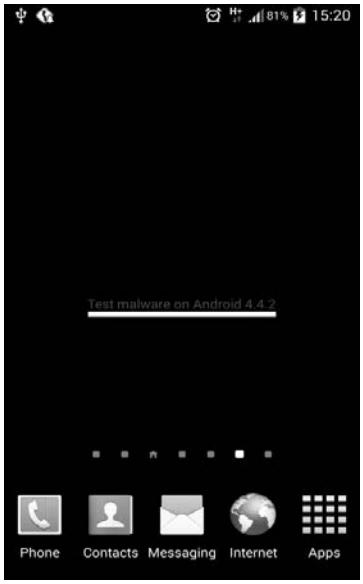

Fig. 10 UI elements cause that Activity is not completely transparent.

3.2.2 – The display freeze fix

Now the running Activity is already fully transparent. But another problem occurs if the Activity gets into the foreground. The mobile device looks "frozen" (it does not respond to any user touches on the screen). This happens because the fully transparent activity takes up the whole screen. For example, if the user pushes the software button on the screen, in fact he does not push the software button but he touches the transparent Activity located over this button. This behavior can be fixed by adding onResume method to the Activity. Then finish() is put [5] to onResume. It is not necessary to put finish() into other methods, as can be seen in Fig. 11. In all cases (onStart, onRestart, Paused, Stopped), onResume method is eventually called. The calling of finish() from onResume ensures that the Activity immediately moves into the background right after it gets into the foreground. Thus the Activity releases the screen of the Android device.

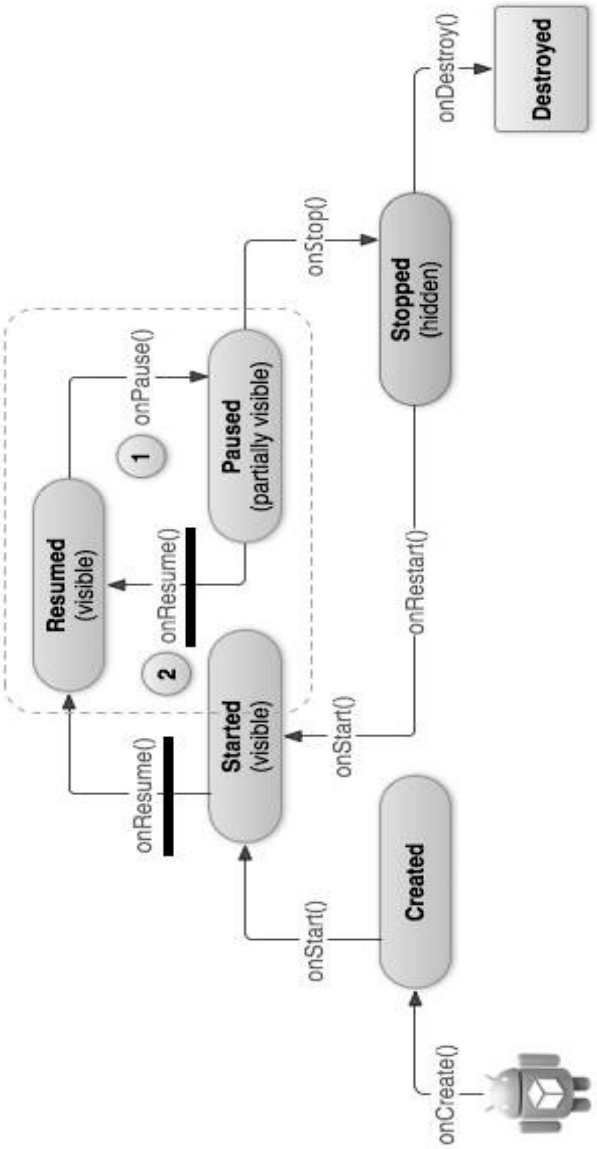

Fig. 11 Lifecycle of an Activity [6]

There is a flash during the Activity transition from the foreground to the background. However, this flash is invisible because the Activity is transparent.

3.2.3 – Camouflage of application icons/GUI

The malware icon is still visible on the list of running applications (Fig. 12) and on the icon list of installed applications (Fig. 13). This can lead to malware uncovering. Therefore, in the end, camouflage of the icon and caption is carried out.

