# Enforcing Multiple Security Policies for Android System*

Tao Guo
China Information Technology Security Evaluation Center
Beijing, China
guotao@itsec.gov.cn

Puhan Zhang
China Information Technology Security Evaluation Center
Beijing, China
zhangph2008@gmail.com

Hongliang Liang
Beijing University of Posts and Telecommunications
Beijing, China
hliang@bupt.edu.cn

Shuai Shao
China Information Technology Security Evaluation Center
Beijing, China
shaoshuaib@163.com

***Abstract*—The popularity of Android makes it the prime target of the latest surge in mobile malware. Protecting privacy and integrity of information is helpful for Android users. Currently, malicious software often achieve the purpose of privacy theft and malicious chargeback by sending short messages, making phone calls or connecting Internet surreptitiously. We develop a novel solution that supports multiple security policies to provide much of the integrity and privacy that users desire. We present and implement a security framework for Android which consists of both mandatory access control in the kernel layer and role-based access control in the framework layer. It allows users to define their own security policy and provides fine-grained access control to (untrusted) applications. We implemented a prototype system MPdroid for Android 4.0 platform. Experiments show that we can apply this solution to really help users control applications, block malicious software without significant performance overhead.**

***Keywords-security policy; access control; permission; Android***

## I. INTRODUCTION

As an open source operating system and associated software stack for smartphones, Google's Android gains increasing popularity recent years. Mcafee's report[1] showed that Android is being targeted by hackers more than any other platform. Android accounts for nearly all mobile malware and more than 14,000 threats have been discovered in the first three months of 2013 alone. Mobile malware is expected to increase in 2013, with some calling it "the year of mobile malware" for Android users. Targeted marwares that steal personal information and make malicious chargeback made up a majority of these attacks[2,3].

Android security depends heavily on discretionary access control(DAC) protection for Linux file system and Java APIs permissions check in Android framework layer. DAC can be easily compromised by malwares. Android uses its permission model to protect sensitive resources and functions. However, it has the following shortcomings: there is no way of granting some permissions and denying others[4]; the permission assignment can only happen during the installation of applications; the permissions cannot be changed or restricted after installation. Moreover, malwares can exploit the vulnerabilities of Android system or call Linux APIs to bypass Android's permissions checking. To address these problems, we propose an security framework for Android which consists of both mandatory access control (MAC) in the kernel layer and role-based access control (RBAC[5]) in the framework layer. It allows users to define their own security policy and provides fine-grained access control to (untrusted) applications. MAC mechanism allows administrators enforcing fine-grained access control to confine applications or process to a tight environment in which they can perform only specific actions according the security policy. Thus, untrusted applications are limited and cannot damage the system. Role-based policies can help users to regulate applications' access to the resources on the basis of the activities that applications execute in the system. A role can be defined as a set of actions and responsibilities associated with a particular activity. Then, instead of specifying the permissions requested by an application, the applications on Android platform are given authorization in terms of roles. Furthermore, RBAC allows for a more comprehensive way of permitting or blocking security relevant actions, because applications are now associated with roles defined by users. Thus, RBAC can allow users to confine applications with the least privilege and separations of duties principles.

Our main contributions are that

(1) A novel solution that supports multiple security policies for Android platform to provide much of the integrity and privacy that users desire is proposed.

(2) We implemented a prototype system MPdroid for Android 4.0 platform. Its effectiveness and performance is tested with real world applications. We shows that privacy theft and malicious chargeback is mitigated by our solution and MPdroid. Our research results can be useful for other related work in Android security area.

The paper is organized as follows. Section 2 describes the security mechanism in Android. Our solution to enforce security policies in Android is given in Section 3. Section 4 describes the implementation of MPdroid system. The features and performance evaluation are discussed in section

5. Related works are discussed in Section 6. We conclude in Section 7.

## II. SECURITY MECHANISM IN ANDROID

Android system uses two levels security mechanisms -- Linux DAC and Android permissions checking. In Android kernel layer, each file is associated with an owner user and group IDs and three tuples of read, write and execute. The kernel enforces the first tuple on the owner, the second on users belonging to the group, and the third on the other users. Generally speaking, an application can not access files created by other applications. In Android framework layer, permission checking for Java APIs is the main security mechanism. Android users can install third-part applications through the Android Market or other application stores, Android treats all applica-tions as potentially buggy or malicious, No application should have default permissions to affect either other applications or the underlying operating system. Each application runs in a single process with a low-privilege Linux user ID, and only can access its own files by default according to the DAC in Android kernel. If an application needs to access sensitive resource, it has to request the corresponding permissions specified in the Androidmanifest.xml file at install-time. To enforce permissions, different parts of Android system components or services invoke the permission checking mechanism to verify whether a given application has a valid permission. Android permissions checking is placed in the API implementation of the system components, not in applications them-selves.

