Asian Journal of Computer Science and Technology
ISSN: 2249-0701 (P) Vol.4 No.2, 2015, pp.6-12

DOI: https://doi.org/10.51983/ajcst-2015.4.2.1757

Permission Tracking Security Model in Android Application

T.Nandhini[1] and V.Arulmozhi[2]
[1]M.Phil, Scholar, Dept. of Computer Science, Bharathiyar University, Coimbatore, Tamil Nadu, India
[2]Associate Professor, Tiruppur Kumaran College For Women, Tirupur, Tamil Nadu, India
E-mail: nandyarasu@yahoo.in
(Received 21 June 2015; Revised 10 July 2015; Accepted 30 July 2015; Available online 10 August 2015)

Abstract **- Android permissions are rights given to applications to allow them to do things like take pictures, use the GPS or make phone calls. When installed, applications are given a unique UID, and the application will always run as that UID on that particular device. The UID of an application is used to protect its data and developers need to be explicit about sharing data with other applications. Android supports building applications that use phone features while protecting users by minimizing the consequences of bugs and malicious software. Android's process isolation obviates the need for complicated policy configuration files for sandboxes. This gives applications the flexibility to use native code without compromising Android's security or granting the application additional rights. Malicious software is an unfortunate reality on popular platforms, and through its features Android tries to minimize the impact of malware. However, even unprivileged malware that gets installed on an Android device (perhaps by pretending to be a useful application) can still temporarily wreck the user's experience. Applications can entertain users with graphics, play music, and launch other programs without special permissions. In this paper we introduce tracking and monitoring of malicious activity of the apps that are installed by the user even from playstore using trusted permission based security model.**

I. INTRODUCTION

Smartphones are more popular than ever. One of the reasons for this is the fact that the Google Android operating system (OS) is a platform that enables developers to write applications and distribute them for free in an open market. According to the recent analyzes, more than one billion Android devices have been activated with an astonishing growth of 1.4 million devices per day. With this sort of growth, it is absolutely necessary for developers to understand how to create secure Android applications.

With the increasing numbers of applications available for Android; spyware is becoming a most worry. Several malicious applications, rolling from fake banking applications to an SMS Trojan implanted into a fake media player tools, have been detected on the Android Market since the year. Still, there are other classes of malware that might too come forth. What about concealing spyware in the backdrop of a well-known app? For example, think an app talking to be the latest version of a notable Twitter client, which really campaigns spyware in the background and uploads all private data to the attacker.

Google management understood that the iPhone success was largely based on the number of applications released for end-users. Google's resulting strategy is to provide developers with an easy way to develop applications that extend the functionality of the devices, using the Android Software Development Kit (SDK) and the Native Development Kit (NDK). In contrast to Apple, where applications must be downloaded from the Apple AppStore after rigorous control and approval (source code review for potential security problems and copyright infringements), Google arrives easier for developers to publish their applications. The Android application publishing procedure makes it easy to develop Android applications, but also provides room for malicious application publishing. Unlike some of the other platforms, Android does not restrict application distribution via application signing and long approval period. Even though an application has to be signed to be installed on a device, it is possible to use self-signed certificates.

Applications can be granted permissions, which are required to access critical phone resources or for inter-application communication. Those permissions are defined in advance (in the AndroidManifest.xml file), by the developer who wrote the application and permissions are displayed to the user for approval before the application installation [1]. For example, a developer might claim that his application requires complete access to the settings of the phone, access to SMS/MMS reading and so on. So it is up to the user to check the validity of these permissions. In spite of the permissions-based security model implemented by Android, anyone can publish an application on the Android Market, which has no built-in method to detect if this application contains malicious code or not [2].

Past studies on smartphone users' privacy concerns have primarily focused on location tracking and sharing [3, 4, 5, 6]. Although location sharing is an important aspect of smartphone privacy, only 2 of 134 Android permissions pertain to location. Concurrently, Roesner et al. [7] studied user expectations for location, copy-and-paste, camera, and SMS security. Our study encompasses all permissions and focuses on how users perceive the existing permission warnings. In concurrent and independent work, Kelley et al. [8] performed twenty semi-structured interviews to explore Android users' feelings about and understanding of permissions. We propose a application that can track and

monitor the behavior of any installed app with concurrence on permissions approved.

