Empirical Analysis of Android Logs Using Self-Organizing Maps

Eric Finickel and Abdelkader Lahmadi
Université de Lorraine, Loria
Vandoeuvre-lès-Nancy, F-54506, France
Email: {abdelkader.lahmadi,eric.finickel}@loria.fr

Frederic Beck and Olivier Festor
Inria
Villers-lès-Nancy, F-54600, France
Email: {frederic.beck, olivier.festor}@inria.fr

***Abstract*—In this paper, we present an empirical analysis of the logs generated by the logging system available in Android environments. The logs are mainly related to the execution of the different components of applications and services running on an Android device. We have analysed the logs using self organizing maps where our goal is to establish behavioural fingerprints of Android applications. Each fingerprint is build using information available in logs and related to the structure of an application and its interaction with the system. The developed methodology allows us the better understand Android Apps regarding their granted permissions and performed actions and it proves to be promising for the analysis of malware applications with a minimal overhead and cost.**

I. Introduction

The Android environment is established as the defacto-standard operating system for several devices including smart phones, tablets, ebooks, watches, etc. The environment offer a wide set of applications to be downloaded and installed by the users through several available markets. The functioning of these applications often requires access to potentially sensitive user data (e.g., contact lists, passwords, photos), sensor inputs (e.g., camera, microphone, GPS), and/or information about user behaviour.

The behaviour of these applications is usually not verified by the providers of the markets. Only the user has to decide to install or not an application according to the requested set of permissions, associated to the access of the device sensitive functions. The user can then only rely on its own knowledge to make the decision. However, recently the number of malware targeting the Android environments is increasing with important consequence on the violation of users privacy. Recently, two major projects have disclosed lists of malwares targeting the Android environment: the Genome Project has a list of 1260 applications and the VirusTotal project has a list of 20 000 applications. The increasing number of malware is due to the lack of of checking and verifications when applications are published on the official market provided by Google or alternative markets.

Several methods and tools have been developed to analyse and detect malwares targeting mobile devices. We mainly find traditional tools such as Anti-viruses including *Antivirus Free*, *Lookout Security & Antivirus* et *Norton Mobile Security Lite* which rely on signatures based methods to detect malware as it has been used in desktops and PCs. Other works [1], [2], [3], [4] have proposed several approaches for detecting malicious applications through the analysis of the permissions associated to them, the instrumentation of the system to analyse calls to sensitive functions or the reverse-engineering of the applications bytecode. However, these approaches despite their detection efficiency are using heavy methods with an important overhead to collect information and properties characterizing Android applications.

In this paper, we present a methodology and an analysis of Android logs to identify behavioural trends of running Android applications. The patterns are then classified using Self Organizing Maps (SOM) with input vector containing permissions and actions performed by the applications. Logs have been collected using the logging system available in Android platform where we have developed an exporting probe running on an Android device and sending collected logs to a server. Each log entry is then analysed and all its respective fields are extracted and stored in a HBase. The empirical analysis of android applications using their logs allows us to identify several behavioural patterns of a set of top ranked free applications available in Google play store. We have also replayed the analysis made by Barrera et al [13] where they have only studied permissions of android applications using SOMs. We have extended their methodology by including more dimensions in the input vector related to the set of actions performed by an android application.

The rest of this paper is organized as follows. In section II we present an overview of the Android environment, its main components, its threat model and available existing approaches regarding the analysis of Android applications to detect malicious behaviours. In Section III, we present our methodology to analyse available Android logs to extract patterns of behaviour of running applications. In section IV, we present our results where we have analysed the top 100 Android application available in the official market. In Section V, we provide concluding remarks and future work.

