Permission Tracking in Android

Michael Kern
Micro Focus
Borland Software Corporation
Linz, Austria
michael.kern@microfocus.com

Johannes Sametinger
Dept. of Business Informatics – Software Engineering
Johannes Kepler University
Linz, Austria
johannes.sametinger@jku.at

***Abstract*—Mobile devices get smarter and increasingly provide access to sensitive data. Smart phones and tablet computers present detailed contact information, e-mail messages, appointments, and much more. Users often install apps on their devices to get additional functionality like games, or access to social networks. Too often, such apps access sensitive data and take privacy less serious than expected by users. In this paper, we will have a closer look at permissions that users grant to apps in Android, a wide-spread operating system for mobile devices like smart phones. As it turns out, Android does not provide sufficient control to their users about what apps are allowed to do. We demonstrate the feasibility of a permission tracking functionality, but conclude that thorough modifications in Android itself will be necessary to provide satisfying control of apps' permissions and users' privacy.**

Keywords-Android, mobile devices, privacy, permissions, tracking.

I. INTRODUCTION

Almost everything that users can access on their desktop can also be accessed on their mobile devices, in particular on smart phones. These devices have become powerful with capabilities that desktops did not even have several years ago. Increased capabilities have come hand in hand with increased security threats. Nowadays, all our private and business data is accessible on our phones. Thus, these devices increasingly have become the targets of malicious attacks. The most successful and most widespread operating system of today's mobile devices is Android, with Apple's iOS following second [1]. Needless to say, the operating system plays a key role for the security and for the privacy of these devices. Permissions play a major role for apps on Android. Giving permissions to apps can lead to data leaks because it may, for example, allow these apps to access contact data and to access the Internet. A malicious app may thus send all our contact data to a server on the Internet. Fine-grained permission setting is not (yet) possible in Android.

In this paper, we will introduce Android and its security features. We will then suggest a mechanism to track permissions in Android. For that purpose, we have developed an Android application that supports permission assignment, permission tracking and permission notifications. In Section II, we will provide an overview of the Android operating system. Section III follows with an overview of security mechanisms of Android. In Section IV, we will provide more details of Android permissions. The *PermissionTracker* tool will be presented in Section V. Implementation aspects follow in Section VI. A comparison with other approaches is given in Section VII. Performance issues are discussed in Section VIII. Eventually, a conclusion ends the paper in Section IX.

II. ANDROID

Android is a Linux-based operating system for mobile devices like smart phones and tablet computers. It is developed by the Open Handset Alliance led by Google. Android apps can be downloaded from online stores like Google's app store *Google Play* (formerly *Android Market*) and also from third-party sites.

Google leads the Android Open Source Project (AOSP) with the goal "to create a successful real-world product that improves the mobile experience for end users" [2]. Since its original release, there have been many Android updates, each of which fixed bugs and added new features. The main Android building blocks are device hardware, the Android operating system, and the Android application runtime [2]. Android supports a wide range of hardware configurations, e.g., smart phones, tablets, or set-top-boxes. Even though Android is processor-agnostic, it does take advantage of some hardware-specific features, e.g., security capabilities such as the no-execute page protection of the ARM architecture.

The core system is built on top of the Linux kernel. All device resources, like camera, GPS, Bluetooth, telephony, network connections, are accessed through the operating system. Most Android applications are written in the Java programming language. Core Android services and applications are native applications or include native libraries. The virtual machine and native applications run within the same security environment, contained within the application sandbox. Applications get a dedicated part of the file system in which they can write private data. Android applications are either pre-installed or user-installed. Pre-installed applications like phone, email, calendar, web browser, and contacts provide key device capabilities that can be accessed by other applications. Development of user-installed applications is supported by an open development environment.

Google also provides cloud-based services for Android devices. They include services to let users discover, install, and purchase applications, update services, and application services that, for example, easily let application developers backup user data in the cloud.

 ISBN: 978-1-61208-236-3

III. ANDROID SECURITY

The Android platform security architecture provides several key security features in an effort to protect user data as well as system resources, and to isolate applications from each other. To achieve these goals, Android provides a sandbox for all applications, secure inter-process communication, and application signing, among others. User-granted permissions at the application level are also part of the security architecture. They are central for the content of this paper and, thus, will be described separately in Section IV. In the following subsections, we describe basic security risks and Android security mechanisms.

