DE GRUYTER
OPEN

TATRA
MOUNTAINS
Mathematical Publications

DOI: 10.2478/tmmp-2014-0026
Tatra Mt. Math. Publ. **60** (2014), 85–100

PRESENTING RISKS INTRODUCED BY ANDROID APPLICATION PERMISSIONS IN A USER-FRIENDLY WAY

Juraj Varga — Peter Muska

ABSTRACT. The emergence of Android as a leading Operating System in terms of market coverage induced the rapid emergence of many mobile applications. A lot of these applications are prone to misuse because of their design. This paper deals with a new method of informing user whether applications installed on his device are potentially harmful or not. The introductory part provides some insight into security mechanisms used in Android. The main part deals with research we based our work on, proposal of our own methodology based on permission model and its implementation in application for real-time offline analysis of installed applications on device running Android. The last part of this paper deals with the evaluation of achievements reached by our methodology implemented in standalone application.

1. Introduction

Operating system Android is currently the most widespread operating system (OS) for mobile phones, tablets and other devices [1]. Its development began in 2003 in Android Inc. The first phone with Android was released by HTC in 2008 in USA and reached central Europe in summer 2009. Android is operating system built on Linux architecture. Due to the fact that mobile devices have limited hardware resources, this architecture had to be modified to fit this situation. Securing an open platform requires robust security architecture and precisely designed security system. Even though it is possible to provide security at the application level (by means of cryptography or steganography [2], [3]), this does not provide protection against the security leaks at the level of operating system modules. But multi-layer architecture of Android provides necessary flexibility in development and also certain level of protection. Security measures

2010 Mathematics Subject Classification: 94A60.
Keywords: Android, mobile devices, permissions, over-privilege.
This work was supported by grant VEGA 1/0173/13.

were designed with regard to developer community—experts can easily work with these mechanisms, beginners are protected by default settings. Despite these measures, there are ways to circumvent them. One of them is misusing over-privileged applications. In this paper we propose a new approach in detecting these applications and presenting the results to the user.

The following part of this paper deals with the overview of the permission model and its connection to Android security. The third part presents prior research in this field, which we used as a reference starting point in our work. Next two parts describe our method of detecting over-privileged applications and tests conducted with proof-of-concept application, respectively. This paper is concluded in the sixth part.

2. Permission model

From the beginning, OS Android is being developed as an open mobile platform. It enables applications to use built-in hardware and software along with both local and remote data to grant users desired comfort and functionality. Along with all this the OS must provide means to secure user data, applications and whole device.

On the OS level Android provides security of Linux kernel, along with secure inter-process communication between applications running in different processes. These security features on OS level make sure that even native applications are subjected to application sandboxing. This way system prevents harmful applications from damaging other applications, device or OS itself [4].

Applications run in application sandbox and have access to limited system resources. System manages application access to resources. These restrictions are implemented by various means. Some possibilities are restricted by lack of corresponding Application Programming Interfaces (APIs), others by, e.g., role separation. Sensitive APIs can only be used by trusted applications and are protected by permission system. Permissions are divided into four groups based on level of protection (for illustration see Figure 1):

- Normal: permissions on the application level, they do not pose a serious risk when they are used by the application.
- Dangerous: permissions which can cause leaks and manipulation with sensitive data or exploit potentially dangerous system resources. They need to be explicitly confirmed by user during installation of application. Here belong for example:
  - Location data from GPS (ACCESS_FINE_LOCATION).
  - Network/data connection (ACCESS_NETWORK_STATE).

  - Telephony (CALL_PHONE).
  - SMS/MMS functions (WRITE_SMS).
  - Access to system configuration (CHANGE_CONFIGURATION).
- Signature: permissions assignable only to applications signed with private key corresponding with certificate of application calling it. They are used by developers to share information among their applications.
- Signature-or-system: special type of permission assignable only to applications installed in system image which are signed with the same certificate as system image [5] [6].

