Acta Informatica Pragensia, 2015, 4(3): 310–317
DOI: 10.18267/j.aip.77
Peer-reviewed paper

Android Access Control Extension

Anton Baláž[*], Branislav Madoš[*], Michal Ambróz[*]

Abstract

The main objective of this work is to analyze and extend security model of mobile devices running on Android OS. Provided security extension is a Linux kernel security module that allows the system administrator to restrict program's capabilities with per-program profiles. Profiles can allow capabilities like network access, raw socket access, and the permission to read, write, or execute files on matching paths. Module supplements the traditional Android capability access control model by providing mandatory access control (MAC) based on path. This extension increases security of access to system objects in a device and allows creating security sandboxes per application.

Keywords: Android, Security, Sandbox, Policy, Profile, Access control, MAC.

1 Introduction

Android represents an operating system for mobile devices being used with approx. 80% of all these devices. Since approx. 97% of all malware is created for this operating system, there is a need to pay a proper attention to security of such devices (Bousquet, 2013). Malware created for mobile devices differs in various goals, e.g. obtaining personal data, SIM card number or IMEI device number which is send to servers of third parties and misused, or creating hidden calls and SMS messages which can cost a lot of money (Novák, 2012).

Security of mobile devices is extensively affected by user behavior, as every potentially dangerous application requires permissions when being installed. Malicious software usually requires inadequate set of permissions according to its purpose. If users paid a proper attention to these permissions, the risk of threats to their devices would be minimized. However, according to many studies, only around 20% of users pay attention to permissions when installing applications to their mobile devices (Barrera, 2010).

The main goal of this work is to propose and implement an architecture which creates a mandatory access control module for kernel of Android operating system which allows managing installed applications permissions as well as the access control of system resources more securely (Smalley, 2013).

2 State of Art

As an operating system for mobile devices, Android is based on Linux kernel inheriting both advantages and disadvantages of this architecture. Linux kernel provides a number of security

[*] Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics,
Technical University of Košice, Letná 9, 042 00 Košice, Slovak Republic
✉ anton.balaz@tuke.sk, branislav.mados@tuke.sk, michal.ambroz@tuke.sk

benefits for Android. These advantages include isolation of active processes, interprocess communication or user security model etc. (Hoopes, 2009). With such a multiuser system, an early goal is to separate resources of one user from another. However, for Android it means that resources of one application are separated from resources of another application, since Android applications are defined by a given user identification number from their installation to their uninstallation. As other systems, yet Linux and Android are still prone to security faults (Wei, 2012). In case of mobile devices, these faults mainly include threats to privacy and sensitivity of user data, possibility of identity theft and possibility of money leak, if permissions related to phone calls and SMS messages are misused (Shan, 2012).

Applications for Android can be distributed to users through developer web pages, though more common way is to use Google Play, a shop with Google applications. Google enters the process of publication of applications in their shop only at minimal level, making the work of developers easier on one hand, but creating a space for spreading insecure applications among users on the other hand. After its installation, such an application can cause harm to the device, eventually it can cause an unexpected behavior of the device (Wagner, 2012).

In order to avoid such an unexpected behavior, every application contains a set of permissions which should be defined by its user. Android provides several mechanisms limiting the interactions between the system and the applications and between the applications themselves. To handle applications privileges, Android uses a specific model of permissions (Wu, 2015). Each application requests a set of permissions, allowing it to perform specific actions. For instance, an application that needs to send SMS has to request the SEND SMS permission. This is a security model based on capabilities (Vargas, 2012). Permissions are explicitly granted by the user during the installation of the application. Nevertheless, since Android does not allow a partial selection, the user must either accept all the permissions or cancel the installation. Moreover, the user cannot change the permissions afterwards: the only way is to uninstall the application. A solution proposed in this paper allows the user to specify exactly what resources an application can use.

Actually, there exist a number of security solutions for Android:

- TaintDroid is an extension of Android that enables to track information flows on Android smartphones (Felt, 2012). TaintDroid uses data tainting to track sensible data. It assumes that the applications installed by the user cannot be trusted. It monitors the user's data and aims to detect whether any data leave the system.
- AppFence makes privacy controls on Android applications by retrofitting the runtime environment. AppFence implements two systems: data shadowing, i.e. providing fake data to the application (empty contact list etc.) instead of sensitive data, and ex-filtration blocking, i.e. preventing sensitive data (tainted by TaintDroid) from leaving the device.
- Aurasium is a protection solution that does not alter the Android OS. Indeed, Aurasium hardens Android applications by repackaging them in order to add its policy enforcement code. Thus, Aurasium can control access to sensitive information, such as IMEI number, location etc.
- CRePE presents a policy enforcement solution based on the contextual environment (geographical location, time of the day etc.). These environments are automatically detected by CRePE, and no action from the user is requested. Thus, users can disable particular functions depending on the current situation.