A. Security Risks

Security risks originate from various sources, e.g., from the operating system or from communication mechanisms. Android consists of a Linux kernel and various open source libraries implemented in C/C++. The effect of open source to security is being discussed contradictory. C/C++ is known to be more prone to vulnerabilities than other languages like Java or C#. A static security analysis of version 2.2 has revealed several hundred errors [3]. Almost 100 of these were rated critical, leading to crashes or constituting potential vulnerabilities.

Smart phones have numerous wireless interfaces like WLAN, Bluetooth, NFC, and of course GSM and UMTS or 3G. If confidential information is transmitted by the user or by an app, then attackers can easily listen in with a simple antenna. Attackers having direct access to a stolen or lost device are seen as one of the biggest threat. Data is not encrypted by default. Devices are used for critical services like online banking or virtual private networks (VPNs) to access sensible information.

Unlike Apple, Google does not impose control mechanisms for apps to be published on the market. If users report about malicious activities, then apps get removed from the market. Until that time, users are exposed to these malicious apps. Unaware users may also deactivate security functions countertrading enhanced usability. They may leave their device unattended or they may permanently turn on services like Bluetooth or WLAN. They may also install apps from un-trusted sources.

B. Sandbox

Android applications run in a sandbox, i.e., an isolated area of the operating system with no access to the rest of the system's resources. Access is granted only when users warrant explicit access permissions during application installation. Before installation, all the required permissions are displayed.

C. Inter-process Communication

Android processes can communicate with any of the traditional UNIX-type mechanisms, e.g., file system, sockets, signals. There are additional Android-specific mechanisms for inter-process communication. Google recommends that Android developers use these mechanisms, i.e., binders, services, intents, and content providers [3].

Binders are lightweight remote procedure call mechanisms that are designed for high performance for in-process and cross-process calls. Android services run in the background. They can run in their own process or in the context of another application's process. Services can provide interfaces that are directly accessible through binders. Intents are simple message objects that represent the intention to do something. For example, if an application wants to display a web page, it creates this intent to view a specific URL and hands it off to the system. The system locates the browser that knows how to handle that intent and runs it. Intents can also be used to broadcast system-wide events, e.g., notifications. A data storehouse provides access to data on Android devices, e.g., the user's list of contacts. Applications can access data that other applications have exposed via a content provider. Applications can also define their own content provider and expose data of their own.

Android developers are encouraged to use best practices to secure users' data and avoid the introduction of security vulnerabilities [4].

D. Application Signing

Android applications must be signed by their developers. Code signing allows users to identify the authors of applications and to update applications without having to deal with permissions again. Unsigned applications that attempt to install will be rejected by either the Google app market or by the package installer on the Android device. In addition to Android's built-in security features, users may use software by various vendors. Solutions are available for tasks like access control, data encryption, traffic counting, anti-theft (device locking, device location, data wiping), and malware protection. Well-known vendors for PC security solutions like BitDefender or Kaspersky also offer products for mobile devices with the Android operating system.

IV. ANDROID PERMISSIONS

A game may, for example, need to activate vibration but should not need to read messages or access contact information. After reviewing the permissions, users can decide whether to install an application [4]. Protected resources include camera, location data (GPS), Bluetooth, telephony, SMS/MMS, and network/data connections. Granted permissions are applied to applications as long as they are installed. Android's permissions are some form of Mandatory Access Control, or MAC for short. In contrast to DAC which stands for Discretionary Access Control, access is not controlled by users or by user ids, but rather by permission labels that are assigned system functions. Accessing a resource requires the call of system functions. If an application wants access to a resource, it needs the permissions required by the appropriate system functions; see Figure 1 [4].