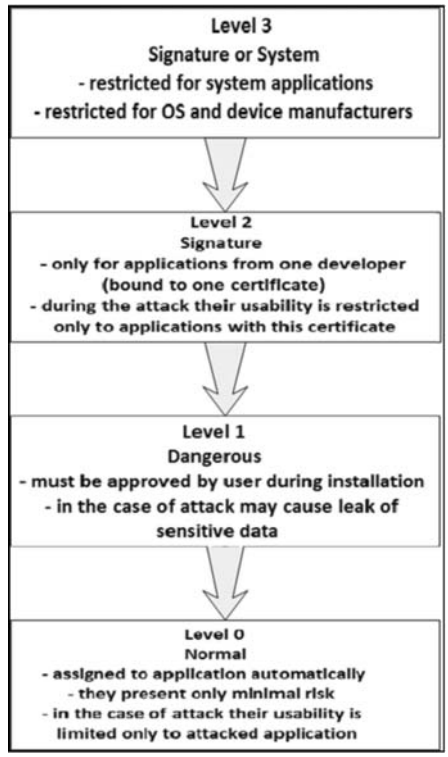

FIGURE 1. Android permission model.

Aside from the above mentioned division, permissions are also divided in a way of accepting them:

- Time-of-use: user must confirm this permission when executing sensitive operation (e.g., access to device location). It is the only way to prevent applications to access device resources.

- Install-time: accepted when installing application, user accepts them as one; he cannot choose which permissions to accept and which to deny, see Figure 2.

FIGURE 2. Example of application installation [7].

System resources marked as *dangerous* are accessible only from the OS. Applications must have these requirements specified by permissions in manifest. During installation these permissions are displayed to user and he can accept or reject them. After accepting these permissions the installation continues and these permissions are accepted by the system. It is not possible to choose which permissions to accept, they must be accepted as one and that can lead to security incidents. Permissions are assigned to an application for the whole time it is installed on the device and cannot be manipulated by anyone. They are removed in the moment the application is deleted from the device. They can be revised in application settings and can be restricted by shutting down global functionality, such as Wi-Fi or GPS. If the application is trying to access a feature that is not allowed to, it invokes a security exception and error message in the application. Security checks for protected API permissions are done on the lowest

possible level to prevent their circumventing. Some device capabilities are not accessible to third-party applications but can be used by pre-installed applications. The complete list of permissions is available on website dedicated to Android development [5] [6]. Other mobile platforms also use security mechanisms based on permissions, e.g., iOS by Apple [8], but their usage is different.

3. Prior research

Since the introduction of the permission model as a part of Android security, there has been an active research in various aspects of this model.

3.1. System of permissions

The basic research is focused on how this system works. This means dealing with detection of over-privileged applications and applying restrictions on installed applications or during their installation.

3.1.1. Detecting over-privileged applications

This topic is closely tied with privilege escalation attacks and is considered to be the first step in enhancing security in Android OS. There are systems that can detect over-privilege in applications, e.g., Stowaway [13]. This type of applications provides information whether the assigned permissions are used to access system resources or not, and if this access is relevant to functions required of the applications.

3.1.2. Applying restrictions

Important part of research on Android platform deals with restricting access to system resources. As it was mentioned in Part 2, the user either allows application access to all required resources and installs it, or does not install it at all. Current research deals with two sub problems here: applying restrictions and tracking the data flow and access to resources when they are allowed. The first framework to implement restrictions in the permission model was APEX [15], which allowed to revoke permissions and allow them when necessary. Unfortunately, this project is no longer active, but its idea continued in various custom ROMs like CyanogenMod or in frameworks like AppOps. Project Pyandrazzi [21] also continues in this idea, but focuses more on the impact of these restrictions on application functionality. Then there is Blue Seal [22], which tracks the data flow and access to resources in monitored applications. Moreover, it provides the details to the user and can rather successfully identify samples of mobile malware.