As shown by this state of the art, a solution that provides a fine-grained access control mechanism is required. Module proposed in this work is designed as a sandbox for

applications installed in the devices. The user defines a set of policies (profile) to control the propagation of data. Sandbox is a mechanism which operates on user level (Vokorokos, Baláž & Ádám, 2015). It tracks untrusted applications and allows controlling potentially insecure system calls. As a result of this assumption, an application is capable to perform any action within its address space in more secure way.

3 Designed Security Architecture

In order to solve the presented problem, we proposed and implemented a sandbox module for the OS kernel which allows a user to manage advanced permissions assigned to installed applications (Spreitzenbarth, 2013). This allows to perform an effective creation of isolated sandboxes over running applications and to securely manage an access to system resources (Fig. 1).

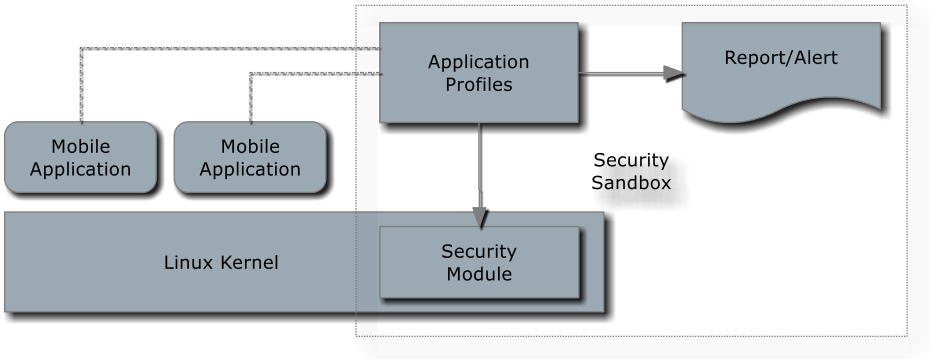

Fig. 1. *Sandbox architecture. Source: Authors.*

The security extension monitors application requests (system calls) for system resources. Each request is referenced to security policy profile, confining data flow from/to application and OS resources.

$$\text{Application} \overset{\text{Request ID}}{\rightarrow} \text{Policy} \overset{\text{Profile}}{\rightarrow} \text{System Resources} \overset{\text{Permit, Deny}}{}$$

Within the design phase, it was necessary to appropriately select an interface allowing to operate with profile files – security policy. Such an interface would apply these profiles and it would manage application permission according to the desired security policy. The most common approach in Linux OS uses a virtual file system called proc or sys, both allowing to create and process files without any impact on the operating system (Danková, 2011). From the previous analysis, the realized solution is based on system call filp open. Module presented in this work does not need to have an access to a superuser account. For the purposes of this work, filp_open interface was used, provided by the OS kernel, allowing to filter all common system operations on files, e.g. open, reading and writing. The main restrictions of the filp_open function include the size which cannot exceed the size of the stack, 1 024 bits in this case. Structure of a profile file is specifically defined. Profile name corresponds to user identification number of the application. Every file line represents one permission adjustment of a given application or an access adjustment to one of the files. Profile files implement black list – entering denying rules, default policy is permit.

Profile example:

```
/sdcard/images/dont/show/picture/pic.png rwf
/sdcadrd/other/dont/show/data/data.txt rp
network
#application
```

Within the presented example, one may notice that the user prohibited to display pic.png in images directory as well as to change this file. Conversely, in data directory, the user only permits to read contents of data.txt, however, the given application is allowed to modify this file. The third line defines the user denied a network access to this application. The last line allows the user to write a comment to which application this profile applies.

Profile syntax:

```
<profile> ::= <permission><operation><mode> | <permission> |
<comment>
<comment> ::= "#" <string>
<permission> ::= <path-to-file>
|"network"|"identity"|"contacts"|"photos, media, files"
<operation> ::= "r"|"w"|"o"|"rw"|"wo"|"or"|"rwo"|""
<mode> ::= "p"|"f"|""
```

In this work, the module design included 2.3.69 Goldfish version of Android. In order to ensure a secure functioning of the module, the first step focused on the necessity to determine address of a table with system calls located in system.map in a directory containing system kernel. The module hooks an access to all system calls performed to installed applications through the table of system calls.