 ISBN: 978-1-61208-236-3

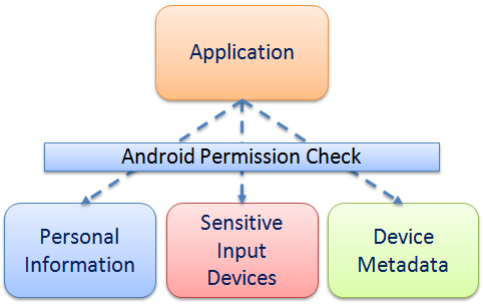

Figure 1. Access to sensitive data through protected APIs [4].

Protected APIs include [4]:

- Camera functions
- Location data (GPS)
- Bluetooth functions
- Telephony functions
- SMS/MMS functions
- Network/data connections

Required permissions of applications are stored in their manifest file; see Figure 2. The application described in the manifest of Figure 2 needs to send and receive SMS messages, as well as access to the user's contact list. During installation the user gets informed about the permissions that are requested by an application. A dialog will show that, for example, the application requests access to services that may cost money, access to the phone's location, to network communication, to account information, and to storage. We can trust the application and install it or we can cancel the installation process as a whole. There is no way in between like installing the application but denying access to account information. The package installer is the single point of interaction with the user; no further checks with the user are done while an application is running.

The Android operating system defines over one hundred different permissions. They are called standard permissions and distinguish various functions of the protected APIs mentioned above, for example, reading contacts, writing contacts, sending SMS, receiving SMS.

```
<manifest package="com.example.myapp">
    <application />
    ...
    <uses-permission android:name=
      "android.permission.SEND_SMS" />
    <uses-permission android:name=
      "android.permission.RECEIVE_SMS" />
    <uses-permission android:name=
      "android.permission.READ_CONTACTS" />
    …
</manifest>
```

Figure 2. Permission declaration in manifest.

A. *Application-defined permissions*

Permissions may also be defined by applications. Thus, developers can restrict access to their applications, i.e., to their activities, services, broadcast receivers, and content providers. The declaration of these application-defined permissions is again written in the application's manifest file. Figure 3 shows an example permission with the name "my.permission.MY_PERM". Other application may request this permission during installation and, thus, get access to activities and services of our sample application that are protected with this permission.

B. *Protection level*

The protection level specified in Figure 3 characterizes the potential risk that is implied in the permission [5]. This level indicates the procedure the system should follow when determining whether or not to grant the permission to an application requesting it. The value can be set to either *Normal*, *Dangerous*, *Signature*, or *SignatureOrSystem* [5]. *Normal* is the default value. Access is granted to isolated application-level features, with minimal risk to other applications, the system, or the user. Access is automatically granted to requesting applications. The protection level *Dangerous* provides access to private user data or control over device. It introduces a potential risk and, therefore, is not automatically granted to requesting applications. If *Signature* is specified, access is granted if a requesting application is signed with the same certificate as the application that declared the permission. The level *SignatureOrSystem* grants access to applications that are in the Android system image or that are signed with the same certificates as those in the system image. This level is used for certain special situations.

C. *Drawbacks*

There are some drawbacks of the Android permission system.

1) *Static permissions*

Android's permission system is rather rigid and lacks flexibility. Users can only install applications by granting all permissions requested by that application. It is not possible to withdraw any permission, neither during installation nor after the installation process. The only option users have is to uninstall an application.

```
<manifest package="com.example.myapp">
    <application />
    ...
    <permission
      android:name="my.permission.MY_PERM"
      android:protectionLevel="normal"
      android:label="@string/myPerm_Label"
      android:description=
        "@string/myPerm_Description">
    </permission>
    …
</manifest>
```

Figure 3. Declaration of an application-defined permission.

 ISBN: 978-1-61208-236-3

a) b) c)

Figure 4. a) PermissionTracker showing permissions' applications. b) Granted, denied and blocked permissions. c) Permission request

2) Missing control

Users have no control over their resources. Once an application has been installed, it can access resources with the permissions that have been granted during installation. Users cannot neither watch which resources an application accesses, nor can they permit or deny any such access.

3) Over-privileged applications

Applications sometimes are over-privileged, which means they require access to resources they do not need to function. Over-privileged applications increase the impact of vulnerabilities.

4) Permission granularity

Some standard permissions are defined at a coarse granularity, e.g., INTERNET, WRITE_EXTERNAL_STORAGE. Applications with the permission INTERNET have arbitrary access to the Internet. There is no way to restrict access for example to specific domains or services.

