Dynamic Permission Mechanism on Android

Lukas Aron, Petr Hanacek
{iaron, hanacek}@fit.vutbr.cz

Faculty of Information Technology, Brno University of Technology
Bozetechova 2, Brno, Czech Republic

Abstract: This paper discusses the implementation of protecting user's data on mobile device with an Android platform. The mobile platform surpasses computers in its popularity in many aspects of one daily routine. The dynamic permission mechanism tracks the data flow of the application and restricts the application permission rules related to a input of the application. The implementation shows the example of protection against data leakage from mobile device. This protection can be applied on every mobile device with an Android operating system and does not need the root access.

Key words: Android, Dynamic Permission Mechanism, Permissions, Protection, Security

1. Introduction

We are the witnesses of the digital era when almost everyone has at least a one mobile device which uses everyday. These devices are used for different purposes. We can dive these purposes to the three basic categories: Business, Fun & Pleasure and Routine activities. Each of this category has its candidate in a real life. For someone this division can be strict, but for another one it can be the one group of activities on the one device. Let's have an example of the person who uses its mobile device at work, at home and sometimes plays game on this device. This person usually needs to split the device data into his personal data and corporate private data. There is possible of confidential information leakage during his home/pleasure activities through application which could send the information outside the device. And exactly for this kind of people we have implemented a solution which can control the data flow of each application and can dynamically change the permission [4] of the application. If the user works with public or personal data which has no restriction then the application has its behavior unchanged, however on the other hand when the user works with a confidential or private data the application dynamically changes its permission model and the user cannot have the leakage of the information from the device. This implementation is the main content of this paper. At first we are going to introduce the permission mechanism and how it works on the last version of Android platform. Then we compare a few related work and afterwards we will introduce the implementation of dynamic permission mechanism on Android. This implementation has been tested on a lot of different applications and the main results are described below.

2. Permission Mechanism Overview

Applications use a lot of permissions including those needed to access the SD card [13], use the Internet and so on. Google does a great job of displaying what permissions an application uses before installing, but does not go into full detail about what exactly those permissions do.

The sky is the limit when comes to ideas and what is possible with the Android application. Knowing this Google added permissions that need to be defined before the app is allowed to do certain things like access

the Internet for example. This stops apps from doing whatever they want without the user's knowledge. When installing an app from the Google Play Store, a dialog box pops up and the user has to agree that the app can do certain things. Permissions [4] are handled by the Android API framework in the system process. This calls the permission validation mechanism to check that the application has the permission to do what it is trying to do. Some permissions, like *Bluetooth* [11] are handled by the Linux kernel [1]. All of the API [10] calling happens automatically so the developer does not have to worry about making sure the permissions are handled correctly. The permissions are usually selected into few groups, but we can categorize them into three main groups (See Fig. 1). These categories are related to the information that the application works with.

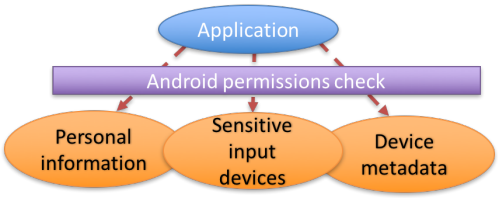

Fig 1. Android Permission Check

2.1. Permission groups

Google provides 17 different permissions developers can use in their apps. Everyone of the list bellow can be part of the main three groups defined earlier. In this article only chosen ones which are related to the topic of the paper are described.

- Device and app history – An application may be able to do read sensitive log data, read web bookmarks and history, retrieve applications and retrieve system internal state. All of these permissions deal with device and app history.
- Cellular data settings – The application can control the device's mobile network settings and possibly intercept the data received.
- Identity – With this permission the application can find accounts on the device, see and modify the owner's contact card and add or remove contacts from the device.
- Contacts – This permission is similar to last one, however, this one can only access contacts on the device, however, it still has the ability to read and modify them.
- Calendar – This permission allows the application to access the user's calendar and events, even if they include confidential information. This also includes the application being able to make events and send emails to guests without the user's knowledge.
- SMS – The application can use the device's SMS and MMS services. This includes the ability to receive text messages (SMS, WAP), read SMS and MMS, edit SMS and MMS, and send SMS and MMS. This can cost money.
- Phone – The application can directly call phone numbers. Read and write the call log, reroute outgoing calls, modify phone state and make calls without the user knowing. Like the SMS permission, this can cost money. This permission group handles everything that has to do with telephony features.
- Photos/Media/Files – The application has the ability to use the file on the device with the application installed. This includes reading and writing to the SD card and USB storage. The application can also mount and un-mount external storage as well as format external storage. This permission deals with reading external storage on newer devices.