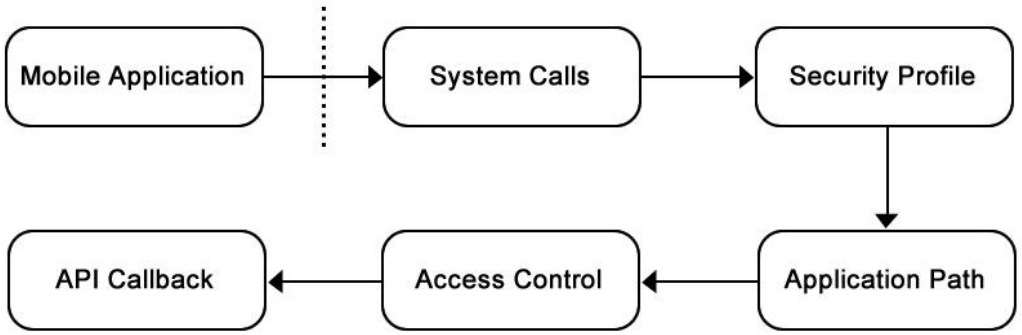

Fig. 2. *Module operation. Source: Authors.*

The module contains two obligatory functions: init_module and cleanup_module. The aim of the init_module function is to initialize the module and further it is used to hook system calls. The cleanup_module function is called when the module ends its work and it assigns original values to every call. However, in case of this work, the cleanup_module function executes on the functionality of hooking the Internet service, since the values are assigned to the system calls when they are captured. Fig. 2 depicts a simplified model of security module processing. The module operates in several steps:

1. In the beginning the module should be introduced to kernel using the insmod command.

2. While the module operates, it is necessary to hook system calls and filter them according to the defined profiles.
3. If a packet is captured by the module, it loads the profile. If the application which initialized the packet already includes a defined profile as well as a defined size of the data to be transmitted, it is necessary to count how many data have been already transmitted by the application out of the data guaranteed by the user. If the application received more data than permitted, then the packet is discarded. Otherwise the packet is received and the evaluation continues.
4. If the module captures a system call to open, read or write to a file, it loads a profile for the application which invoked the call.
5. The system calls of read and write to file do not contain the name of the file to which they approach but they contain its descriptor. Since it is possible to determine a path to the file from this number, this step is recommended in such a case.
6. After the module receives a path to the file and reads the file, it is possible to evaluate permissions from the profile.
7. Since some permissions contain their own identification number, they are monitored separately according to files to which the application approaches.
8. If permission located in the profile matches with the file to which the application approaches, the call is returned with the -EPERM macro of the kernel, marking that the permission was removed from the application. Otherwise the original system call is returned.

4 Module Evaluation

Testing of the proposed security policy was performed on the Android OS using the 2.3.69 Goldfish kernel. For evaluation purposes of the enhanced security policy and comparison to the status quo, we have chosen several applications which are commonly installed by users and ones containing malware, e.g. Flappy Bird with a malware secretly sending SMS messages, applications of iMatch and iCalendar containing the Zsone malware, and Tencent, an application which cumulates and sends data to remote servers. In order to test file security, another ES File Manager was selected as well, simulating opening of sensitive files. As an example, this paper selected an application containing several suspicious permissions as it is displayed in Fig. 3.

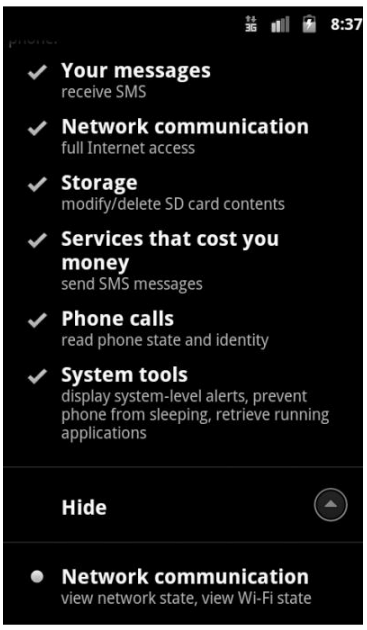

Fig. 3. *FlappyBird permissions. Source: Authors.*

The application requires permissions to send SMS messages, causing a financial harm to the user. Based on this information, a profile was created and as it is visible in Fig. 3, the module successfully prevented the application from its insecure malicious behavior. Similarly were tested all selected applications against defined security policy by application profiles.

Fig. 4. *Call/SMS denying. Source: Authors.*

Tab. 1 contains an evaluation of the selected set of tested insecure applications. Lines labeled by X mark successful attack prevention. The - character indicates the permission was not dangerous to the application and N marks such a behavior that was impossible to reproduce.

	FlappyBird	iMatch	iCalendar	Tencent	ES
Internet Access	-	X	X	X	-
SMS and Calls	X	X	X	-	-
Identity	-	-	-	N	-
File Access	-	-	-	-	X

Tab. 1. *Security module evaluation. Source: Authors.*

5 Conclusion

Android represents an operating system for mobile devices being used with majority of all these devices. Since approx. 97% of all malware is created for this operating system, there is a need to pay a proper attention to security of such devices.