II. COMPONENTS OF ANDROID

A. Activity

In Android development terms, an "Activity" refers to single, focused window that interacts with a user and provides functionality. An activity forms the fundamental building blocks of the application.

An activity has the following states:

Active

When an activity is interacting with a user, it is at the top of the activity stack and visible to the user. Android will kill any other services or activities on the stack to keep the active activity alive.

Paused

This is a state where an activity is not in focus but is actually visible to user. For example, this state is reached when a pop-up appears when activity is running.

Stopped

This is the state where activity is not visible to user, but resides in memory and retaining all data. This activity will be killed to save memory if needed for an active activity.

Inactive

An activity just before launching, after it has been killed, is said to be in an inactive state.

Knowing about the states of an activity allows a developer to understand how data is handled and aids in implementing activities securely.

B. Intents

Intent is commonly used to start an activity or service. Intents can be broadcast and received within the application itself and with other applications. This allows for great flexibility in application development, information sharing, and the ability to trigger operations in other applications.

There are two main types of intents:

Explicit intents:

Explicit intents specify the exact class that needs to be invoked to launch an activity within the application. These are limited to the application context in which they are run.

Implicit intents:

These are the intents that hold information about the type of operation to be performed. It's up to the OS to decide the best operation based on the information provided.

C. Service

A service represents a background operation or an operation that does not require user interaction and takes a lengthy amount of time to complete. These operations are performed without affecting the main application running on the front end. Service continues to run in background, even when application is not running.

Content Providers

Content providers store data persistently. They manage the storage of application data and interact with a number of local SQL databases. Content providers also provide the best means to share data between applications.

WebView

WebViews act like a web browser to display HTML content to the user. Android's WebKit engine is used to display web pages. Any vulnerability found in WebKit directly impacts the WebView. This component allows a user to navigate forward and backward through the history, zoom in and out, and perform text searches, just like Internet Explorer, Firefox, or any other browser.

Permissions

The core security of an application is defined by its permissions. The extent to which an application can perform an action is limited to the permissions defined in its AndroidManifest.xml. By default, every application is sandboxed by the OS and restricts access to the data of another application. At the time of application installation, the user is presented with the list of permissions that are required by the application. Once the user grants those permissions, only then the application will be installed. Granting of permissions dynamically at runtime is not supported.

D. Secure coding Recommendations

Lock-down application permissions

It is necessary to follow the principle of least privilege when assigning permissions. Permissions should not be assigned unless they are required. The application should be granted only the minimum required permissions at the architecture level. For instance, READWRITE permissions should not be granted when only READ permissions are required. This is a common mistake made by developers due to a lack of understanding of the functionality at the application. Examples like these underscore the importance of strong application development planning and requirements documentation.

For example, the Android: protection Level element of the AndroidManifest.xml file defines the protection/risk level associated with the installed application. It also provides the procedure the OS should follow to determine whether the permission can be granted. When the value of the parameter is dangerous, the application, when installed, gains permission to access user data and to control the device. Developers should exercise extreme caution while assigning applications with high-risk permissions.

File permissions

File permissions apply to files stored on external storage. Any file created using openFileOutput is private to the application and cannot be accessed by other applications. Pay close attention before providing a file with the MODE_WORLD_READABLE/MODE_WORLD_WRITABLE permissions. This allows other applications to access the file. Do not provide the writable option until it is required to enforce the principle of least privilege. The

standard way to share a file between applications is to use Content Provider.

III. RELATED WORKS

Enck et al. [9] describe the design and implementation of a framework to detect potentially malicious applications based on permissions requested by Android applications. The framework reads the declared permissions of an application at install time and compares it against a set of rules deemed to represent dangerous behavior. For example, an application that requests access to reading phone state, record audio from the microphone, and access to the Internet could send recorded phone conversations to a remote location. The framework enables applications that don't declare (known) dangerous permission combinations to be installed automatically, and defers the authorization to install applications that do to the user. Ontang et al. [10] present a fine-grained access control policy infrastructure for protecting applications. Their proposal extends the current Android permission model by allowing permission statements to express more detail. For example, rather than simply allowing an application to send IPC messages to another based on permission labels, context can be added to specify requirements for configurations or software versions. The authors highlight that there are real-world use cases for a more complex policy language, particularly because untrusted third-party applications frequently interact on Android.