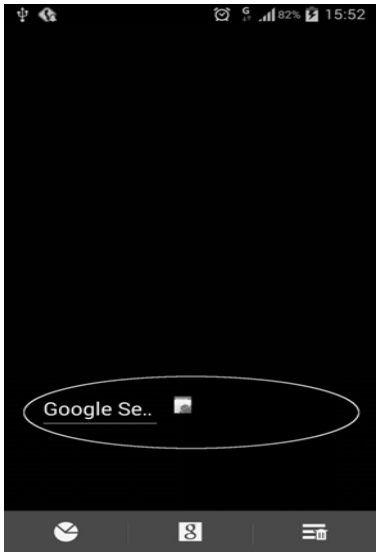

Fig. 12 The malware application is still visible on the list of running applications

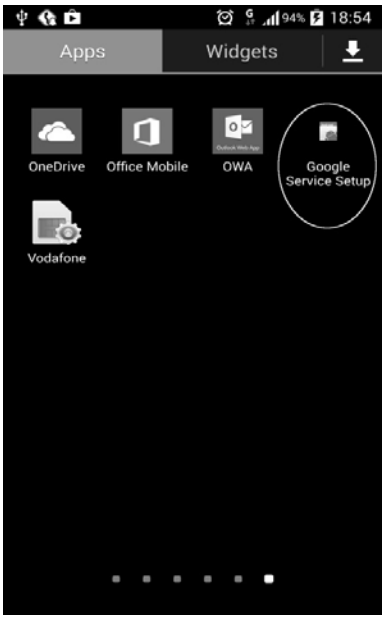

Fig. 13 The malware application is visible on the icon list of installed applications

After editing the file: .../res/values/strings.xml, the value has to be changed from <string name="app_name">Google Service Setup</string> to <string name="app_name"> </string>. Notice that the new value is not null, it is a space. Next, it is essential to ensure that the parameter android:label of the application tag, which is in the AndroidManifest.xml file, refers to this value: android:label="@string/app_name". The parameter android:label should not be edited directly in AndroidManifest.xml file. The next step is replacing standard icons with transparent PNG images. It is done in the directories: .../res/drawable-hdpi, .../res/drawable-mdpi, .../res/drawable-xhdpi and .../res/drawable-xxhdpi.

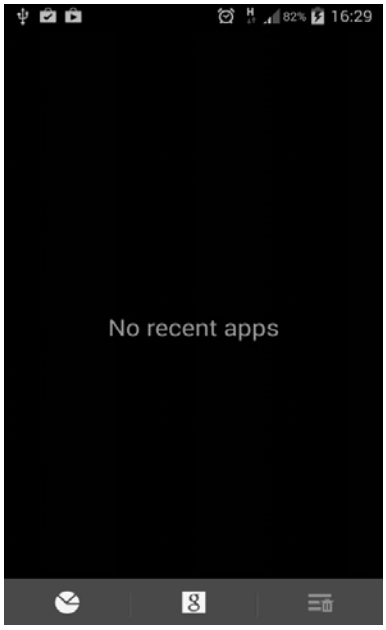

Fig. 14 The malware application is not on the list of running applications

As can be seen in Fig. 14, this malware application is not on the list of running applications. However, the malware application is running (as shown in Fig. 15) and it is no longer visible on the list of installed applications either (Fig. 16). If the user clicks into the place where the malware icon was, she or he does not see anything happen. The whole process of camouflaging is done. Now, the malware has the same behavior as on Android 2.3 which means that the Google's security improvement has been bypassed.

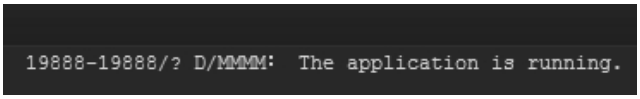

Fig. 15 The log of Logcat

Fig. 16 The malware application is no longer visible on the icon list of installed applications

These tests have been performed on Samsung i9505 Galaxy S4 with Android 4.4.2

4. Test results

Two malware samples have been created, the first is designed for Android 2.3.3 and the other for Android 4.4.2.

Our results flowing from the tests on Android 2.3.3 proved that it is quite easy to develop BroadcastReceiver based malware for this version of Android. As described in section 3.1.4 – Experiment summary, malware does not have to contain an Activity and it can handle events silently in the background. If the Android 2.3.3 malware does not contain an Activity, it results in the fact that the malware icon does not appear on the icon list of applications.