- Camera – The application has the ability to take pictures and video. This may occur with or without the user's permission or knowledge.
- Microphone – The application can use the device's microphone, this may include recording audio. This may happen with or without the user's permission.
- Wi-Fi connection information – The application can access the device's Wi-Fi connection information including whether or not Wi-Fi is turned off as well as connected devices.
- Bluetooth connection information – The application can control Bluetooth which includes sending and receiving data from nearby devices.
- Device ID & call information – An application can access the device ID, phone number, whether the device is making a call, and the number connected on the other end of the call.

Every group of the permission can be split into *sink* or *source* permission [5]. This means that the permission is the source of data or the sink. The sink in this content means that the data is sent there or saved to specific place such as internet, SD card or send to another device through any available connection. For the implementation of the dynamic permission mechanism is very important to follow all *sink* permissions, because this can lead to data leakage.

2.2. Permission Enforcement

Classical permission mechanism works in three phases (See Fig. 2). The first phase is during the installation of the application. The user has to agree with all permissions that the application requires, otherwise the installation failed.

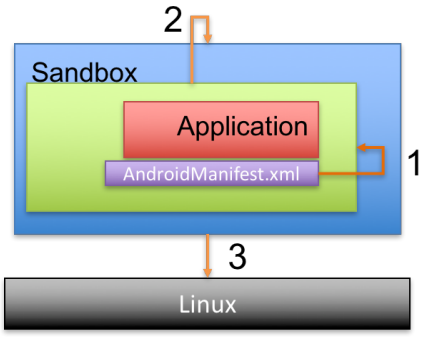

Fig 2. Android Permission Enforcement

During the installation there is checking mechanism that provide mapping permission rules from *AndroidManifest.xml* [10] (file which is part of the application and define requested permissions) to the system permissions. This step is identified as number 1 in the Fig. 2. The second phase is during the opening the application. When a user run the application the application is starting to the running process (See Fig. 2 number 2). In that moment the permissions are mapped to the Linux groups. The last phase of permission enforcement is during the request of the application to provide any activity which needs the specific permission. During this activity the permission marked as Linux group now is checked and executed by the Linux kernel [1]. This last phase is identified as number 3 in Fig. 2.

2.3. Permission mechanism on Android Marshmallow

Android 6.0 Marshmallow [2] is the current last version of Android platform. Marshmallow brought large changes to the permissions model. It introduced the concept of runtime permissions. These are permissions that are requested while the app is running (instead of before the app is installed). These permissions can then be allowed or denied by the user. For approved permissions, these can also be revoked at a later time.

This means there are a couple more things to consider when working with permissions for a Marshmallow application. When the user does not permit the specific permission and the application tries to use it the pop-up screen appears and ask the user for allowing the current permission. As was said this behavior is available in applications which where developed for at least SDK [10] version 23. For older applications there is backwards compatibility mode and this mode has two scenarios:

1. The application is targeting an API less than Android 6.0 Marshmallow (*TargetSdkVersion* [10] < 23), but the emulator/device is Android 6.0 Marshmallow:
    - The application will continue to use the old permission model.
    - All permissions listed in the *AndroidManifest.xml* will be asked for at install time.
    - Users will be able to revoke permissions after the app is installed. It is important to test this scenario since the results of certain actions without the appropriate permission can be unexpected.
2. The emulator/device is running something older than Marshmallow, but the application targets Marshmallow (*TargetSdkVersion* >= 23):
    - The application will continue to use the old permissions model.
    - All permissions listed in the *AndroidManifest.xml* will be asked for at install time.

3. Related Work

During the evolution of the Android operating system a lot of additional features were defined. There is an enormous list of updates, recommendations by community or novel application approaches to upgrade this system. In this section is a small group of the extension to the Android which is related to the topic of security. Diverse threats are identified from mobile services, including permission bypass [9], [10]. The following work is more related to implementing improvements of Android operating system to cover more security principles than its basic version.

First of all there is the project Aurasium [3] which is the cornerstone of this proposal. Aurasium introduces the solution that bypasses the need to root the device when modification of the Android OS is required. The behavior of the application can be modified or the flow of the information can be followed. This project automatically repackages arbitrary applications to keep the sandboxing mechanism and policy enforcement code, which closely watches the behavior of security and privacy intrusions such as attempts to retrieve a users sensitive information, etc. Aurasium has the ability to detect and prevent cases of privilege escalation attacks. Experiments show that these principles can apply this solution to a larger scale of malicious applications with a near 100 percent success rate, without significant performance and space overhead.