Research on Android's security infrastructure includes studies on how permissions are enforced [11], used [12], and misused or attacked [13, 14, 15, 16]. Some try to secure Android applications against attackers by performing static or dynamic analysis of apps (ex. [17, 18, 19]). Xu, et al. [20] developed Aurasium, a tool that uses static analysis and code injection to detect or prevent privilege escalation attacks. Like Android, Aurasium does not require modifications to the operating system. Conti, et al. [21] developed Crepe, a system capable of enforcing rule based context aware security policies. Naumann, et al. [22] extended Android permission with custom user defined constraints. None of the above work includes formal analysis or verification.

Research on formalization of the Android stack and API includes Chaudhuri [25] who gave a formal model of a subset of the Android communication system; Enck, et al. [24] who developed TaintDroid to track the flow of sensitive information between Android apps (extended by Shreckling, et al. [25] with more complicated, dynamic run time policies); and Armando, et al. [1] who presented a more complete model of the Android middleware using types. With respect to formalizations for secure sharing of resources, Blanchet and Chaudhuri [23] developed a formally verified protocol for secure file sharing on untrusted storage (a tool which could be used to secure Android's SD card) and Fragkaki, et al. [26] gave formal typing rules to explain Android's security model. Similar to our work, Fragkaki et al. described Sorbet, a modification to Android which enforces secrecy and integrity properties written by app developers. In contrast, Android is developed to enable the easy specification of authorization policies and relies upon existing Android mechanisms without requiring changes to the operating system.

Android Permission Analysis This category includes advancements in analyzing Android permissions. Kirin [31] maps dangerous functionalities with the permissions required to perform them after specifying permission based security rules. Barrera et al. [32] studied the permission usage among a variety of categories of applications in Android market by mapping an application to a category based on its requested permissions. Felt et al. [33] manually compare the functionalities of 36 Android applications to the permissions requested by these applications. Their results show that 4 out of 36 applications are over-privileged. Felt et al. [34] also propose Stowaway, which identifies over-privileged applications by detecting unnecessary permissions for API calls in applications. The mapping provided in [34] is very helpful, but the mapping alone cannot explain the cause and the purpose of the use of permissions.

Smartphone platform security includes a wide range of approaches [35],[36] that aims at improving the security and privacy of smartphone platform. For example, TaintDroid [37], based on the Android platform, provides a scheme to monitor a third-party application's usage of sensitive information such as what information leaves a device and where it is sent. PiOS [38] uses control flow analysis followed by data flow analysis to confirm whether private information reaches outbound sink. Privacy Oracle [37] and TightLip [39] are both black-box-based differential testing schemes for PCs to detect sensitive information leakage by third-party applications via network traffic. These approaches leverage various analysis techniques to enhance the security of the platform.

Application Security includes approaches to study the security of Android applications. For example, Felt et al. [40] propose inter-process communication (IPC) inspection to monitor messages used for IPC and reduces privilege of the recipient to the intersection of recipient's and the requester's permissions. Dietz et al. [41] propose QUIRE to track the call-chain of IPC in order to defend the confused deputy attack and to provide a mutual verification scheme for applications. Chin et al. [42] ComDroid to detect the vulnerabilities in the inter-application communication. Bugiel et al. [43] proposed XmanDroid to prevent privilege escalation. XmanDroid monitors the communications between applications and apply policy to restrict the interaction. Among the most related, Gilbert et al. [44] proposed AppInspector, which leverages information-flow tracking on sensitive information to automatically identify security and privacy violation in an application.

IV. PROPOSED WORK

We propose a framed work for analyzing the use of permissions in android applications. This framework identifies the apps installed and it behavior according to the permission granted upon installation. This real time tracking framework monitors the installed apps for any violation in the permissions agreed. This help user identify the malicious activity and its real intention on the data usage and permissions. Our framework only tracks on the applications in the runtime so it is impossible to conclude that the app is malicious or infected before installing, i.e., we don't tracking the source code level evaluations.