These above security mechanisms are insufficient and coarse-grained to defend increasing kinds of security threats. Linux Security Modules (LSM[6]) is an effective security solution for Linux system. LSM had been developed as lightweight and general-purpose access control framework for Linux kernel. It supports various access control models which are implemented as loadable kernel modules, including SELinux[7], Smack[8] and so on. LSM framework and certain a mandatory access control will be helpful if they are introduced in Android..

## III. SECURITY FRAMEWORK ENFORCING MULTIPLE POLICIES FOR ANDROID

We propose an security framework for Android system, which supports multiple security policies and provides mobile phone users fine-grained access control[9] on the level of Android applications. The security framework is shown in figure 1. In the application layer, applications process are running and requesting some permissions, RBAC policy tool is used by uses to define and manage the policies. In Android framework layer, for each permission request form applications, RBAC engine makes access decision according to the information of RBAC database. In Android runtime, init process and zygote process will read/write them into RBAC database and load the MAC policy rules into memory.

We implement the framework by leveraging Smack in the Android kernel and implementing RBAC in the Android framework layer. The role-based policy management tool is placed on the Android application layer through which mobile phone users can create/edit/assign/delete roles for specific applications, in addition, this tool can also generate automatically smack rules in terms of roles. For example, when a user defines “contact” role which have the permissions of making phone calls, sending text messages or accessing contact files, the smack rule will be formed that permit the application process to communicate with the radio process and contact process. RBAC database contains roles information, associated permissions information and smack rules for specified applications, which was developed as SQLite database. RBAC engine decides if an application can access a resource according to the RBAC database. Linux kernel with Smack uses LSM infrastructure to attach labels to kernel data structures, including tasks, inode and so on. Smack Labels are stored as extended attributes (xattrs) on files. The only operation that is carried out on the labels is comparison for equality. A task can access an object only if their labels match or there are explicit smack rules to allow it. For example, a process A wants to send a packet to another process B, if the smack labels of the two process are not the same, then, a smack rule must be designed that the smack label of A process have “write” permission to the smack label of B process. A smack rule consists of a subject label, an object label and the access mode desired, this triple is written to “/smack/load” file, which installs the rule in Linux kernel. The smack rule is executed by smack kernel module which can control the behaviors of Linux process. When Android system runs, the *init* process or z*ygote* process will read the RBAC database and load/write smack rules.

The idea underlying our security framework is to enforce the fine-grained access policy after dynamic permission checking. We implement the idea by hooking the permission checking function. Permission checking occurs in “checkUidPermission” method of the “Package Manager Service”. Hence, we insert a hook in “checkUidPermission” method, When an application invokes a system service or compon-ent, Android checks the access permission with the method “checkUidPermission” of the class Package ManagerService. After the Android permission checking, From RBAC database, our framework locate the permissions set “permset” with the uid and judge whether the permissions requested by the appli-cation with such uid can be granted. By introducing different roles, important permissions such as “Android.permi-ssion.CALL_PHONE”, “Android.permission.INTERNET”, can be restricted to some specific roles. For example, if a user wish to install a game application, meanwhile he does not want the game application access network or send SMS messages, firstly, he can create a role (for example named as “game”) without the permissions of SEND_SMS or INTERNET, then he can assign the “game” role to the game application, thus, anytime the behavior of sending SMS or accessing internet from the game application will be denied by the access control framework.

## IV. IMPLEMENTATION OF MPDROID

When implementing MPdroid, we built the modules shown in figure1, improved the smack LSM module, and created some specific smack rules for Android system. We encountered several challenges as follows.

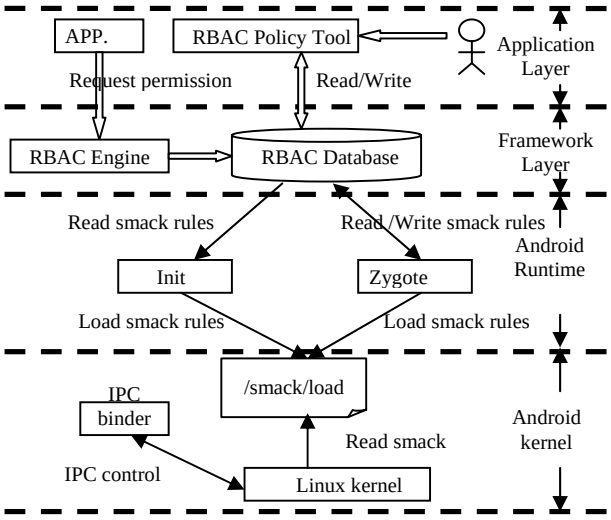


Fig. 1. The architecture of access control system for Android

1. Android does not support smack: we need to reconfig and recompile the Android4.0 goldfish kernel, and let goldfish kernel support ext4 file system.