The tests on Android 4.4.2 have shown that each application with BroadcastReceiver must also have an Activity (if BroadcastReceiver requires some permission). If the malware created for Android 2.3.3 and lower, as it was described above, is installed on devices running Android 3.1 or higher, the installation is successful but the application does not work. For this reason, the Activity has to be added to the malware which means the hiding strategy must be different. This process is described in the chapter 3.2 Malware for Android 3.1 and higher. The test of the malware application was successful again. The written malware was hidden inside the testing smartphone. The activity was camouflaged and its GUI was hidden. The result shows that it is possible to write BroadcastReceiver malware for these newer versions of the Android operating system. Thus, the protection implemented in versions higher than 3.0 has been bypassed.

5. Recommendations

Based on our research, we recommend antivirus companies to include in their mobile antivirus programs a repeated check whether the option to install software from unknown sources is disabled. Due to the possible presence of Clickjacking malware on Android mobile devices, it is necessary to carry out this checking repeatedly. It is also necessary for an antivirus to notify the user every time the option to install software from unknown sources is enabled. We also recommend antivirus companies to inspect the .../res/values/strings.xml file whether the string tag with the name="app_name" parameter does not contain empty string or whitespace character(s). For example:

- <string name="app_name"></string> or
- <string name="app_name"> </string> or
- <string name="app_name"> </string>
- and etc.

Finally, we recommend creation of an antivirus test to check whether icons in directories: .../res/drawable-hdpi, .../res/drawable-mdpi .../res/drawable-xhdpi and .../res/drawable-xxhdpi do not contain transparent png images.

6. Conclusion

One of the most important problems which every successful malware writer has to resolve is hiding malicious software from users. If they do not know that malware is on their mobile devices, they are not able to take steps to remove it.

This research has shown that writing a piece of malware for Android 2.3.3 and lower is very easy and a common user has very little chance to detect such malware. This result fully confirms the conclusions of "Roll call release for police, fire, ems, and security personnel" by FBI. Namely, that "Android 2.3 has a number of security vulnerabilities that were fixed in later versions" [9]. For these reasons, we strongly recommend not to use smartphones and tablets running Android 2.3.3 or lower because these devices represent a considerable security risk. The different situation is with the new versions of the operating system where Google Inc. tried to fix this security vulnerability. Now, each application with BroadcastReceiver must also have an Activity, however, this research has shown that this security protection can be avoided. As described above, it has been managed by using camouflage of Activity. Also the icon and the label of the malicious application have been hidden. The results of our research are consistent with opinions of many IT security experts: "users should never allow the device to install apps from "unknown" sources" [13]. That is the only reasonable way how to prevent this type of malware to infiltrate Android mobile devices.

All our recommendations, described in the section 5, are relatively easy to implement. Running the tests created according to our recommendations is not time consuming and the tests can significantly increase the probability of finding malware with a hidden Activity. That is the reason why the plan is to carry out further research in a cooperation with selected anti-virus companies that focus on malware for mobile devices.

Presented research in the context of previously published scientific papers:
The segment of mobile devices is one of the fastest growing sectors of the IT industry. For this reason, involved companies dynamically supply the market with new mobile devices, new versions of the Android operating system and new user applications. The development cycle of these technologies is constantly getting shorter so it is difficult to monitor the latest trends and to respond to them in a research. Nevertheless, there is a range of excellent works, such as:

- Mobile malware detection based on energy fingerprints - A dead end? [11];
- Kernel-based behavior analysis for android malware detection [12];
- Android malware detection system classification [15].

These works mostly deal with malware detection based on some anomalies (deviations from the standard operation of mobile devices). A different approach to the issue, similar to the point of view of malware writers, has been used in this paper. Study of the Android documentation on Google's web [7] was essential to this work. Furthermore, posts published on developer forums by programmers have been analyzed and then special test applications for searching weaknesses of the Android operating system have been created (not all security vulnerabilities we have discovered are published in this article). It has been time-consuming and technically demanding research that has brought many interesting and still unpublished results.

Acknowledgments

This work was supported by the European Regional Development Fund under the project CEBIA-Tech Instrumentation No. CZ.1.05/2.1.00/19.0376 and also by Internal Grant Agency of Tomas Bata University under the project No. IGA/FAI/2016/016.