A.Android Defined Permissions

Both the Android system and an application can define permissions, but most of the permissions requested by Android applications are defined by the Android system. This is because Android-defined permissions control the access to sensitive resources and functionalities. There are 130 Android-defined permissions [27], among which 122 permissions are available to third party applications [28]. Permissions are defined with one of the four different protection levels, which characterize the potential risks implied in the permission and enforce different install-time approval processes. These four levels include: 1) Normal 2) Dangerous 3) Signature and 4) SignatureOrSystem. Only dangerous permissions are prompted to users for their explicit approval. Signature permissions are automatically granted when requesting application is signed with the same certificate as the application that declared the permissions. SignatureOrSystem permissions are essentially limited to applications that are pre-installed in Android's "/system" partition OR signed with the firmware key. Normal permissions are always granted by the system automatically. Android-defined permissions are checked when an application tries to interact with the Android API or to access a system content provider or to send and receive specific system Intents.

Android applications are distributed in a compressed file format (i.e., .apk file) that contains a manifest file (i.e., AndroidManifest.xml), compiled Dalvik executables (i.e., class.dex) and other resource files (e.g., files in the "res/"folder). The manifest file not only lists all the permission requests and permission definitions, it also enumerates all the components of the application. The resource files include definitions of UI layouts, application's menu, raw resource files, etc. The information in these files is used to render UIs. Android applications are built upon application components, which include four types: activities, services, content providers and broadcast receivers. Each of these components has its own life cycle and UI. Most of the components can be invoked individually. We focus on activities and service components since most functionality of an application is implemented in these two types of components.

B.Essential of the application

To design a tool that is capable of in-depth analysis of the use of permissions in Android applications, we outline the following design requirements: (R1) Capability to analyze permissions from various aspects (e.g., locations, causes and purposes). Information collected from different aspects can characterize the use of permissions and provide detailed information to users and developers; (R2) Resiliency to static evasions. As we pointed out, existing static analysis-based approaches can be spoofed by inserting unnecessary or unreachable API methods that require sensitive permissions. Therefore, it is important to identify the real "user" of a requested permission and the necessity of the permission; (R3) Capability to analyze different permissions. Different permissions have different purposes. Some permissions protect the access of sensitive information, and some permissions restrict the invocation of sensitive operations. As a consequence, an analysis approach that can only track sensitive information may not be able analyze permissions that do not involve any sensitive information. Given this, the analysis approach should be generic so that it can be applied to various permissions with different purposes; (R4) Scalability to analyze a large number of applications. Since both the number of existing Android applications and the increasing rate of new Android applications are high, the analysis approach should be efficient so that it can analyze millions of applications.

It extracts meta-information (e.g., list of requested Android permissions) about the application. The framework lists the Android API methods, the invocation of which can trigger the permission check, based on the permission-to-API-calls map [29]. After that, it automatically explores the functionality of the application and logs the execution. These log files are then processed using method profiling to identify the permission triggering API calls and to analyze the context of these calls. Based on the call stacks of the permission triggering API calls, Also our application framework can analyze the use of each checked Android permission in terms of location, cause and purpose of the permission use. In the analysis, our framework can also evaluate the potential security/privacy risks in the use of Android permissions.

Our Application explores the functionality of an application by invoking its activity and service components since most application components (i.e., activity and service) can be executed individually. The framework first identifies all the activity and service components, and it then starts each of the identified components individually to reveal the functionality implemented in that component. The identification of activity/ service components is achieved by parsing the meta information stored in the AndroidManifest.xml file. Later our framework parses the ids of "<activity>" and "<service> " tags in the AndroidManifest.xml file. To start an identified component, the application parses the intent-filter and sends out Intent

messages to the target component using the Android debug bridge (adb) console.