II. Android Environment

A. Overview

The Android environment, as depicted in Figure 1, relies mainly on an optimized Linux kernel to manage system resources of the mobile device. System services, native and Java applications are executed as Linux processes. Each installed

978-1-4799-2003-7/14/$31.00 ©2014 IEEE

application has its own user identifier and a set of group identifiers associated to its requested permissions. The two types of identifiers are used to control the access of the application to the system resources. Each Android application is executed within its own Dalvik virtual machine for better isolation of running applications. An android application relies

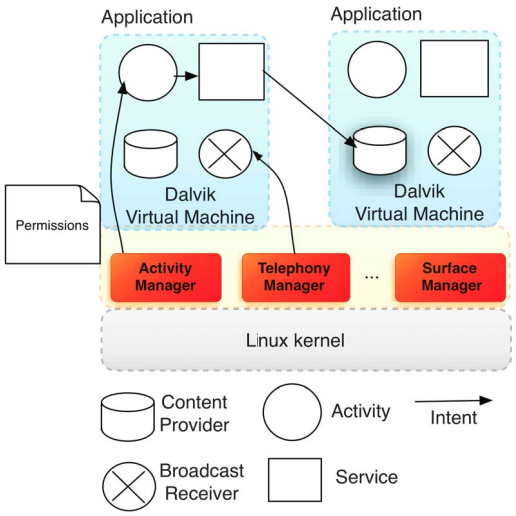


Fig. 1: A simplified overview of the Android environment.

on several component types. The main component types are activities, services, broadcast receivers and content providers. The instantiated components are communicating between them using intents relying on an inter-process messaging mechanism. The activities components are mainly defined to build user interfaces. The services components are used as background tasks without any user interaction. They are instantiated through activities or broadcast receivers when receiving one or several intents from the system or other components. The broadcast receivers are used as listeners of system or application events. Each Android application has also a set of permissions specified by the developer that should be accepted by the user at installation to guarantee access to a specific set of sensitive system resources such as location service, reading contacts, Internet access, writing and reading storage, making calls, sending SMS, etc [5].

B. Threats and attacks

Android devices are subject to several threats and attacks [6] due to an increasing number of malicious applications with an important consequence on their user's privacy and financial information.

Several threats are related to financial charging where an application may increase user's billing when sending on behalf of him premium number calls and SMS. The action is hardly detected by the user since an application may send them silently without providing a feedback to him [6]. Several attacks are also targeting the content of the SMS, MMS, call logs and different accounts information of a user [7]. For example, the malware FakeNetflix was able to steal users Netflix account information. Other malwares are less privacy threatening, and their goal is to install a botnet on the user's device and generate an important number of requests to increase the rank of a web site [8]. Traditional Denial of Service attacks are also targeting Android devices. Attackers are able to exhaust the battery of the device, generate an important network traffic or make the device unusable. For example Bickford et al [9] have developed a rootkit able to exhaust the battery on smart phone by only activating several power-intensive services such as GPS, WIFI and bluetooth. Attackers are using different techniques to infect an android device [6]. The repackaging technique allows an attacker to download an existing application and piggyback a malicious payload, then resubmit the application to an official/or alternative market. In addition attackers usually use legitimate classes naming for the added malicious code into the modified application. For example, the malware AnserverBot uses *com.sec.android.provider.drm* as a package name in their malicious code. Other attackers are using fake applications that are similar to existing applications but they are performing malicious actions. Other techniques for installing and building malwares are detailed in [6].

C. Android malware detection

Android malware detection has been addressed by several works [10], [1], [11], [2], [12], [3]. The proposed approaches may be classified into two main categories. The first category relies on static analysis of the code source or the bytecode of the applications to identify malicious actions. The second category of approaches relies on dynamic analysis of the applications behaviour and their interaction with the system using information collected from Linux kernel or by the instrumentation of the Dalvik virtual machine.

Batyuk et al [10] has proposed a technique using static analysis of the code source of an Android application. The code source is obtained using reverse engineering tools such as the *Apktool* and *Java Decompiler*. Then several pattern matching algorithms are applied to verify the presence of sensitive functions calls into the code. Finally, they generate a security report regarding the potential of leaking privacy information by the analysed application.

Burguera et al [1] have developed a detection technique based on system call traces performed by an application on a Linux kernel. The traces are collected using a crowdsourcing approach where users deploy their crowdroid application to collect the desired information. The collected data are then analysed and clustered to identify malicious patterns and signatures. The generated patterns are then used to detect other malwares.