References

[1] Android, “Application Fundamentals”, [online], http://developer.android.com/guide/components/fundamentals.html, 2014.
[2] Android, “Activity onCreate” [online], http://developer.android.com/reference/android/app/Activity.html#onCreate(android.os.Bundle), 2014.
[3] Android, “Activity onStart”, [online], http://developer.android.com/reference/android/app/Activity.html#onStart(), 2014.
[4] Android, “Activity onResume“, [online], http://developer.android.com/reference/android/app/Activity.html#onResume(), 2014.
[5] Android, “Activity finish ()“, [online], http://developer.android.com/reference/android/app/Activity.html#finish(), 2014.

[6] Android, "Lifecycle of an Activity", [online], http://developer.android.com/images/training/basics/basic-lifecycle-paused.png, 2014.
[7] Android, "Android Developer", [online], http://developer.android.com/develop/index.html
[8] Android Community. "Android distribution: KitKat now over 20%", [online], http://androidcommunity.com/android-distribution-kitkat-now-over-20-20140813/, 2014.
[9] Federal Bureau of Investigation, *"(U//FOUO) DHS-FBI Bulletin: Threats to Mobile Devices Using the Android Operating System"*, [online], https://publicintelligence.net/dhs-fbi-android-threats/, 2013.
[10] Google, Company, [online], https://www.google.com/about/company/, 2014.
[11] Hoffmann, J., Neumann, S., Holz, T., Research in Attacks, Intrusions, and Defenses: Mobile malware detection based on energy fingerprints - A dead end?, *Lecture Notes in Computer Science*, Vol. 8145, pp 348-368, ISBN 978-3-642-41283-7, 2013.
[12] Isohara, T., Takemori, K., Kubota, A., Kernel-based Behavior Analysis for Android Malware Detection. In: *Seventh International Conference on Computational Intelligence and Security (CIS 2011).* pp. 1011 – 1015, ISBN 978-1-4577-2008-6, 2011.
[13] Nelson, P., "Device from Malware", [online]. http://www.linuxinsider.com/story/79469.html.
[14] Trout, Ch., "Android still the dominant mobile OS with 1 bilion active users", [online], http://www.engadget.com/2014/06/25/google-io-2014-by-the-numbers/, 2014.
[15] Zaki Mas`ud, M., Sahib, S., Faizal Abdollah, M., Rahayu Selamat, S., Yusof, T., Android Malware Detection System Classification. *Research Journal of Information Technology*, 6: 325-341, 2014.
[16] Number of smartphone users* worldwide from 2014 to 2019 (in millions). 2014. The Statistics Portal. Available from: http://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/
[17] Gartner Says 4.9 Billion Connected "Things" Will Be in Use in 2015. 2014. Gartner [online]. Barcelona. Available from: http://www.gartner.com/newsroom/id/2905717
[18] Mobile Malware Report: Threat Report: H2/2014. 2014. G DATA Software [online]. Trust in German Sicherheit. Available from: https://public.gdatasoftware.com/Presse/Publikationen/Malware_Reports/G_DATA_MobileMWR_H2_2014_EN.pdf
[19] Kaspersky Security Bulletin 2014: Overall statistics for 2014. 2014. Secure List [online]. Available from: https://securelist.com/analysis/kaspersky-security-bulletin/68010/kaspersky-security-bulletin-2014-overall-statistics-for-2014/

Milan Oulehla Born in Sternberk, Czech Republic, 15th November 1976.
Bachelor degree in Applied Computer Science 2011, Palacky University in Olomouc. Master degree in Information technology 2013, Tomas Bata University in Zlin. Ph.D. student at Tomas Bata University in Zlin, Faculty of Applied Informatics. Main specialization: mobile security, mobile viruses, wearable hardware, cryptography on mobile platform.

David Malanik
Born in Zlin, Czech Republic, 1. March 1984. Master degree in Information technology 2008, Tomas Bata University in Zlin. Ph.D. Thesis based on the user identification provided by the neural network, 2011, Tomas Bata University in Zlin.
SENIOR LECTURER at Tomas Bata University in Zlin, Department of Informatics and Artificial Intelligence.
Main specialization: computer security, computer viruses, penetration testing, artificial neural network, computer networks.