Each activity component defines multiple functions that can only be triggered by proper user events. To trigger these functions, the framework first leverages the layout information of the UI. Each activity has its layout information, which specifies the types of the UI elements (e.g., textview, button, etc.) as well as their positions. With the layout information, which is stored in the "/res/layout/", the framework can send specific user events to the positions of the target UI elements so that it can automatically trigger the functions. Then the application uses the adb tool and a testing tool MonkeyRunner [30] to generate and to send user events. However, the positions of some UI elements are not explicitly defined. This is because either the positions are inherited from the parent objects in the view tree, or the positions are relative in order to fit in various screen sizes.
The framework then uses the trackball movement events, the function of which is similar to the Tab order in a form. By sending enough trackball movement events, each UI element that can receive focus will be selected at least once. Following each trackball movement event, the framework sends a set of user events so that it can trigger the function associated with the selected UI element. By applying both activity-based and layout-based UI elements interaction, to automatically explore most of the functionality of an activity component.

C.Permission Use Instance

A permission use instance (i.e., a permission check) consists of two pieces of information: the action/event that triggers the permission check; the type (i.e., activity or service) of the application component where the triggering API method resides in. Our framework determines the type of the component based on the user-defined class that contains the triggering API method and the component type information parsed in the functionality exploration step. That is, the application locates the container user defined class in the component where the class is defined based on the source tree structure of the application.

We are interested in the most direct action/event that causes the permission check. In Android, action/events are processed by corresponding event handlers, the identification of which is not straightforward, since an event handler can be registered in several ways: 1) overriding the default event handler of a View class; 2) registering a customized event listener; 3) implementing a customized event handler. To this end, we leverage the fact that events are program injected (i.e., by MonkeyRunner). More specifically, to identify event handlers, our application instead identifies the event injection methods since these methods are always followed by the corresponding event handlers. After that, the applicatoin traces back the call stack of the triggering API call to determine the most direct event/action that causes the permission check.

D.Purpose of Permission Use

The framework determines the purpose of a permission usage instance from two aspects. First, the functionality of the API call that triggers the permission check. For example, a call to API "android.location LocationManager getLastKnownLocation()" indicates the reason to check the permission "ACCESS_FINE_LOCATION" is to obtain the last known (cached) geographic location information on the smartphone. Meanwhile, a call to API "java.net.HttpURLConnection.<init>" indicates that the reason to request "INTERNET" permission is to start a HTTP connection to a remote server. The information obtained from examining the functionality of triggering API calls is helpful in determining the purpose of permission checks, but not comprehensive. Since one permission check may be related to another check, and only their relation can expose the true purpose of both permission checks. For example, a check of "READ_PHONE_STATE" (e.g, to collect phone identification information) followed by a check of "INTERNET" (e.g., to communicate with a remote server via Internet) suggests that the purpose of both checks is to send collected identification information to a remote recipient.

The framework uses the correlations between multiple permission checks as the second aspect in the analysis of the purpose of permission use. Two API calls are correlated if they appear on the same execution path. The framework discoveries correlations among individually identified permission checks based on the call stacks of their triggering API calls. More specifically, it compares one call stack with another to find a common sub-sequence of calls between them. When correlations are discovered, the application combines the call stacks together to form a new call stack to represent the correlation.

E.Evaluation of Potential risks

Our application evaluates potential risks in permission use by comparing the analyzed instances of permission use to known malicious use patterns. These patterns are obtained from our analysis of malicious applications. Given the large number of applications, the first step in the comparison is to filter out applications that do not request any combination of permissions that is necessary to perform malicious behavior. These combinations are also obtained from our analysis of malicious applications. This step can effectively reduce the number of applications need to be compared. For the rest of the applications, our framework compares the correlations (if any) of their permissions use with a set of correlations of permission use found in malicious applications to determine whether the correlations indicate any malicious behaviors of the application.

V. CONCLUSION

This paper proposes a solution for permission tracking and the use of permissions installed in android applications.

The proposed framework identifies the apps installed and its behavior according to the permission granted upon installation. This real time tracking framework monitors the installed apps for any violation in the permissions agreed. This help user identify the malicious activity and its real intention on the data usage and permissions. Also using this app we can identify the strength and weakness of android app with respect to its functionality and approach on information gathering.

REFERENCES

[1] Google. Android 4.1 Compatibility Definitions. Android Compatibility Program, 7 Sep 2012. Rev 2.