2. Android bionic library does not support extended attributes: In order to label smack labels for Linux files and tasks, we need to use the Linux system call--"setxattr", "getxattr". We modified bionic and regenerate these system calls.

3. Android cannot load smack policy: The straightforward solution is to develop an external tool kit to load smack rules and label smack labels for tasks and files. we developed a library "libsmack" which provides core functions such as "setsmackrule", "setfilelabel", "settasklabel" and so on. Then, we cross-compiled "libsmack" as dynamic and static link library.

4. It is difficult to create a custom smack policy for Android: Although smack rule is simple, it is not easy to find the subject and object and create smack rules to provide integrity and privacy protection for Android. We observed that many important services such as sending SMS or phone calling are achieved on the basis of binder driver. For example, radio process whose uid is 1001 and which is forked by Zygote exchanges IPC binder data with the radio process forked by Init, in order to access contact, processes have to exchange IPC binder data with contact process whose uid is 10000 and which is forked by Zygote. So we added a hook in smack LSM and modified the binder driver in the kernel.

If two tasks want to exchange IPC binder data in binder driver, they must have "write" permission to each other. Malwares can bypass the permission checking, but they cannot bypass IPC checking. Furthermore, we modified Init code so that Init process will load smack policy early. Meanwhile, we modified Zygote code, when a privileged process is forked by Zygote, it can read or write smack policy from RBAC database and load smack rules into "/smack/load" file.

## V. EVALUATION

To evaluate the effectiveness of our solution and MPdroid prototype system, we selected Baidu contact, Kugou music, DroidDream[10] applications as testing cases. We created "contact" role for Baidu contact, "mediaplayer" role for Kugou music and "malware" role with some specific permissions for DroidDream. Only "contact" role can send SMS, make phone calls and access contact, when roles are created, the smack rules are automatically generated by our system(see fig. 2). The uid of an application is used as either subject label or object label, "sms" is defined as the smack label of SMS database file, "contact" as the smack label of contact database file. In order to make sure that any app can start normally, it should communicate with tasks or files whose smack labels are "_". When we tried to send SMS with Baidu contact, the result showed that SMS sending failed shown as fig.3.

```
_ 1001 rwxa
1001 _ rwxa
_ 10000 rwxa
10000 _ rwxa
1001 sms rwxa
10017 sms rwxa
1001 10017 rwxa
10017 1001 _
10017 _ rwxa
10037 _ rwxa
10037 sms rwxa
10000 10037 rwxa
10037 10000 rwxa
10017 10037 rwxa
10037 10017 rwxa
1001 10037 rwxa
10037 1001 rwxa
10000 contact rwxa
……
```

Fig. 2. smack policy rules

JavaBinder( 615): Failed binder transaction!

Fig. 3. Testing Baidu contact application

Baidu contact process whose uid is 10017 really exchanged IPC binder data with "1001" radio, the hook in the binder driver stopped Baidu contact from sending SMS because of a rule "10017 1001 -". Fig.4 shows, Advanced File Manager and Super Ringtone Marker which are two variants of DroidDream malware cannot access SDcard or Internet.


Fig. 4. Blocking DroidDream malware from SDcard and Internet access

To test the performance of MPdroid, we modified LmBench3 and cross-compiled it to make it run in Android4.0, then we ran LmBench3 independently 100 times to compare the performance overhead without and with our framework on Android system. The comparison results shown in Table 1 show that MPdroid causes little performance loss.

TABLE 1. PERFORMANCE COMPARISON

| **Measure** Item | **Android** | **MPdroid** | **Overhead** |
|---|---|---|---|
| Simple syscall(ms) | 0.2118 | 0.2126 | -0.38% |
| Simple read(ms) | 0.3032 | 0.3033 | -0.03% |
| Simple write(ms) | 0.2627 | 0.2692 | -0.03% |
| Simple stat(ms) | 1.2076 | 1.2170 | -0.78% |
| Simple open/close(ms) | 2.0931 | 2.1253 | -0.02% |
| Single handler installation | 0.4317 | 0.4313 | +0.09% |
| Single handler overhead | 1.0362 | 1.0239 | +1.1% |
| Protection fault(ms) | 0.3620 | 0.3692 | -2% |
| Pipe latency(ms) | 8.6882 | 8.7909 | -1.2% |
| AF_UNIX_sock_stream latency(ms) | 11.1714 | 11.3212 | -1.3% |
| Process fork+exit(ms) | 100.7778 | 100.7925 | 0% |
| Process fork+execve(ms) | 350.1285 | 350.1333 | 0% |

## VI. RELATED WORK

Several security extensions have recently been developed to enhance Android security[11] in different ways. CRePE[12] takes the mobile devices context into account when making security related decision, however its access control ability is not mandatory and can be bypassed. Our security solution does not take context into account, which is considered in our future work. Apex[13] is a policy enforcement framework for Android that allows a user to selectively deny or grant Android permissions for applications at installation time. However, a previously rejected permissions will never be granted again, effectively crippling the application. Our solution allows a more flexible way of grouping permissions into different roles which can be changed depending on the current activity and user. So users can grant or deny permissions for an application in terms of role when the application is running.