5) Permissions across applications

Applications that have been signed with the same certificate and have the same user id can share their permissions. Shared user ids may grant permission to applications without explicitly declaring them in the application's manifest file. Applications may also combine their permissions. For example, application A may have access to the user's contact data but no access to the Internet. Application B may have access to the Internet but no access to the user's private data. This inoffensive situation may become threatening when application A hands over sensitive data to application B which in turn may send it to a server on the Internet.

V. PERMISSION TRACKER

We have developed an Android application that allows users to administer permissions of their applications. We have extended the existing permission concept and enable users to allow or deny permissions at any time. Additionally, we facilitate the observation of application's access to resources.

The *PermissionTracker* tool provides three consistent views with different levels of detail. Users can view application categories and inspect the permissions of single applications or groups of applications. Users can alternatively view permission categories and inspect applications with specific permissions or permissions in specific permission groups. At any time, users can modify the permissions of applications or groups of applications. Check boxes are available and can be selected individually for granting or denying permissions, for blocking access to resources, for monitoring access to resources, and for sending notifications when a resource is being accessed. If users block access to a resource, they will be asked for approval every time an application wants to access the resource. If access to a specific resource by a specific application or a group of applications is monitored, then statistics with information about resource access will be available.

The *PermissionTracker* also allows a detailed view with information about specific permissions, including the protection level, see Figure 4a. We can also see in Figure 4b the number of granted, denied and blocked access for the "send SMS messages" permission. The table in the bottom of the figure shows the numbers for today, for this week, this month, and the total number. These numbers get collected only if monitoring has been activated.

Notifications inform users immediately when an application requests access to a specific resource. Figure 4c shows the dialog that appears when the user opens a notification. We can see that the application SMS Messenger requests the permission to send SMS messages. We can either grant or deny this permission. If the user does not react to a notification, for example, because the phone has been left home,

 ISBN: 978-1-61208-236-3

then the dialog will close automatically after five minutes. In this case, access will be denied.

PermissionTracker generates several reports to ease an analysis of applications and permissions. The reports are created in HTML format and either shown in the browser or sent to an email address. Reports include permissions of applications, the permissions' status at the time of access, as well as date and time of access, see Figure 5 for an example.

VI. Implementation Aspects

In Section IV, we have described Android's permission mechanism. The regular Android permissions do not support the implementation of an application like the *PermissionTracker*. Therefore, a few adaptations had to be made in the Android system. Subsequent sections describe these modifications.

A. Android

Android developers use an application framework that serves as a layer between applications and the mobile device. These Android APIs, i.e., short for application programming interfaces, support the creation of GUI elements, data repositories, data communication, etc. The manager classes get started during initialization of Android and run in separate threads. Figure 6 shows part of these manager classes. The first two boxes border the package manager service and the activity manager service. The activity manager interacts with the overall activities running in the system. It is responsible to consider permissions when components of applications interact among each other or access components of Android. The package manager retrieves various kinds of information related to packages of currently installed applications on the device. It stores the permissions that applications request during their installation and provides functions to application developers to retrieve information, for example, about these permissions. In order to implement the functionality of our *PermissionTracker* application, we had to add two functions to this manager, i.e., a block permission dialog and a permission notification, see the bottom box in Figure 6.

User settings about permission tracking are stored in a file app_configuration.xml in the directory of the *PermissionTracker*. As mentioned above, these user settings contain information about whether a user wants to individually grant or deny access to a resource, whether notifications should be sent upon access to a resource, or whether access should get logged.

The Android Activity Manager interacts with the overall activities running in the system. For example, it returns attributes of the device configuration or information about the memory usage of running processes [6]. Our extension to the Activity Manager checks the permissions and performs monitoring and sending notifications if necessary. The Android Package Manager can be used to retrieve information about application packages that are installed on the device [7]. This information includes permissions that applications have assigned to. However, the package manager does not contain any methods that allow the modifications of permissions. This is not necessary as current Android implementations assign these permissions during the installation of an application. Later modifications have not been planned so far. The implementation of our *PermissionTracker* needs such methods. Therefore we have simply added them, i.e., methods to grant, to revoke and to log permissions.