The existing approaches available in literature have used several types of information at the application or system levels to analyse and characterize the behaviour of Android applications. We have observed that the collection of such information is usually time consuming and requires heavy techniques, mainly in case of system or virtual machine instrumentation.

III. Analysis Methodology

The characterization of Android applications requires the collection of information related to their execution and their interaction with the system. We have observed that the Android platform is providing a logging system where several interesting information related to applications execution is available and can be used to characterize their behaviour. In this work, our aim is to analyse the available logs provided by the default logging system of the Android platform to characterize running applications on a device. We make use of Self-Organizing Maps (SOM) to project the information available in logs with a 2-dimensional visualization, and application component plane analysis to identify existing correlations between applications components. The adopted methodology is close to the work of Barrera et al [13] where they have used SOM to analyse the permissions of a set of downloaded Android applications. Their goal was to provide insights on how the developers are using the given permission model in their applications.

A. Logging system

Android platform provides a logging system which keeps a record of the execution of different applications and system services running on a device. The logs are giving information about the execution of applications components (started, stopped, resumed, paused and destroyed), their associated linux processes and their interaction with the system services (received broadcasts, received intents, requested permissions, etc). Log entries are recorded and stored within several files. They could be accessed using two available tools provided by the Android platform which are *logcat* and *dumpsys*. The two tools could be executed in the command line interface of a device to view the different logs. In this work, we have developed an Android application which relies on these tools to export the available logs to a server where they are stored for further analysis. The Android application checks periodically the available logs and sends them using the *syslog* protocol to a server where they are stored into a MySQL database. In a next step, a parsing and conversion script is applied on the collected logs from database to analyse them and extract different fields for each log entry. We used a HBase to store the different fields of each parsed log associated to each application run on the user device. The snippet 1 depicts an example of a parsed entry log of the CandyCrush game.

Each row contains a row key identifying the name of the application and the set of parsed fields of the entry log. We find mainly the name of the action performed by the application, for example the creation of an activity and the different parameters associated to this action (the task identifier, the name of the started activity, the corresponding intent). We have observed that after parsing all the available logs of an application, we are able to identify its structure, mainly all the components that have been activated and created during its execution. We were also able using logs provided by the dumpsys tool to identify the interaction of the application with the system during its execution. Figure 2 depicts an example of entry logs obtained using the dumpsys tool. We observe mainly

Snippet 1 An example of a parsed entry log stored with a HBase table.

```
com.king.candycrushsaga00000000578627
    column=component:, timestamp=1372762739311,
    value=am_create_activity
com.king.candycrushsaga00000000578627
    column=datetime:, timestamp=1372762739301,
    value=2013-07-01 14:46:17.390
com.king.candycrushsaga00000000578627
    column=params:Action, timestamp=1372762739316,
    value=android.intent.action.MAIN
com.king.candycrushsaga00000000578627
    column=params:Component Name,
    timestamp=1372762739315,
    value=com.king.candycrushsaga/.
    CandyCrushSagaActivity
com.king.candycrushsaga00000000578627
    column=params:Task ID, timestamp=1372762739313,
    value=72
```

that we are able to obtain the list of permissions granted to the application and also the list of received events from the system. For example, we observe that the application identified with *user id 10190* has received an event indicating that an outgoing call has been made.

B. Dataset

For the empirical analysis of logs, we have used a dataset containing 98 applications obtained from the top free applications available in Google Play Store. The set of applications has been installed on an off-the-shelf Samsung S3 device running the official firmware version 4.1.2 of Android. GPS is activated on the device and it is using a 3G mobile data network connectivity. For each application, we have specified two categories of information. The first category is relative to the set of permissions granted to the applications, where we have identified a total set of 432 permissions. The second category is related to the actions performed by the applications where we have identified a total set of 125 actions. This set of actions includes the creation of activities, services, the reception of intents and several actions related to its interaction with the system. Thus we have characterized each application with a bit vector of size 557 in which each bit denotes where the permission has been requested or an action has been performed. For example Table I presents a partial view of the bit vector associated to the application *com.facebook.katana* which is the android client of the social network FaceBook.