[2] R Xu, H Sadi, and R Anderson. Aurasium: practical policy enforcement for android applications. In 21st USENIX Conf on Security (SEC '12).

[3] L. Barkhuus and A. Dey. Location-based services for mobile telephony: a study of users' privacy concerns. In Proceedings of the International Conference on Human-Computer Interaction, 2003

[4] S. Consolvo, I. E. Smith, T. Matthews, A. LaMarca, J. Tabert, and P. Powledge. Location disclosure to social relations: why, when, & what people want to share. In Proceedings of the ACM CHI Conference on Human Factors in Computing Systems, 2005.

[5] J. Lindqvist, J. Cranshaw, J. Wiese, J. Hong, and J. Zimmerman. I'm the mayor of my house: examining why people use Foursquare - a social-driven location sharing application. In Proceedings of the ACM CHI Conference on Human Factors in Computing Systems, 2011.

[6] P. Kelley, M. Benisch, L. Cranor, and N. Sadeh. When are users comfortable sharing locations with advertisers? In Proceedings of the ACM CHI Conference on Human Factors in Computing Systems, 2011

[7] F. Roesner, T. Kohno, A. Moshchuk, B. Parno, H. Wang, and C. Cowan. User-Driven Access Control: Rethinking Permission Granting in Modern Operating Systems. In Proceedings of the IEEE Conference on Security and Privacy, 2012.

[8] P. G. Kelley, S. Consolvo, L. F. Cranor, J. Jung, N. Sadeh, and D. Wetherall. A Conundrum of Permissions: Installng Applications on an Android Smartphone. In Proceedings of the Workshop on Usable Security (USEC), 2012.

[9] W. Enck, M. Ongtang, and P. D. McDaniel. On Lightweight Mobile Phone Application Certification. In E. Al-Shaer, S. Jha, and A. D. Keromytis, editors, ACM Conference on Computer and Communications Security, pages 235–245. ACM,

[10] M. Ongtang, S. E. McLaughlin, W. Enck, and P. D. McDaniel. Semantically rich application-centric security in android. In ACSAC, pages 340–349. IEEE Computer Society, 2009.

[11] A Felt, E Chin, S Hanna, D Song, and DWagner. Android permissions demystied.In 18th ACM Conf on Computer and Comm Security (CCS '11).

[12] 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 17th ACM Confine Computer and Comm Security (CCS '10).

[13] PH Chia, Y Yamamoto, and N Asokan. Is this app safe? A large scale study on application permissions and risk signals. In WWW '12.

[14] L Davi, A Dmitrienko, A Sadeghi, and M Winandy. Privilege escalation attacks on android. In 13th Intl Conf on Information Security (ISC '10).

[15] A Felt, H Wang, A Moshchuk, S Hanna, and E Chin. Permission re-delegation: attacks and defenses. In 20th USENIX Conf on Security (SEC'11).

[16] P Hornyack, S Han, J Jung, S Schechter, and D Wetherall. These aren't the droids you're looking for: retorting android to protect data from imperious applications. In 18th ACM Conf on Computer and Comm Security (CCS '11).

[17] W Enck, M Ongtang, and P McDaniel. On lightweight mobile phone application certification. In 16th ACM Conf on Computer and Comm Security (CCS '09).

[18] P P F Chan, L C K Hui, and S M Yiu. Droidchecker: analyzing android applications for capability leak. In ACM Conf on Security and Privacy in Wireless and Mobile Networks (WISEC '12).

[19] A Fuchs, A Chaudhuri, and JS Foster. SCanDroid: Automated security certification of android applications. Technical report, U of Maryland College Park, 2009.

[20] R Xu, H Sa□di, and R Anderson. Aurasium: practical policy enforcement for android applications. In 21st USENIX Conf on Security (SEC '12).

[21] M Conti, V Nguyen, and B Crispo. Crepe: context-related policy enforcement for android. In 13th Intl Conf on Information Security (ISC '10).

[22] M Nauman, S Khan, and X Zhang. Apex: extending android permission model and enforcement with user-dened runtime constraints. In 5th ACM Symp on Information, Computer and Communications Security (ASIACCS '10).