B. Installation

Due to the extensions made to Android, the *PermissionTracker* cannot be installed on a system without these extensions. Care has to be taken to use a device where an original Android version is installed. If the installed Android contains extensions of the device's manufacturer, these extensions would be overwritten when installing the new Android version for the *PermissionTracker*. For details about how to install an Android build see [8].

A device's operating system that comes when buying it is called stock ROM. A custom ROM is a version of Android that includes the kernel, apps, and services, i.e., everything

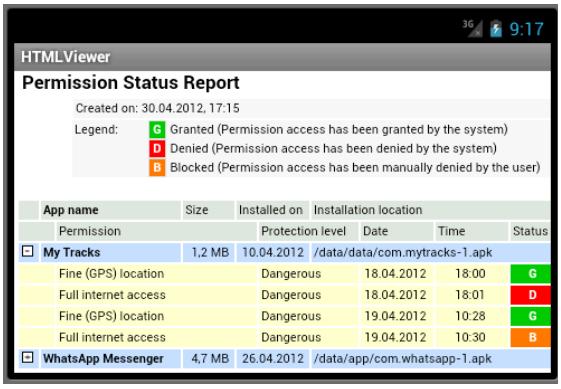

Figure 5. Permisson status report.

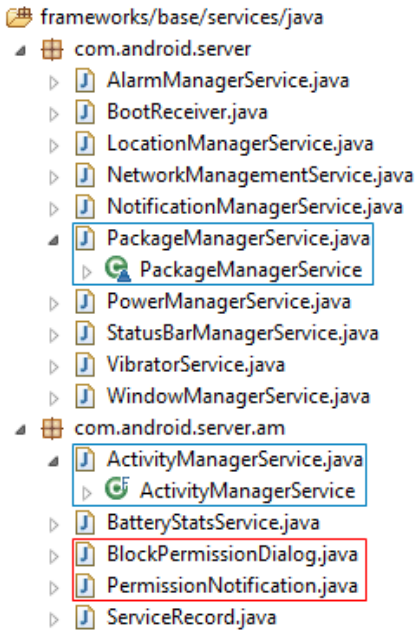

Figure 6. Android's manager classes.

that is needed to operate the device. As the name suggests, this version has been customized by someone in some way. These custom ROMs can be installed via the recovery console. There are some downsides to custom ROMs. For example, when installing a new version with a custom ROM, then all existing data is lost. Thus, installation a new device may be easy, but doing such an install at a later point of time may be more sophisticated [9][10].

VII. RELATED WORK

Weaknesses of the Android permission system have been known for a while. Concepts and tools have been suggested for improvements. We will start with extensions to Android, continue with applications and conclude with a comparison to our approach.

A. Android Extensions

APEX stands for Android Permission Extension. It is a framework that allows users to specify runtime constraints to restrict applications' access to resources. Users can specify constraints through a simple interface of the extended Android installer. The extensions are incorporated in Android with little changes of the code and the user interface [11]. Kirin is an alternate application installer and security framework to Android. It ensures that applications meet predefined security requirements. Applications are rejected, if any specified requirements are not met. Devices stay in a secure state without users having to make any security decisions. Challenges include the modeling of security mechanisms and the acquiring of appropriate policy primitives [12].

SAINT stands for Secure Application Interaction, a mechanism that allows application developers to define install-time and run-time constraints. Policies can be specified by application developers, but not by users [13]. MockDroid is also a modified version of Android. It allows users to 'mock' applications' access to resources. Mocking means that resources are reported as empty or unavailable whenever an application requests access. In contrast to Android's static permission system, users can revoke access to specific resources at run-time [14]. XMandroid stands for eXtended Monitoring on Android. It performs runtime monitoring and analysis of communication across applications in order to prevent potentially malicious control and data flow based on a defined policy [15].

TISSA stands for Taming Information-Stealing Smartphone Applications. It is a system that implements a privacy mode that empowers users to flexibly control application's access to personal information in a fine-grained manner. Granted access can be dynamically adjusted at run-time [16].