The obtained bit vectors of the application are suitable to be used as an entry for a machine learning algorithm to identify and classify their behaviours and study their relationships. We have made use of Self-Organizing Map algorithm (SOM) [14] to study their behaviour. This algorithm has also been used by Barrera et al [13] to analyse Android permissions where they were able to show trends in Android permissions usage by developers. In this work, we keep the same direction with the goal of better understanding the relationship between the permissions and the action performed by Android applications obtained from the generated logs.

```
Package [fr.collimator.mymalware] (40cbf990):
   userId=10190 gids=[3003, 1015]
   sharedUser=null
   pkg=Package{40b61cd8 fr.collimator.mymalware}
   .....
   dataDir=/data/data/fr.collimator.mymalware
   targetSdk=10
   .....
   grantedPermissions:
     android.permission.READ_PHONE_STATE
     android.permission.READ_SMS
     android.permission.WRITE_EXTERNAL_STORAGE
     android.permission.INTERNET
     android.permission.PROCESS_OUTGOING_CALLS
     android.permission.WRITE_SMS
     android.permission.RECEIVE_SMS
     android.permission.CALL_PHONE
     android.permission.READ_CONTACTS

APP* UID 10190 ProcessRecord{4054da80 11394:fr.collimator.mymalware/10190}
  .....
   pid=11394 starting=false lastPss=0
   ....
   services=[ServiceRecord{40bd11b0 fr.collimator.mymalware/.background}]
   receivers=[ReceiverList{40a98510 11394 fr.collimator.mymalware/10190 remote:40569cb8}]


* ServiceRecord{40bd11b0 fr.collimator.mymalware/.background}
   intent={cmp=fr.collimator.mymalware/.background}
   ....
   app=ProcessRecord{4054da80 11394:fr.collimator.mymalware/10190}
   ....
   startRequested=true stopIfKilled=false callStart=true lastStartId=1

ReceiverList{40a98510 11394 fr.collimator.mymalware/10190
              remote:40569cb8}
   app=ProcessRecord{4054da80 11394:fr.collimator.mymalware/10190}
   pid=11394 uid=10190
   Filter #0: BroadcastFilter{40a985b0}
    ...
    Action: "android.intent.action.NEW_OUTGOING_CALL"
    ...

Historical Broadcast #36:
   BroadcastRecord{40a54748 android.intent.action.NEW_OUTGOING_CALL}
   Intent { act=android.intent.action.NEW_OUTGOING_CALL (has extras) }
   ...
   Receiver #0: BroadcastFilter{40a985b0 ReceiverList{40a98510 11394 fr.collimator.mymalware/
10190 remote:40569cb8}}
```

Fig. 2: A snapshot of logs provided by the dumpsys tool of a running android application.

com.facebook.katana	
Bit vector entry	Associated action or permission
1	am_create_activity
1	am_finish_activity
1	am_pause_activity
1	am_create_service
1	am_destroy_service
1	android.permissions.INTERNET
1	android.permissions.ACCESS_WIFI_STATE
1	android.permissions.ACCESS_FINE_LOCATION
1	android.permissions.BATTERY_STATS
1	android.permissions.CAMERA

TABLE I: A partial view of the bit vector associated to the Android client of the FaceBook application.

C. Self-Organizing Maps

The Self-Organizing Map (SOM) has been proposed by Kohonen [14] as a neural network algorithm for unsupervised learning and data visualization. A SOM is able to map high-dimensional input vectors onto a discrete space usually defined as a map. The map is defined as set of nodes where each region represents an area of the input space. This mapping shows the similarity between input patterns as a proximity on the map. Thus, it provides an understandable tool to capture the properties of Android applications regarding their permissions and actions, and organize their trends. Each node of the SOM is associated with a weight vector that has the same size as the input vector. The learning algorithm iterates over the input vectors where they are presented successively and in each presentation the weight vectors are adjusted. For each input vector the similar weight vector is selected and modified to be more similar to it. Then the neighbors of the best matching weight vector are also adjusted using a learning function which decreases monotonically with the number of iterations to ensure convergence. After the training phase of the neural network, we are able to visualize the map using a D-Matrix (Distance Matrix) [14] which visualizes average distances between a map unit and its topological neighbors. Visualization is realized using colors where for example a red color denotes large distances and blue color denotes a small distance. Another interesting analysis technique is the visualization of component planes which denotes feature maps extracted from the SOM. Each component plane denotes the values of the weight vectors of the map units for each dimension of the vector. It shows the projection of the map for each property, in our case a permission or an action. They are useful to reveal correlation between application properties.