Many other security solutions try to prevent the so-called permission re-delegation attack, which let malwares without certain a permission to misuse another application with that permission to act on behalf of the malicious application. IPC inspection[14] could prevent the attack by reducing the overall permission set of a calling and a called application to their intersection of permissions. Our system can also do this by IPC checking occurred in kernel. In essence, the above research work is based on the Android permissions checking, while our system not only give a fine-grained Android permissions checking mechanism, but also improve smack module to control Linux process in Android kernel. Ongtang[15] introduced a security framework called Saint that governs fine-grained access control at run-time by analyzing and restricting the communication channels between applications. Saint's policy does not consider single application and therefore does not provide the type of access control we propose in this paper. SEAndroid[16] is an on-going project to identify and address critical gaps in the security of Android, however, our system enforces the RBAC besides of MAC, is more light-weighted and need not add/modify excessive codes to Android.

## VII. CONCLUSION

We develop a novel solution that supports multiple security policies to provide integrity and privacy for users. We present and implement a security framework for Android which consists of both mandatory access control in the kernel layer and role-based access control in the framework layer. It allows users to define their own security policy and provides fine-grained access control to (untrusted) applications. We implemented a prototype system MPdroid for Android 4.0 platform. Experiments show that we can apply this solution to really help users control applications, block malicious software according their self-defined policy, without significant performance overhead.

## ACKNOWLEDGMENT

The paper is supported by the National Natural Science Foundation of China (61272493), the National High Technology Research and Development Program of China (2012AA012903).

## REFERENCES

[1] Mcafee report, http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q1-2013.pdf.

[2] Yajin Zhou, Xuxian Jiang, "Dissecting android malware: characterization and evolution", IEEE Symposium on Security and Privacy 10.1109/SP. 2012.

[3] J. Cheng, S. H. Wong, H, Yang, "Virus detection and alert for Smartphones", In Proc. of MobiSys'07, pp. 258–271, June 2007.

[4] Android Reference: http://developer.Android.com/guide/ topics

[5] Ravi S.Sandhu, Pierangela Samarati, "Access control: principle and practice", IEEE Communication Magazine 0163-6804/94.

[6] Marshall D. Abrams, Leonard J. LaPadula, Kenneth W. Eggers, Ingrid M. Olson., "A generalized framework for access control, An informal description". Proc. of the 13th National Computer Security Conference, pp135–143, October. 1990.

[7] P. Loscocco and S. Smalley. "Meeting critical security objectives with security-enhanced linux". Proceedings of the 2001 Ottawa Linux Symposium, July 2001.

[8] Casey Schaufler, "Smack in embedded computing", Proceedings of the Linux Symposium Volume Two 2008.

[9] D.F.Ferraiolo, D.M.Gibert, N.Lynch. "An examination of federal and commercial access control policy needs", 6th NIST-NCSC National Computer Security Conference, pp. 107-116, Baltimore, MD, 1993.

[10] http://virus.netqin.com/Android/a.spread.DroidDream .z/.

[11] Enck William, Machgar ongatang, Patrick mcdaniel, “Understanding android security”, IEEE computer society 1540-7993/09 2009.

[12] Mauro Conti, Vu Thien Nga, Nguyen, and Bruno Crispo, “CRePE: context-related policy enforcement for android”. ISC 2010, LNCS 6531, pp. 331–345, 2011.

[13] Mohammad Nauman, Sohail Khan, Xinwen Zhang, “Apex: extending android permission model and enforcement with user-defined runtime constraints”, ACM 978-1-60558-936-7/10/04.

[14] Adrienne Porter Felt, Helen J. Wang, Alexander Moshchuk, “Permission re-delegation: attacks and defenses”, USENIX, 2011.

[15] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel. “Semantically rich application-centric security in Android”. Journal of Security and Communication Network, 2011.

[16] Stephen Smalley and Robert Craig, “Security Enhanced (SE) Android: Bringing Flexible MAC to Android”, 20th Annual Network and Distributed System Security Symposium (NDSS '13), Feb 2013.