Dr. Android and Mr. Hide [8] Fine-grained security policies on unmodified Android is another approach for checking an Internet connection. They presented a concept for replacing coarse Android platform permissions with finer-grained permissions that lower needed privilege levels, decreasing the potential threat from malicious apps. The system contains two novel parts: Mr. Hide and Dr. Android. Mr. Hide is an Android service that

4. Dynamic Permission Mechanism

Dynamic permission mechanism deals with files which are located on the mobile device. These files can be split into two domains: Public and Private. Public domain represents classical user's files such as photos, documents, music or movies. Private domain is the same data as public with additional rule, this data is usually classified or private for reason. This data should be hidden for the others than the user of the device. There exists a lot of solution of protection data from the operating system and also another modification of Android platform. We have different approach of protecting data. We do not modify the operating system we do not even need root access to the system to preserve the original protection of the operating system. We

provide additional layer of protection for existing application without modification of the source code.

4.1. Concept

We have defined the concept of dynamic permission mechanism [7] and the implementation is the proof of concept which is the main part of this paper. On the device are only finite subset of possible leakage of the data. The leakage is usually done through one of the *sink* permission rule [5]. The application should be restricted according to its input. This means that the application should dynamically remove all these *sink* permissions when the application works with a private file. This behavior can be seen on the Fig 3.

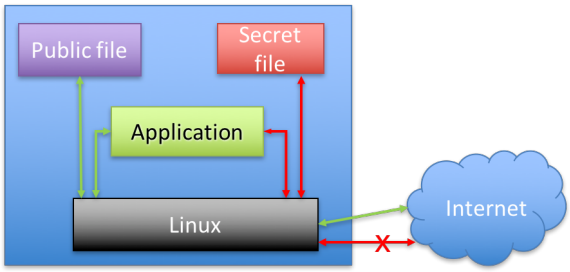

Fig 3 Dynamic Permission Mechanism Concept

The application can open public file through system call *fopen* [1] and can work as was designed, for example send this file to someone else (share it with someone through internet). This activity is identified as green line of data flow on the Fig 3. There is the same flow with a private file, but there should be dynamically removed the ability to use an internet or other type of the *sink* permission. This behavior is identified as red line on the Fig. 3. This is the basic principle of the protection. During the opening of the file we have to identify if the file is public or private. This lead to the classifier which we do not implement but we have two folders on the SD card with public and private folders. We then follow the file path if it is public or private and then we can control the permission mechanism.

The mechanism of checking if the file is public or private is done through checking few low-level system calls such as *fopen* and others. The ability to track these low level system calls has been done with Aurasium project [3]. Aurasium project has two main cores. The first core un-pack the *APK* (installation package)[6] of the application and add a hooks for the system calls. This hooks can be tracked and modified during the application running status. The second core define the flow system of the hooks and also packaging the application into *APK* with signing process.

The permission mechanism can restrict the rules for leakage data through calling the Linux *ioctl* function. The *ioctl* [1] function is on the Android system responsible for permission requested by the application.

4.2. Implementation

The implementation has been split into two main parts. First part is modification of the *APK* package which can provide the ability to add specific *hooks* for Linux kernel system calls. This modification brings the ability tracking all these calls and also modify the responses. This part is the negligible modification of the Aurasium project and is not described in this paper.

The second part is the Android service (or daemon thread) [11] which is tracking all calling from the previously added hooks into applications. We can call this part as the arbiter of the calling the Linux kernel from the modified application. Arbiter decides if the application has the original behavior (classical mode) or restricted mode when the private file is opened. This status the arbiter has during the whole lifecycle of the application and can persists even when the application is terminated.

The implementation of this dynamic permission mechanism (see Fig. 4) is inside process boundary (running application) and communicate with the Native framework API [14] based on C/C++ language. We communicate with specific libraries in this process boundary: *libdvm.so, libandroid.runtime.so, libbinder.so*. On the lower level we communicate with libraries which are responsible to direct calling the Linux kernel of the Android platform (see Fig. 4). These low level libraries are *libso.so, libstdc++.so, libs.so*. All libraries are described in more detail in [1].

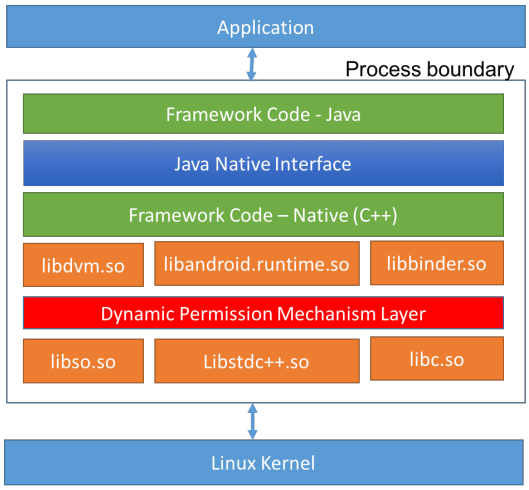

Fig 4 Dynamic Permission Mechanism Stack

5. Experiments

The implementation of the proposal has been done and tested on Android 6 Marshmallow. We have chosen this version because it has the ability to change permission rules separately as was discussed earlier. We had randomly downloaded applications from Google Play market which worked with user data. Usually these applications were document editors, image viewers, music players or video players. We have prepared data on the SD card of the formats: txt, doc, xls, png, jpg, avi and mp3. These testing files have been separated into two folders: public and private.