IV. Analysis Results

In a first step, we have only focused on an input bit vector for each application containing only the set of permissions. This first step allows as to verify some results obtained by Barrera et al [13] regarding our dataset. Table II shows the distribution of the requested permissions of our analysed applications. Our first observation is that as stated by Barrera et al, the permission granting internet access is widely requested by the applications. In our dataset, all the applications are requesting this permission. We observe also that permissions related to data storage and location services are also widely requested by the applications. Another interesting observation that has also been made by Barrera et al, is that only few permissions are widely requested by Android applications. In our dataset it is around 25 permissions have more than 10 requesting applications. The rest of permissions are only requested by one or or two applications. Figure 3 shows the D-Matrix visualization of a trained SOM with only permissions bit vector of our dataset. The blue color indicated a small distance and red color indicated a large distance between

Number of requests	Permission
98	$a.p.INTERNET$
94	$a.p.ACCESS\_NETWORK\_STATE$
74	$a.p.WRITE\_EXTERNAL\_STORAGE$
74	$a.p.READ\_EXTERNAL\_STORAGE$
34	$a.p.ACCESS\_COARSE\_LOCATION$
27	$a.p.ACCESS\_FINE\_LOCATION$
20	$a.p.RECEIVE\_BOOT\_COMPLETED$
10	$a.p.WRITE\_CONTACTS$

TABLE II: The most requested permissions by the applications of our dataset.

neighbors. We observe that the blue color is dominating which means that applications are requesting similar permissions. Except, few applications in the half right part of the map where applications are requesting different permissions.

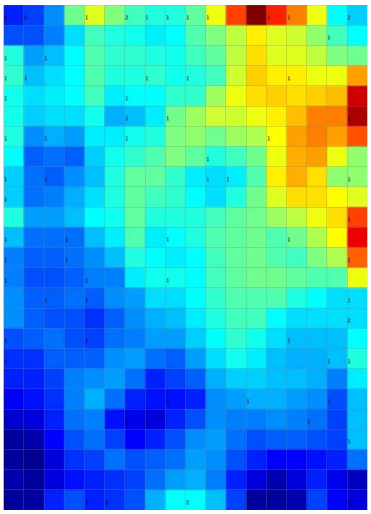
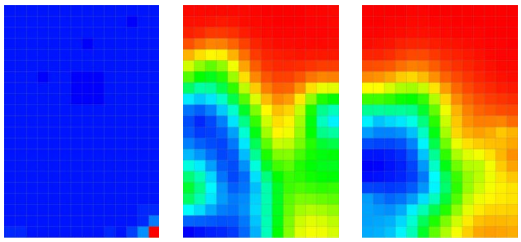


Fig. 3: D-Matrix visualization of the SOM for the requested permissions by the applications of our dataset.

As depicted in Figure 4, component planes analysis confirms the result of Barrera et al regarding permissions usage where we have also observed that the Internet permission covers all the map which means that all the applications are requesting this permissions. We have also observed that several applications are requesting both the two permissions associated to the location service ($a.p.ACCESS\_FINE\_LOCATION$ et $a.p.ACCESS\_COARSE\_LOCATION$).



(a) INTERNET (b) ACCESS_COARSE_LOCATION (c) ACCESS_FINE_LOCATION

Fig. 4: Component plane visualization for various type of permissions.

In a next step, we trained the SOM with a bit vector containing the permissions and the actions performed by the applications extracted from the logs. Table III summarizes the most performed actions by the applications. We observed that the most performed actions are mainly related to activities life cycle regarding their creation, restart, pause and their associated processes.