B. Applications

There are several applications available that help in getting an overview about the permissions that installed applications request. Some of these applications also allow to grant or to deny access to specific resources. aSpotCat is an ad-supported application that eases the process of finding out the permissions that specific applications have. It also provides lists of applications that use specific resources. For example, which applications use GPS or SMS? [20]

PermissionDog is an application that lists applications and checks the permissions they are using. Based on these permissions, PermissionDog rates the potential danger of applications. If wanted, every time an application is launched, a notification pops up and shows the number of used permissions and the potential danger of the application [21]. PermissionsDenied is an application that lists the amount of active and disabled permissions of each application on the device. Permissions can be enabled and disabled. Any changes require a reboot of the device in order to become effective. Brief descriptions of permissions can also be shown [17][22].

LBE Privacy Guard features a back-ground service that constantly monitors applications' activities. Users get alerted whenever an application attempts to access a sensitive resource like the location, the phone ID or the Internet. The requested access can then be permitted or denied by the user [17][23][24]. CyanogenMod offers a variety of features and enhancements to Android. Among others, permissions can dynamically be granted and denied [18]. WhisperCore is a security application for Android with firewall and encryption functionality. It also offers an extended permission mechanism. Similar to MockDroid, access to resources is not blocked, but results in empty or dummy data [19].

C. Comparison

Table 1 shows a comparison of the solutions introduced in this section. We use the following criteria for the comparison.

1) Availability

Not all the solutions that we have introduced are available for end-users. Some are available as applications (marked with an A in Table 1) while others can be installed as Custom ROM (marked with CR). *PermissionTracker* is a simple application but has to be installed as Custom ROM, because it depends on extensions in Android. The same holds for MockDroid.

2) Modified Android

Modifications or extensions in the Android source code are required by several solutions, because the original Android's permission mechanism is too simple to provide enhanced permission functionality. Some applications operate with the original Android system and, thus, can provide only limited functionality like listing permissions that had been requested and granted to applications during their installation process.

3) Policy

Control of resource access is based on policies by the system. The end-user does not have control over permissions other than to change policies of the system. This is in contrast to manual control, where the user has direct control over the permissions of single applications.

4) Conditions

Restriction of resource access can be controlled by the definition of specific conditions. This is only possible in APEX, where users have fine-grained control about permissions of applications.

 ISBN: 978-1-61208-236-3

TABLE I. COMPARISON

	APEX	Kirin	SAINT	MockDroid	XmanDroid	TISSA	aSpotCat	Permission Dog	Permissions Denied	LBE Privacy Guard	Cyanogen Mod	WhisperCore	PermissionTracker
1. Availability	-	-	-	CR	-	-	A	A	A	A	CR	CR	CR
2. Modified Android	X	X	X	X	X	X	-	-	-	-	X	X	X
3. Policy	-	X	X	-	X	-	-	-	-	-	-	-	-
4. Conditions	X	-	-	-	-	-	-	-	-	-	-	-	-
5. Blocking	X	-	X	-	X	-	-	-	X	X	X	-	X
6. Mocking	-	-	-	X	-	X	-	-	-	-	-	X	-
7. User confirmation	-	-	-	-	-	-	-	-	-	X	-	-	X
8. Logging	-	-	-	-	-	-	-	-	-	X	-	-	X
9. GUI	-	-	-	-	-	-	AP	AP	A	A	-	-	AP

5) Blocking

Permission to access a specific resource may be blocked by raising a security exception. It depends on an application how it handles such a situation. It may crash, close down or continue to run.

6) Mocking

An alternative to blocking access to a resource is to provide dummy or mock data. Thus, an application will not realize that it won't get access to real data and will continue to run as expected. We have not yet implemented this feature, but believe that this will be necessary. Tricking applications will prevent them from not functioning as expected.

7) User confirmation

In some situations users may prefer to explicitly confirm access to a resource by a specific application. Doing so for all applications and all permissions is too much trouble. But it is useful to have control over specific resources, especially when a new application has just been installed and is not (yet) completely trusted by the user.