3.2. Privilege escalations

The most severe problem of the permission model is a fact that developers assign more permissions to their applications than they need to work properly. For example calculator application does not need access to the Internet to work, but uses it to download commercial banners which are main sources of income for developers. However, they can pose a security risk [12]. If the application has been assigned more permissions than it actually uses, it can be misused for privilege escalation attack [18]. Essentially, it means that an application with more permissions than it needs can be used by some malware application to do operations which this malware cannot do alone. These are the most common types of attacks on Android platform, and they are often caused by the lack of best practices for secure development among developers. B u g i e l et al. in [19] presented a framework XManDroid capable of detecting and preventing these attacks. This framework was later improved to cover more possible sub types of this attack [20].

3.3. Methods of malware detection using permissions

Permission model can be also used as a mean of mobile malware detection. However, in most cases they are only used as an additional factor in determination of application harmfulness.

3.3.1. ScanDroid

One of the first usable applications is ScanDroid by F u c h s et al. [11]. The authors focus on statistical analysis of data stream going through various applications. In addition Scandroid analyses permissions for each application stored in the manifest file. Based on these results and knowledge of permissions it can determine if it handles the data correctly or not, and therefore determine whether it is harmful or not. The application is then marked and presented to the user as potential risk.

3.3.2. Kirin

Kirin was also one of the first detection methods [16]. This system uses certification of applications during installation. It reads a list of required permissions from the manifest file and evaluates this configuration according to the set of security rules. Currently, this system is ineffective against new threats, but there are several new detection methods inspired by Kirin [20].

3.3.3. K-mean clustering

The latest method of malware detection based purely on permission model was presented in [17]. The authors extract features from *.apk* files and containing

permission requests. On these features they apply Information instal feature selection method and K-mean clustering algorithm supported by machine learning and decision trees. This work reached promising results, but further improvements are needed to determine whether this approach is successful or not.

3.4. User recognition and comprehension of permission model

The only research in this area so far was conducted by F e l t et al. in [7]. Results show, that complexity of permission model in regard to common user is a significant problem. Survey conducted by the authors shows that:

- Only 42 % of users is aware of which permissions application requires and know what are they for.
- Almost 70 % of users is influenced by reviews into installing some application.
- On the average only 30 % of users can correctly tell what operations can application do with specific permissions.

Based on these results we can say, that society is still not educated enough in this matter and therefore our research can provide some improvement in this regard.

The main issue with the majority of above mentioned solution is, that they are not publicly available. It means that they are either in development process or are used only for the academic purposes. Those publicly available—Kirin and its modification called TaintDroid [27]—have too complicated installation process, that even we had some problems with their installation. We believe that standard users will not want to spend a lot of time installing applications, they want this process to be quick. Moreover, they are somehow obsolete and do not offer sufficient level of protection any more. Therefore, in the next chapters we present our own solution, which is user friendly, quick to install and easy to comprehend.

4. Design

4.1. State-of-the-art

On *Google Play* there are currently many applications dealing with permission management. However, the majority of these applications is of informative character. For example applications like [23], [24] or [25] show the user which permissions chosen application uses and provide general information about these permissions. The more advanced ones like [23] and [25] can divide installed applications to lists depending on potential harm they can do, but these divisions are based only on general recommendations of Android API documentation found in [4]. Moreover, these applications do not quantify the risk presented by installed applications. Then there are applications capable of changing permissions, like [26]. This application can remove permissions from application manifest file

and thus removing the need for this permission. However, this approach does not change the API calls in the application install file and when such method is used, the application crashes. Also, many other studies linking permissions and malware were conducted over past years. The most significant ones were [9], [10], [14]. Apart from other research authors tried to find some connection between malware and permissions it requires. Authors in these papers conclude that mobile malware is nowadays almost identical to benign applications regarding required permissions. This is a direct result of overprivileging during the development process. Current permission model does not offer sufficient granularity in dividing permissions into functional groups—Signature and Signature-or-system cannot be accessed by potentially malicious application, Normal group is not interesting, so we are left with only one group—Dangerous. Above mentioned studies show, that this group of permissions is both used in benign and malicious applications. Therefore it is important that this category is further divided into smaller subgroups. In this work we propose division into four groups based on severity of possible misuse of given permission in over-privileged application, based on above mentioned research in this area:

- RED—high to critical risk—these permissions can only be used by a selected group of applications with legitimate claim to use them.
- ORANGE—moderate risk—these permissions should be used only by specific application, which requires them to function correctly.
- YELLOW—slight risk—these could be assigned to the GREEN group, but in some cases of application types could be misused.
- GREEN—little to no risk—typical, commonly required permissions which are safe, if they are not in combination with some other permissions.

Since this research intends to raise awareness among common users, it is vital that these groups immediately reflect potential threat to user. That is why we decided to use color distinction—the final score of each application is shown as a progress bar of some color depending on the potential threat level.

It may not be suitable to evaluate permissions based on some global view, because there are permissions which are necessary for one type of application and prone to misuse for other type. Fine example is SEND_SMS, allowing sending SMS messages—it is typical for messengers but very suspicious for games. That is why we have decided to also divide applications into logical groups, e.g.:

- GAMES,
- SOCIAL MEDIA,
- MESSENGERS,
- FINANCIAL.

Each of these categories has a set of permissions which are (according to our observations) necessary for them to be fully functional. On the other hand, there is a set of permissions the application should not use at all. These lists were created based on general expectations of what permissions the applications from specific group should require. Furthermore, we included in these lists permissions which are often misused my malicious applications as described in [9], [10], [14] or we deem them suspicious. These permissions would be treated differently in calculation of the final score - either their score will be lowered (if they are required) or raised (if they are suspicious). We believe this addition would make greater impact on the final score than the permissions alone and provide greater precision in final results.

4.2. Analysis and evaluation

Proposed method consists of two parts. The first part calculates the risk based on permissions the application requires. These permissions are distributed to the coloured groups based on the overall risk they pose (Part 4, 4.1). The second part consists of risk calculation based on the same permission list, only evaluated according to the category which the tested application belongs to. For example, INTERNET permission is required for some messenger or communication application, but suspicious for some utility application (calculator).

The evaluation of application is based on calculation of negative score, the higher the score, the higher the overall threat posed by the application. The evaluation model consists of two components. Each of these components uses evaluation function: general permission risk and category based permission risk.

The first component of evaluation R_1 is based on incrementing risk score according to the number of required permissions belonging to one of the four coloured groups (1). In our simplified model used to demonstrate the overall performance we use experimental values: green group increments score by 0.5, yellow by 1, orange by 3 and red by 5 points. When this is done, we get a sum of these values. Then we compute a ratio of achieved score with maximal possible score for this application (as if all permissions were from the RED group). This result is then multiplied by 100 to get a value in percentage. This calculation can be represented as:

$$R_1 = \frac{g \cdot k_g + y \cdot k_y + o \cdot k_o + r \cdot k_r}{(g + y + o + r) \cdot k_r} \cdot 100, \tag{1}$$

where g, y, o, r – total number of permissions from GREEN, YELLOW, ORANGE and RED groups,
k_g, k_y, k_o, k_r – coefficients of risk for permissions from each group.

The second component begins its evaluation after the first phase of analysis is done. Score is reset to zero. Required permissions are checked again, this time using the list of permissions for chosen application type (2). For any suspicious

permission found, the score is incremented by a predefined value set experimentally according to the category. Then we calculate the ratio of the reached score and the maximal possible score obtained for the given category (if all permissions found would be in the list of suspicious permissions for given category). This ratio is again multiplied by 100 to get a value in percentage. This calculation of risk R_2 can be represented by equation:

$$R_2 = \frac{p_s \cdot k_s + p_n \cdot k_n}{p \cdot k_s} \cdot 100, \tag{2}$$

where p – total number of permissions,
p_s – number of suspicious permissions for current category,
k_s – coefficient of risk for suspicious permissions from this category,
p_n – number of non-suspicious permissions for current category,
k_n – coefficient of risk for non-suspicious permissions from this category.