Figure 5 shows the D-Matrix visualization for permissions and performed actions. We observe that when introducing actions in input vectors, the blue color is still dominating

Hit	Performed action
88	am_proc_bound
88	am_proc_start
83	am_proc_died
81	am_create_activity
81	activity_launch_time
81	am_restart_activity
79	am_pause_activity

TABLE III: The most performed actions by our dataset of Android applications.

which means that applications are granted similar permissions and performing closer actions. In the lower right corner of the map, we observe that a set of 10 applications have created a separated small cluster. This set of applications includes a video player, several games and tools. This set of applications was sparsely located within the SOM depicted in Figure 3. For example the application *com.google.earth* was in the middle of the permissions based SOM, and in logs and permissions based SOM it is moved to this small cluster of applications. The other applications of this small cluster was located in the large blue region at the bottom of the permissions based SOM. We have also observed that the *com.android.chrome* was located in the middle of the permissions based SOM with a color close to the blue and it has a set of permissions close to *org.mozilla.firefox* and *com.android.twitter*. However, in the logs and permissions based SOM, this application is located at the most left middle of the SOM with an orange color. The other two applications are also located at the most middle of the SOM but with colors close to green which mean that they still have a closer behaviour.

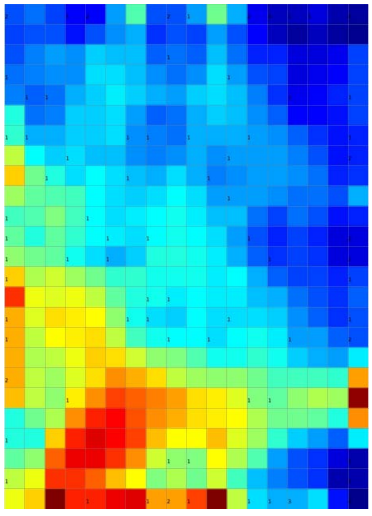
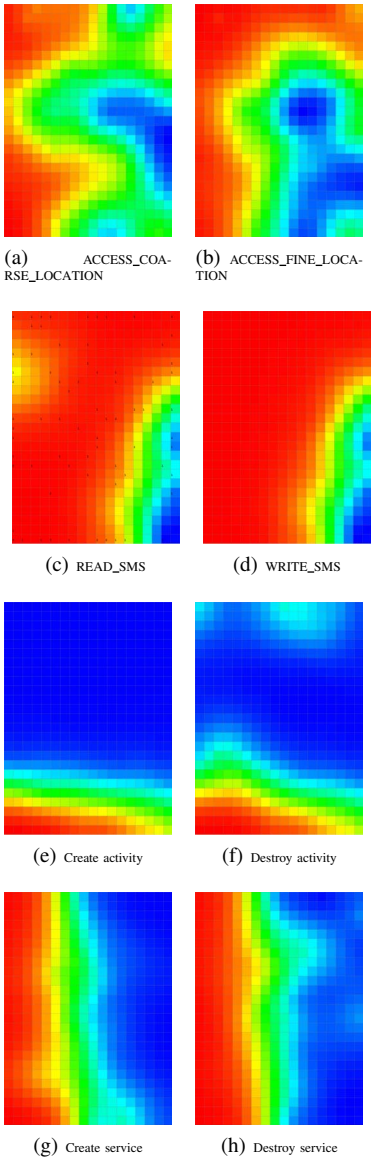


Fig. 5: D-Matrix visualization of the SOM for the requested permissions and actions performed by the applications of our dataset.

Figure 6 shows component plane visualization of various types of performed actions and the requested permissions. We mainly observe that activities related actions are performed by the same applications. This result is trivial since this set of

actions are performed during the a life cycle of an activity. The same observation happen with service creation and termination. We mainly observe that applications requesting location and SMS reading and writing related permissions are mainly using services. This set of application could be dangerous since a service component is not visible to the user, so they are able to perform SMS reading and writing without the user's knowledge.