8) Logging

Logging the access to resources allows for later inspection. Logs make it easy to find out which resources really had been accessed by specific applications. Logging is only provided by LBE Privacy Guard and *PermissionTracker*.

9) GUI

Android extensions typically provide some minor extensions to the Android permissions user interface. Permission applications provide extended user interfaces. For example, grouping applications and permissions into categories, which makes it easier to grant or deny resources. aSpotCat, PermissionDog and *PermissionTracker* support categories for applications and for permissions (AP in Table 1). PermissionDenial and LBE Privacy Guard provide categories for applications only (A in Table 1).

As can be seen in Table 1, *PermissionTracker* is an application that needs an extended Android version. Its permission control is based on user manipulation rather than the definition of policies and it provides blocking of permissions. It stands out by providing dynamic user confirmation and logging which is also available for the LBE privacy guard. However, the LBE privacy guard offers less functionality because it is based on an unmodified version of Android.

VIII. PERFORMANCE

Performance is a crucial issue that has to be taken into account for any changes to the existing permission control mechanism, as users are typically not willing to sacrifice performance for security. With *PermissionTracker*, users can define policies for apps to manage and monitor access to features. These policies are stored in an Xml file that is read into memory during each system startup. If an app wants to access certain permissions, the policy as specified in the *PermissionTracker* is evaluated.

We have measured the performance overhead of our code changes on the Android emulator with Android 2.3.6. 45 apps were installed on the system. For that purpose, we have exemplarily measured the time taken to resolve the permission checks for sending SMS (SEND_SMS) and for accessing GPS data (ACCESS_FINE_LOCATION). In particular, we have timed the intervals for checking permissions in the existing security mechanism of Android, in our modified permission checks and in our modified permission checks where we log permission access.

The difference between the time taken by the original permission-check mechanism of Android and that of our system enhancement is rather small, an average of 3.5 milliseconds for both permissions. When logging is activated, the additional time needed for policy evaluation increases to an average of 35 milliseconds. This is mainly caused by the fact that each permission access is written to a log file in the working directory of *PermissionTracker*. All in all the enhancements to the existing permission checking model of

 ISBN: 978-1-61208-236-3

Android are efficient and require only a little performance overhead that is not noticeable to users.

IX. Conclusion and Future Work

We have introduced Android with its security mechanisms. The permission system is rather rigid in Android and suffers from a few drawbacks. We have developed a more flexible permission system with a permission tracker tool. The system allows users to block and monitor access to resources by arbitrary applications. For implementation purposes, the Android system had to be slightly extended.

The use of an app like the *PermissionTracker* application is recommended for users who want to control access to their resources on a finer granularity than what is currently possible with Android. Android users who want to have advanced control over permissions granted to their apps have three options. First, they can download and use one of the permission apps with limited functionality. This helps in getting a better overview about permissions granted to apps, but does not prevent privacy breaches. Second, users can use a modified version of Android with a better handling of permissions. This provides more protection against privacy breaches. However, installing a modified version of Android is only practicable when activating a new device. The installation process would remove any settings and user data on a device that is already in use. In order to empower all end-users with flexible permission control, it will be necessary to include more flexible permission control into the regular Android system.

References

[1] IDC - Press Release. Android- and iOS-Powered Smartphones Expand Their Share of the Market in the First Quarter, According to IDC. 24 May 2012. www.idc.com/getdoc.jsp?containerId=prUS23503312 [retrieved: Aug., 2012]

[2] Android Open Source Project. Android. source.android.com/ [retrieved: Aug., 2012]

[3] Coverity Scan: 2010 Open Source Integrity Report. Featuring the Coverity Software Integrity Report for the Android Kernel. www.coverity.com/library/pdf/coverity-scan-2010-open-source-integrity-report.pdf [retrieved: Aug., 2012]

[4] Android Developers . Permissions. developer.android.com/guide/topics/security/security.html [retrieved: Aug., 2012]

[5] Android Developer. The AndroidManifest.xml File. developer.android.com/guide/topics/manifest/manifest-intro.html [retrieved: Aug., 2012]

[6] Android Developer. ActivityManager. developer.android.com/reference/android/app/ActivityManager.html [retrieved: Aug., 2012]