After both parts of our evaluation method are complete, the final score of analysed application based on the required permissions is calculated. The final score is calculated as a sum of partial scores (3) divided by 2 to achieve a value from 0 to 100. This result represents potential risk R of misuse of selected application:

$$R = \frac{R_1 + R_2}{2}, \tag{3}$$

where R – final score,
R_1 – score from the first part,
R_2 – score from the second part.

We remark that high score does not immediately mean a malicious application. The score is just a security recommendation and a mean of prevention against an over-privileged application (and only consequently a possible malware threat).

4.3. Proof-of-concept application

We implemented proposed scoring method as a standard Android application written in Java language. Application *Suspicious Apps Checker* is very simple and consists of several connected screens—activities.

After the application starts, it shows the user a list of all user-installed applications on the current device. We filter system applications and those from the device manufacturer, because they should not pose any threat. Each item from this list contains application title and an icon for better orientation and user experience. Each item is also linked with two buttons: one for launching and one for further information. Clicking the second button launches the *App Info* activity. Here the user can see all information regarding this application: current version, package info, date of the last update and all required permissions. These permissions can be clicked on and the user is shown the information about the desired permission—Figure 3.

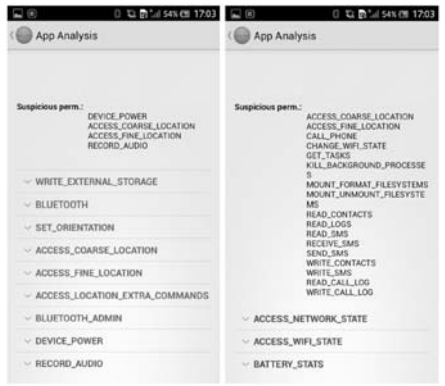

FIGURE 3. Clickable list showing information about required permissions.

Besides these information there is an *Analysis* button which calls the above mentioned analysis and evaluation. It is necessary to choose the type of application (examples mentioned in 4., 4.1), as it can significantly alter the results if chosen inaccurately. If the user does not choose the category, the analysis will be performed on default *Games* choice (as the least privileged category).

The result is shown on the next screen with the coloured progress bar and the suspicious permissions are listed as click-able list (as in Figure 4), where the user can see what possible damage the application with the given permissions can do (see Figure 3).

5. Tests and results

Implemented application *Suspicious Apps Checker* is fully functional according to the specifics in design. It does not require any permissions, therefore we can consider it secure. Tests were conducted in two steps:

- Testing of applications based on categories.
- Testing of individual applications.

5.1. Testing of applications based on categories

For each category we tested 20 most downloaded free applications in *Google Play* store (280 applications in total). In the Table 1 we can see the average result in each category. It also contains average values in these aspects:

- Overall ratio of how much is a specific category over-privileged.
- Number of required permissions for each category.
- Number of permissions that should not be assigned for each category.

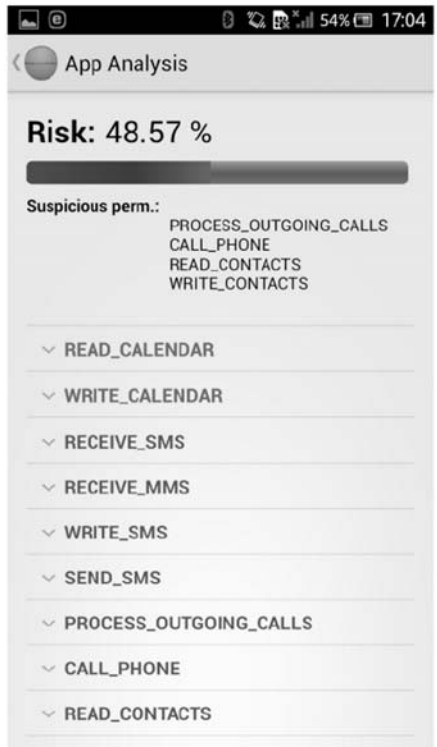

FIGURE 4. Evaluation result.