(a) ACCESS_COARSE_LOCATION (b) ACCESS_FINE_LOCATION

(c) READ_SMS (d) WRITE_SMS

(e) Create activity (f) Destroy activity

(g) Create service (h) Destroy service

Fig. 6: Component planes visualization for various type of actions and permissions.

V. Conclusion and Future work

In this paper, we presented a methodology for the analysis of Android applications behaviour using Self-Organizing Maps. The methodology relies on the extraction of logs provided by the logging system available on the Android platform. The generated logs are exported to a server to be analysed where we extracted the different fields of each log entry. Log entries are then mapped within a bit vector where each dimension denotes a requested permission or an action performed by the application. The obtained vectors are then fed to a SOM for training. The obtained SOM allows us to identify applications trends regarding their requested permissions and their performed actions.

In future work, our goal will be to apply this methodology on known malicious applications to better understand their behaviours and trends.

References

[1] Burguera, Iker, Zurutuza, Urko, Nadjm-Tehrani, and Simin, "Crowdroid: behavior-based malware detection system for android," in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, ser. SPSM '11. New York, NY, USA: ACM, 2011, pp. 15–26.

[2] Yan, L. Kwong, Yin, and Heng, "Droidscope: seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis," in *Proceedings of the 21st USENIX conference on Security symposium*, ser. Security'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 29–29.

[3] Grace, Michael, Zhou, Yajin, Zhang, Qiang, Zou, Shihong, Jiang, and Xuxian, "Riskranker: scalable and accurate zero-day android malware detection," in *Proceedings of the 10th international conference on Mobile systems, applications, and services*, ser. MobiSys '12. New York, NY, USA: ACM, 2012, pp. 281–294.

[4] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets," *Proceedings of the 19th Network and Distributed System Security Symposium (NDSS 2012), San Diego, CA*, February 2012.

[5] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proceedings of the 18th ACM conference on Computer and communications security*, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 627–638.

[6] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," *Proceedings of the 33rd IEEE Symposium on Security and Privacy. San Francisco*, May 2012.

[7] G. Delac, M. Silic, and J. Krolo, "Emerging security threats for mobile platforms," *MIPRO 2011. Opatija, Croatia*, May 2011.

[8] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A survey of mobile malware in the wild," *SPSM11. Chicago, Illinois, USA*, October 2011.

[9] J. Bickford, R. OHare, A. Baliga, V. Ganapathy, and L. Iftode, "Rootkits on smart phones: Attacks, implications and opportunities," *HotMobile10. Annapolis, Maryland, USA*, February 2010.

[10] L. Batyuk, M. Herpich, S. A. Camtepe, K. Raddatz, A.-D. Schmidt, and S. Albayrak, "Using static analysis for automatic assessment and mitigation of unwanted and malicious activities within android applications," in *6th International Conference on Malicious and Unwanted Software (MALWARE 2011)*. Fajardo, Puerto Rico, USA: IEEE Conference Publications, October 2011, pp. 66–72.

[11] Isohara, Takamasa, Takemori, Keisuke, Kubota, and Ayumu, "Kernel-based behavior analysis for android malware detection," in *Proceedings of the 2011 Seventh International Conference on Computational Intelligence and Security*, ser. CIS '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 1011–1015.

[12] Zhou, Wu, Zhou, Yajin, Jiang, Xuxian, Ning, and Peng, "Detecting repackaged smartphone applications in third-party android marketplaces," in *Proceedings of the second ACM conference on Data and Application Security and Privacy*, ser. CODASPY '12. New York, NY, USA: ACM, 2012, pp. 317–326.

[13] D. Barrera, H. G. Kayacik, P. C. van Oorschot, and A. Somayaji, "A methodology for empirical analysis of permission-based security models and its application to android," in *Proceedings of the 17th ACM conference on Computer and communications security*, ser. CCS '10. New York, NY, USA: ACM, 2010, pp. 73–84.

[14] T. Kohonen, M. R. Schroeder, and T. S. Huang, Eds., *Self-Organizing Maps*, 3rd ed. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2001.