[7] Android Developer. PackageManager. developer.android.com/reference/android/content/pm/PackageManager.html [retrieved: Aug., 2012]

[8] Android Open Source Project. Initializing a Build Environment. source.android.com/source/initializing.html [retrieved: Aug., 2012]

[9] Artem Russakovskii. Custom ROMs For Android Explained - Here Is Why You Want Them. www.androidpolice.com/2010/05/01/custom-roms-for-android-explained-and-why-you-want-them/ [retrieved: Aug., 2012]

[10] Android Code. Installing the Latest Custom ROM. code.google.com/p/android-roms/wiki/Install_Custom_ROM [retrieved: Aug., 2012]

[11] Mohammad Nauman and Sohail Khan. Design and Implementation of a Fine-grained Resource Usage Model for the Android Platform. Department of Computer Science, University of Peshawar, 2010, csrdu.org/pub/nauman/pubs/apexext-iajit10.pdf [retrieved: Aug., 2012]

[12] William Enck, Machigar Ongtang and Patrick McDaniel. Mitigating Android Software Misuse Before It Happens. Department of Computer Science and Engineering, Pennsylvania State University, 2008, www.enck.org/pubs/NAS-TR-0094-2008.pdf [retrieved: Aug., 2012]

[13] William Enck, Patrick McDaniel, Stephen McLaughlin and Machigar Ongtang. Semantically Rich Application-Centric Security in Android. Department of Computer Science and Engineering, Pennsylvania State University, 2010, www.enck.org/pubs/acsac09.pdf [retrieved: Aug., 2012]

[14] Alastair Beresford, Andrew Rice, Nicholas Skehin and Ripduman Sohan. MockDroid: Trading privacy for application functionality on smartphones. Computer Laboratory, University of Cambridge, 2011, www.cl.cam.ac.uk/~acr31/pubs/beresford-mockdroid.pdf [retrieved: Aug., 2012]

[15] Sven Bugiel, Lucas Davi, Alexandra Dmitrienko, Thomas Fischer and Ahmad-Reza Sadeghi: Android Security XManDroid: A New Android Evolution to Mitigate Privilege Escalation Attacks: System Security Lab, Technische Universität Darmstadt, 2011, www.informatik.tu-darmstadt.de/fileadmin/user_upload/Group_TRUST/PubsPDF/xmandroid.pdf [retrieved: Aug., 2012]

[16] Vincent Freeh, Xuxian Jiang, Xinwen Zhang and Yajin Zhou. Taming Information-Stealing Smartphone Applications (on Android). Department of Computer Science, NC State University, 2011, www.csc.ncsu.edu/faculty/jiang/pubs/TRUST11.pdf [retrieved: Aug., 2012]

[17] CNET.de. Permissions Denied für Android: Alle App-Berechtigungen voll im Griff. (in German) www.cnet.de/blogs/mobile/android-app/41552649/permissions_denied_fuer_android_alle_app_berechtigungen_voll_im_griff.htm [retrieved: Aug., 2012]

[18] CyanogenMod. CyanogenMod Wesbsite. www.cyanogenmod.com/ [retrieved: Aug., 2012]

[19] Whisper Systems. Selective permissions for Android. whispersys.com/permissions.html [retrieved: Aug., 2012]

[20] Sam Lu. aSpotCat (app by permission). play.google.com/store/apps/details?id=com.a0soft.gphone.aSpotCat [retrieved: Aug., 2012]

[21] Android Freeware. PermissionDog. www.androidfreeware.net/download-permissiondog.html [retrieved: Aug., 2012]

[22] Google Play. Permissions Denied. play.google.com/store/apps/details?id=com.stericson.permissions [retrieved: Aug., 2012]

[23] Google Play. LBE Privacy Guard. play.google.com/store/apps/details?id=com.lbe.security.lite [retrieved: Aug., 2012]

[24] Sameed Khan. LBE Privacy Guard For Android Monitors Access Requests, Guards Privacy. addictivetips. www.addictivetips.com/mobile/lbe-privacy-guard-for-android-monitors-access-requests-guards-privacy/ [retrieved: Aug., 2012]

 ISBN: 978-1-61208-236-3