Based on these results we can say that the applications from the *Sports* category present the lowest potential risk with the score of 15,27 %. On the other hand, the highest risk of over-privilege is presented by applications from the *Finances* category with the score of 37,21 %. The *Communications* category ranks first as a category with the most required permissions with the score 23,43, which is not very surprising due to their purpose. The last category in this view is *Themes* with the average score of required permissions 4. The highest number of suspicious permissions require applications from *Shopping* with the score 1,67, which indicates possible case of over-privileged application. The lowest number of suspicious permissions required applications from *Travel* group, only 0,14.

5.2. Testing of individual applications

In case of individual applications the best results obviously achieved applications that does not require any permissions, thus our application labelled them secure with zero risk of misuse. However, the highest positions of potential risk were achieved by anti virus applications. This happened due to the fact that

TABLE 1. Results from the first round of testing.

Category	Avg. risk (%)	Avg. # of perms	Avg. # of susp. perms
Sports	15.27	6.86	0.29
Health	21.18	7.29	0.57
Games	22.14	6.72	0.72
Travel	23.05	6.00	0.14
Lifestyle	23.11	6.00	1.00
Multimedia	23.19	6.20	0.80
Entertainment	23.43	8.00	0.50
Social	25.96	16.25	0.50
Tools	26.47	16.00	1.40
Communication	26.85	23.43	0.86
Shopping	27.77	8.00	1.67
Productivity	30.01	8.40	1.40
Themes	32.33	4.00	1.00
Finance	37.21	8.17	0.83

these applications require significant amount of permissions to access various system resources needed to perform security scans on the device. The second highly risky group are financial applications. Since they need to work with account numbers, passwords and other sensitive data, they also require permissions belonging to the most dangerous group—this is why our application evaluated them as suspicious. There is a list of applications that reached the highest values of potential misuse as over-privileged applications in Table 2. As we mentioned in the previous section, the values used in evaluation equations were experimentally set for demonstration purpose. In our proof-of-concept testing we did not test our proposed method on actual malicious applications. According to previously conducted research [9], [10], [14], we believe, that malicious applications hiding behind legitimate functionality would obtain very similar results as benign applications. We plan to conduct these tests as a part of our future research.

6. Conclusion

In comparison with existing solutions for detection of over-privileged applications based on permission model, our new method appears to be a better solution. It combines suitable parts from the existing work and tries to expand them and make them more precise. A standard approach only informs the user about required permissions. Our approach adds a quantitative expression of the potential risk. Moreover, the risk is based on intuitive division of permissions

TABLE 2. Results from the second round of testing.

App	Category	Risk	Perms	Susp. perms
mBankSk	Finance	59.75	8	1
Volley Hangout	Games	51.15	18	4
Steves World	Games	48.82	16	4
Swap The Box	Games	48.82	16	0
Mobile Security Antivirus	Tools	48.57	32	4
Heureka	Shopping	46.67	6	2
Sleep as Android	Lifestyle	46.52	22	6
Orange Go	Productivity	45.56	9	2
Smart Banking	Finance	45.29	12	2
TB VIAMO	Finance	40.91	9	2
Telekom	Productivity	38.75	6	2
Lokator	Finance	35.71	7	0
Skype	Communication	35.09	30	1
iTransit	Travel	33.33	3	0
WhatsApp	Communication	33.29	31	1
Snapchat	Social	33.13	14	1
Sports Tracker	Health	32.92	11	1
AVG Antivirus	Tools	32.88	44	4
Go Weather Forecast	Themes	32	4	1
CM Security	Tools	31.54	19	3

according to the risk category (based on existing research) and application category as well (proposed method). Unlike other solutions, our approach does not need any permissions or root access to work properly.