All applications which were developed for Android version 6 have worked properly and the mechanism permissions were successfully dynamically changed. Older applications have issues with dynamic permission mechanism related to its restrictions. When the application opened the private file and dynamic permissions has been set everything worked. But when the application developed for older *targetSdkVersion* tried to use the permission which was previously restricted Exception was shown and the application was terminated.

6. Conclusion and Future Work

The security issues in the mobile world has become more and more sophisticated and their detection is more complicated. The Main responsibility is on the user, even if he uses the confidential data. The user is the weakest point in the whole string of the security chain. The proposal of protecting application in order to change its permission ability move the responsibility from user to the system. This approach could save the user (or corporation) data and should deny the leakage of confidential passwords and other type of sensitive information from the device. According to experiments the dynamic permission mechanism works, but there are still limitations to the last version of Android operating system. The future work should implement the save removal of permission rules such as return fake or empty data. The permission mechanism currently works properly when the data is separated into two folders. This restriction should be replaced by automatic file classifier which can decide if the file belongs to the public or private folder.

7. Acknowledgements

This work has been supported by the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070), by the project CEZ MSM0021630528 Security-Oriented Research in Information Technology and by project FIT- S-11-1 Advanced Secured, Reliable and Adaptive IT.

References

[1] Love, R., Are, S. H. W., Linus, A. C., Kernels, L. V. C. U., & Begin, B. W. (2005). Linux kernel development second edition (pp. 124-130). Pearson Education, USA.

[2] Allen, G. (2015). Android Security and Permissions. In Beginning Android (pp. 343-354). Apress.

[3] Xu, R., Saidi, H., & Anderson, R. (2012). Aurasium: Practical policy enforcement for android applications. In Presented as part of the 21st USENIX Security Symposium (USENIX Security 12) (pp. 539-552).

[4] Felt, A. P., Chin, E., Hanna, S., Song, D., & Wagner, D. (2011, October). Android permissions demystified. In Proceedings of the 18th ACM conference on Computer and communications security (pp. 627-638). ACM.

[5] Gibler, C., Crussell, J., Erickson, J., & Chen, H. (2012). AndroidLeaks: automatically detecting potential privacy leaks in android applications on a large scale (pp. 291-307). Springer Berlin Heidelberg.

[6] Shabtai, A., Fledel, Y., Kanonov, U., Elovici, Y., Dolev, S., & Glezer, C. (2010). Google android: A comprehensive security assessment. *IEEE Security & Privacy*, (2), 35-44.

[7] Aron, L., & Hanacek, P. (2016, February). A concept of dynamic permission mechanism on android. In *Progress in Applied Mathematics in Science and Engineering Proceedings* (Vol. 1705, p. 020022). AIP Publishing.

[8] Jeon, J., Micinski, K. K., Vaughan, J. A., Fogel, A., Reddy, N., Foster, J. S., & Millstein, T. (2012, October). Dr. Android and Mr. Hide: fine-grained permissions in android applications. In *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices* (pp. 3-14). ACM.

[9] Ongtang, M., McLaughlin, S., Enck, W., & McDaniel, P. (2012). Semantically rich application-centric security in Android. *Security and Communication Networks*, *5*(6), 658-673.

[10] Wu, L., Du, X., & Zhang, H. (2015, February). An effective access control scheme for preventing permission leak in Android. In *Computing, Networking and Communications (ICNC), 2015 International Conference on* (pp. 57-61). IEEE

[11] Meier, R. (2012). Professional Android 4 application development. John Wiley & Sons.

[12] Specification, B. (2001). Version 1.1. Includes: IMS Learning Resource Meta-data Information Model IMS Learning Resource Meta-data XML Binding Specification IMS Learning Resource Meta-data Best Practice and Implementation Guide Available at: www. imsproject. org.

[13] Park, Y., Park, J., Kim, Y., & Jun, S. (2005). U.S. Patent Application No. 11/098,462.

[14] Kim, Y. J., Cho, S. J., Kim, K. J., Hwang, E. H., Yoon, S. H., & Jeon, J. W. (2012, October). Benchmarking Java application using JNI and native C application on Android. In *Control, Automation and Systems (ICCAS), 2012 12th International Conference on* (pp. 284-288). IEEE.