[23] A Chaudhuri. Language-based security on android. In ACM SIGPLAN Fourth Workshop on Programming Languages and Analysis for Security (PLAS '09).

[24] W Enck, P Gilbert, B Chun, L Cox, J Jung, P McDaniel, and A Sheth. Taintdroid: an information-how tracking system for realtime privacy monitoring on smartphones. In 9th USENIX Conf on Operating Systems Design and Implementation (OSDI '10).

[25] M Nauman, S Khan, and X Zhang. Apex: extending android permission model and enforcement with user-defined runtime constraints. In 5th ACM Symp on Information, Computer and Communications Security (ASIACCS '10).

[26] E Fragkaki, L Bauer, L Jia, and D Swasey. Modeling and enhancing android's permission system. In ESORICS 2012.

[27] K. W. Y. Au, Y. F. Zhou, Z. Huang, P. Gill, and D. Lie, "Short paper: a look at smartphone permission models," in Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, ser. SPSM '11, 2011, pp. 63–68.

[28] "Androidmanifest.permission."http://developer.android.com/ reference/android/Manifest.permission.html 2012.

[29] 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, 2011, pp. 627–638.

[30] "Android developer: monkeyrunner,"http://developer.android.com/ guide/developing/tools/ monkeyrunner_concepts.html, 2012.

[31] W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," in Proceedings of the 16th ACM conference on Computer and communications security, ser. CCS '09, 2009, pp. 235–245.

[32] 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, 2010, pp. 73–84.

[33] A. P. Felt, K. Greenwood, and D. Wagner, "The effectiveness of application permissions," in Proceedings of the 2nd USENIX conference on Web application development, ser. WebApps'11, 2011, pp. 7–7.

[34] 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, 2011, pp. 627–638.

[35] Y. Zhou, X. Zhang, X. Jiang, and V. W. Freeh, "Taming informationstealing smartphone applications (on android)," in Proceedings of the 4th international conference on Trust and trustworthy computing, ser. TRUST'11, 2011, pp. 9

[36] J. Andrus, C. Dall, A. V. Hof, O. Laadan, and J. Nieh, "Cells: a virtual mobile smartphone architecture," in Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, ser. SOSP '11, 2011, pp. 173–187.

[37] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones," in Proceedings of the 9th USENIX Symposium on Operating System

Design and Implementation, ser. OSDI '10. USENIX, October 2010.

[38] M. Egele, C. Kruegel, E. Kirda, and G. Vigna, "Pios: Detecting privacy leaks in ios applications," in 18th Annual Symposium on Network and Distributed System Security. San Diego, California: Internet Society, February 2011.

[39] A. R. Yumerefendi, B. Mickle, and O. P. Cox, "Tightlip: Keeping applications from spilling the beans," in Proceedings of the 4th USENIX Symposium on Networked Systems Design and Implementation, ser. NSDI '07. USENIX, April 2007.

[40] A. P. Felt, H. J. Wang, A. Moshchuk, S. Hanna, and E. Chin, "Permission re-delegation: Attacks and defenses," in Proceedings of the 20th USENIX Security Symposium, ser.USENIX'11, 2011.

[41] M. Dietz, S. Shekhar, Y. Pisetsky, A. Shu, and D. S. Wallach, "Quire: Lightweight provenance for smart phone operating systems," in 20th USENIX Security Symposium, San Francisco, CA, Aug. 2011.

[42] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner, "Analyzing interapplication communication in android," in Proceedings of the 9thinternational conference on Mobile systems, applications, and services, ser. MobiSys '11, 2011, pp. 239–252.

[43] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischera, and A.-R. Sadeghi, "Xmandroid: A new android evolution to mitigate privilege escalation attacks," Technische Universitat Darmstadt, Center for Advanced Security Research, Tech. Rep., 2011.

[44] P. Gilbert, B.-G. Chun, L. P. Cox, and J. Jung, "Vision: automated security validation of mobile apps at app markets," in Proceedings of the second international workshop on Mobile cloud computing and services, ser. MCS '11, 2011, pp. 21–26.