The results we achieved in this work significantly depend on the created rules in the proposed analysis methodology. This process was based on subjective view, which is shown in the results. The achieved results can be different, if other parameters are used in the scoring methodology—including acceptance thresholds, different compositions of the lists of permissions, different evaluation functions etc. The existing parameters inform the user of potential over-privilege, and require his active participation in detecting potential malware. Further research in this area is required to optimize the various parametric settings, before the method is suitable for a more automated over-privilege, and potential malware, detection.

The proof-of-concept application *Suspicious Apps Checker* is being further developed and tested, therefore is not published for public use. However, researchers keen on contributing to this project could receive installation file upon request by mail.

Acknowledgement. The authors are grateful to anonymous reviewers for their helpful comments and remarks that helped to improve the quality of this paper.

REFERENCES

[1] *Android and iOS continue to dominate the worldwide smartphone market with Android shipments just shy of 800 million in 2013,* According to IDC, http://www.idc.com/getdoc.jsp?containerId=prUS24676414

[2] JOKAY, M.: *The design of a steganographic system based on the internal MP4 file structures,* Internat. J. Comput. Commun. **5** (2012), 207–214.

[3] JÓKAY, M.—KOŠDY, M.: *Steganographic file system based on JPEG files,* Tatra Mt. Math. Publ. **57** (2013), 65–83.

[4] *Android security overview,* http://source.android.com/tech/security/index.html

[5] SHABTAI, A.—FLEDEL, Y.—KANONOV, U.—ELOVICI, Y.—DOLEV, S.: *Google Android: A comprehensive security assessment,* Security & Privacy, IEEE **8** (2010), 35–44.

[6] Android permissions overview, http://developer.android.com/reference/android/Manifest.permission.html

[7] FELT, A. P.—HA, E.—EGELMAN, S.—HANEY, A.—CHIN, E.—WAGNER, D.: *Android permissions: user attention, comprehension, and behavior,* in: Symposium on Usable Privacy and Security—SOUPS '12, ACM, New York, NY, USA, pp. 1–14.

[8] ANTAL, E.—BARANEC, F.: *Techniques of obtaining sensitive data from Apple iOS devices,* in: 43. Konference EurOpen.CZ, Vranov, Czech Republik, 2013, Plzeň, EurOpen.CZ, 2013, pp. 21–32. (In Slovak)

[9] ZHOU, Y.—JIANG, X.: *Dissecting Android malware: characterization and evolution,* in: Proc. of the 33rd IEEE Symp. on Security and Privacy, San Francisco, CA, 2012, IEEE Computer Society, Washington, DC, USA, 2012, pp. 95–109.

[10] ENCK, W.: *Defending users against smartphone apps: techniques and future directions,* in: Proc. of the 7th Internat. Conf. on Information Systems Security—ICISS '11 (S. Jajodia, C. Mazumdar, eds.), Kolkata, India, 2011, Lecture Notes in Comput. Sci., Vol. 7093, Springer-Verlag, Berlin, pp. 49–70.

[11] FUCHS, A. P.—CHAUDHURI, A.—FOSTER, J. S.: *SCanDroid: automated security certification of Android applications,* Technical Reports of the Computer Science Department, 2009, 15 pp.

[12] GRACE, M. C.—ZHOU, W.—JIANG, X.—SADEGHI, A.-R.: *Unsafe exposure analysis of mobile in-app advertisements,* in: Proc. of the 5th ACM Conf. on Security and Privacy in Wireless and Mobile Networks—WISEC '12, Tucson, AZ, USA, ACM, New York, NY, USA, 2012, pp. 101–112.

[13] FELT, A. P.—SONG, D.—WAGNER, D.—HANNA, S.: *Android permissions demystified,* in: Proc. of the 18th ACM Conf. on Comput. and Commun. Security—CCS '11, Chicago, IL, USA, 2011, ACM New York, NY, USA, pp. 627–638.