The aim of this work was to propose and implement a security enhancement to Android OS through the module for the operating system kernel. Based on a profile this module is capable of restricting threats to applications installed in mobile devices. Designed extension is an application security tool designed to provide an easy-to-use security framework for installed applications. Module proactively protects the operating system and applications from external or internal threats, even zero-day attacks, by enforcing expected behavior and preventing even unknown application flaws from being exploited. Module security policies, called profiles, completely define what system resources individual applications can access, and with what privileges.

Proposed security extension was evaluated on selected samples of malicious code, which causes several malicious activities such as unwanted SMS delivering, privacy data gathering, unsolicited data transmitting. From the results, active proposed module successfully eliminates mentioned security threats of the evaluated samples. However, the proposed solution contains actually several restrictions, e.g. no customized GUI interface to create and modify profiles as well the need to compile the module for a specific version of the system kernel.

Acknowledge

This work was supported by the Slovak Research and Development Agency under the contract No. APVV-0008-10 and project KEGA 077TUKE-4/2015 Promoting the interconnection of Computer and Software Engineering using the KPIkit.

References

Barrera, D., Kayacik, H. G., van Oorschot, P. C., & Somayaji, A. (2010). 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* (pp. 73-84). New York: ACM. doi: 10.1145/1866307.1866317

Bousquet, A., Briffaut, J., Clévy, L., Toinard, C., & Venelle, B. (2013). Mandatory Access Control for the Android Dalvik Virtual Machine. *ESOS: Workshop on Embedded Self-Organizing Systems*. Retrieved from https://www.usenix.org/conference/esos13/workshop-program/presentation/bousquet

Danková, E., Ádám, N. & Jakubčo, P. (2011). An anomaly-based intrusion detection system. In *Proceeding of the Electrical Engineering and Informatics II* (pp. 260-264). Košice: FEI TU.

Hoopes, J. (2009). *Virtualization for security: including sandboxing, disaster recovery, high availability, forensic analysis, and honeypotting.* New York: Elsevier.

Novák, D., Ádám, N. (2012). Route planner for mobile devices. In Kollár, J. (ed.) *Computer Science and Technology Research Survey*. Košice: FEI TU.

Felt, A. P., Ha, E., Egelman, S., Haney, A., Chin, E., & Wagner, D. (2012). Android permissions: User attention, comprehension, and behavior. In *Proceedings of the Eighth Symposium on Usable Privacy and Security* (no. 3). New York: ACM. doi: 10.1145/2335356.2335360

Shan, Z., Wang, X., Chiueh, T. C., & Meng, X. (2012). Facilitating inter-application interactions for os-level virtualization. In *Proceedings of the 8th ACM SIGPLAN/SIGOPS conference on Virtual Execution Environments* (pp. 75-86). New York: ACM. doi: 10.1145/2151024.2151036

Smalley, S., & Craig, R. (2013). *Security Enhanced (SE) Android: Bringing Flexible MAC to Android*. Retrieved from http://www.internetsociety.org/doc/security-enhanced-se-android-bringing-flexible-mac-android

Spreitzenbarth, M., Freiling, F., Echtler, F., Schreck, T., & Hoffmann, J. (2013). Mobile-sandbox: having a deeper look into android applications. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing* (pp. 1808-1815). New York: ACM. doi: 10.1145/2480362.2480701

Vargas, R. J. G., Huerta, R. G., Anaya, E. A., & Hernandez, A. F. M. (2012). Security controls for Android. In *Proceedings of the 4th International Conference on Computational Aspects of Social Networks* (pp. 212-216). New York: IEEE. doi: 10.1109/CASoN.2012.6412404

Vokorokos, L., Baláž, A., & Ádám, N. (2015). Secure Web Server System Resources Utilization. *Acta Polytechnica Hungarica*, 12(2), 5-19. doi: 10.12700/APH.12.2.2015.2.1

Wagner, D., Goldberg, I., & Thomas, R. (1996). A secure environment for untrusted helper applications. In *Proceedings of the 6th conference on USENIX Security Symposium, Focusing on Applications of Cryptography*. Berkeley: USENIX Association

Wei, X., Gomez, L., Neamtiu, I., & Faloutsos, M. (2012). Permission evolution in the android ecosystem. In *Proceedings of the 28th Annual Computer Security Applications Conference* (pp. 31-40). New York: ACM. doi: 10.1145/2420950.2420956

Wu, L., Du, X., & Zhang, H. (2015). An effective access control scheme for preventing permission leak in Android. In *Proceedings of the International Conference on Computing, Networking and Communications* (pp. 57-61). IEEE. doi: 10.1109/ICCNC.2015.7069315