[14] FELT, A. P.—FINIFTER, M.—CHIN, E.—WAGNER, D.: *A survey of mobile malware in the wild,* in: Proc. of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices—SPSM '11, Chicago, IL, USA, ACM, New York, NY, USA, 2011, pp. 3–14.

[15] NAUMAN, M.—KHAN, S.—ZHANG, X.: *Apex: Extending Android permission model and enforcement with user-defined runtime constraints,* in: 5th ACM Symposium on Information, Comput. and Commun. Security—ASIACCS '10, Beijing, China, 2010, ACM, New York, NY, USA, 2010, pp. 328–332.

[16] ENCK, W.—ONGTANG, M.—MCDANIEL. P.: *On lightweight mobile phone application certification,* in: Proc. of the 16th ACM Conf. on Comput. and Commun. Security––CCS '09, Chicago, IL, USA, 2009, ACM, New York, NY, USA, 2009, pp. 235–245.

[17] ZARNI, A.—WIN, Z.: *Permission-based Android malware detection,* Internat. J. of Sci. and Technology Research (IJSTR) **2** (2013), 228–234.

[18] DAVI, L.—DIMITRENKO, A.—SADEGHI, A.-R.—WINANDY, M.: *Privilege escalation attacks on Android,* in: Proc. of the 13th Internat. Conf. on Inform. Security—ISC '10 (M. Burmester et al., eds.), Boca Raton, FL, USA, 2010 Lecture Notes in Comput. Sci., Vol. 6531, Springer-Verlag, Berlin, 2011, pp. 346–360.

[19] BUGIEL, S.—DAVI, L.—DMITRIENKO, A.—FISCHER, T.—SADEGHI, A.-R.: *XManDroid: A New Android Evolution to Mitigate Privilege Escalation Attacks,* Technical Report TR-2011-04, 2011, 18 pp.

[20] BUGIEL, S.—DAVI, L.—DMITRIENKO, A.—FISCHER, T.—SADEGHI, A.-R.–SHASTRY, B.: *Towards taming privilege-escalation attacks on Android,* in: Proc. of the 19th Annual Network & Distributed System Security Symp.—NDSS '12, San Diego, California, 2012, pp. 1–18.

[21] KENNEDY, K.—GUSTAFSON, E.—CHEN, H.: *Quantifying the effects of removing permissions from Android applications,* in: IEEE Mobile Security Technologies—MoST '13, San Francisco, CA, 2013, pp. 11.

[22] HOLAVANALLI, S.—MANUEL, D.—NANJUNDASWAMY, V.—ROSENBERG, B.–SHEN, F.—KO, S.Y.—ZIAREK, L.: *Flow Permissions for Android,* in: IEEE/ACM 28th Internat. Conf. on Automated Software Engineering—ASE '13 (E. Denney et al., eds.), Palo Alto, USA, 2013, IEEE, Piscataway, NJ, 2013, pp. 652–658.

[23] *F-Secure App Permissions,* `https://play.google.com/store/apps/details?id=com.fsecure.app.permissions.privacy`

[24] *S2 Permission Checker,* `https://play.google.com/store/apps/details?id=com.byte256.permissionchecker`

[25] *Permission Friendly Apps,* `https://play.google.com/store/apps/details?id=org.androidsoft.app.permission&hl=sk`

[26] *Adv Permission Manager,* `https://play.google.com/store/apps/details?id=com.gmail.heagoo.pmaster.pro`

[27] ENCK, W.—GILBERT, P.—CHUN, B.-G.—COX, L. P.—JUNG, J.—MC-DANIEL, P.–SHETH, A. N.: *TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones,* in: 9th USENIX Symposium on Operating Systems Design and Implementation—OSDI '10, Vancouver, BC, Canada, 2010, USENIX Association Berkeley, CA, USA, pp. 393–409.

Received November 11, 2014

Institute of Computer Science and Mathematics
Slovak University of Technology in Bratislava
Ilkovičova 3
SK–812-19 Bratislava
SLOVAKIA

E-mail: juraj.varga@stuba.sk
peter.muska1@gmail.com