Access Control Framework 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

***Abstract*—Protection for confidentiality and integrity of information is a very critical requirement for many mobile device users. Currently, malicious software on the Android platform often achieve the purpose of privacy theft and malicious chargeback by sending short messages, making phone calls or network communication surreptitiously. In this work we present and implement an Access Control Framework for Android which consists of both mandatory access control (MAC) in the kernel layer and role-based access control (RBAC) in the framework layer. It provides fine-grained access check to control the permissions requested by applications. A friendly security policy manage tool is given for users to define role-based policies for managing applications. Finally, the experiment shows that our system can really help users control applications, block malicious software from the behavior of privacy theft and malicious chargeback.**

Keywords—MAC, RBAC, permission, Android

I. INTRODUCTION

Google's Android is an open source operating system and associated software stack for smartphones. A report[1] showed that the malwares[14] on Android platform was killing up to 3523 kinds, the privacy theft[10] and malicious chargeback are the main feature of malwares. Android system depends heavily on DAC protection for Linux file system and Java APIs permissions check in Android framework layer. DAC can be easily compromised by viruses and Trojans. An important part of Android's security framework is its permission model which is used to access sensitive resources and functions. However, it has the following shortcomings: there is no way of granting some permissions[15] and denying others; the permission assignment can only happen during the installation time; 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 permissions checking. To address these problems, we propose an access control framework for Android which consists of both mandatory access control (MAC) in the kernel layer and role-based access control (RBAC[3]) in the framework layer. It provides fi-ne-grained access check to control the permissions requested by applications. MAC mechanism gives administrators low-level, fine-grained access control capabilities to confine applications or process to a tight environment in which they can perform only specific actions in terms of rules. Thus, untrusted applications are limited to safe actions and cannot damage the system. Role-based policies can help mobile phone 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 per-missions 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 specified by users. Thus, RBAC can provide users the capability to confine applications with the least privilege and separations of duties.

We provide the following contributions in this paper: (1) an access control framework for Android platform is proposed and implemented. (2) Its effectiveness and performance is tested with real world applications. We shows that privacy theft and malicious chargeback is mitigated by limiting permissions with RBAC and executing MAC check. 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, Role-based access control in Android is given in Section 3, Section 4 describes the improvement to smack for Android system. Then the performance and function are tested in section 5. Related work are discussed in Section 6. We conclude in Section 7.

II. SECURITY IN ANDROID

There are two levels security mechanisms in Android system--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 An-droid 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 ap-plication needs to access sensitive resource, it has to request the corresponding permissions specified in the manifest file at install-time. To enforce permissions, different parts of

Funds support: the National Natural Science Foundation of China (61272493) ; the National High Technology Research and Development Program of China (2012AA012903)

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.

Android's security mechanisms are insufficient and coarse-grained to defend the security threats. Linux Security Modules (LSM[8]) 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[17], Smack[2] and so on. LSM framework and certain a mandatory access control will be helpful if they are introduced in Android.

III. ACCESS CONTROL FRAMEWORK FOR ANDROID

We propose an access control framework for Android system, which provides mobile phone users fine-grained access control[12] on the level of Android applications. The core components of the framework are designed and built in the modular form, as figure 1 shows. 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 access control 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 "PackageManager 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 PackageManager Service. 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.

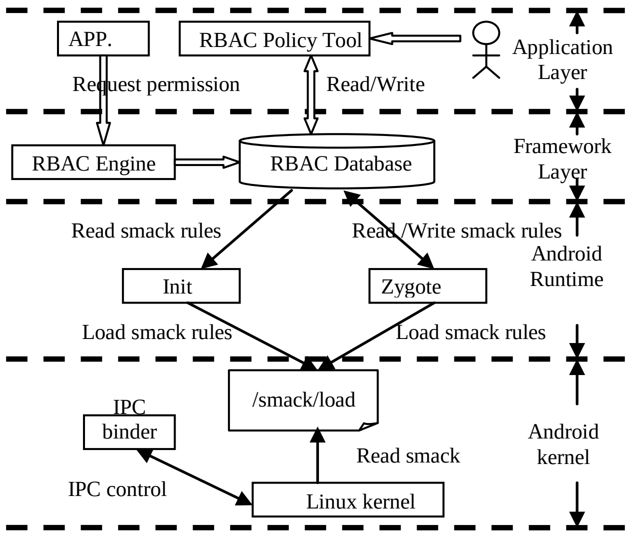


Fig. 1. The architecture of access control system for Android

IV. IMPLEMENTATION CHALLENGES AND SOLUTIONS

When implementing the access control framework, we improved the smack LSM module, and created some specific smack rules for Android system. We encountered several challenges during the improvement as follows.

1. Android does not support smack: LSM and smack are disabled in the Linux kernel that Android uses, to solve this problem, we added some configuration such as "CONFIG_NETLABEL=y", "CONFIG_SECURITY_NETWORK=y", "CONFIG_SECURITY_SMACK= y" and so on in Android4.0 goldfish kernel, and let goldfish kernel support ext4 file system, then we recompiled the goldfish kernel to support Smack.

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 “xattr.h” file and “SYSCALL.txt” file in bionic, then we regenerate these system calls.

3. Android does not have methods or tools for loading smack policy: The most straightforward solution to this problem is to develop an external tool kit to load smack rules and label smack labels for tasks and files. Consequently, we designed and 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.

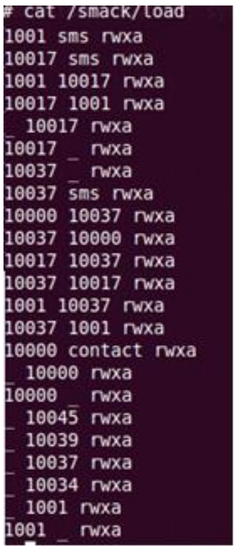
4. It is difficult to create a custom smack policy for Android: Although smack rule is simple, it is not easy to find the subject and object and create smack rules to provide integrity and confidentiality protection for Android. We studied Android IPC binder and found that many important services such 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 code[9] and modified the code of binder driver in the kernel.

```
/* to check binder transaction between two tasks */
static int smack_binder_transaction(struct
task_struct *from, struct task_struct *to) {
    int rc1, rc2;
    rc1 = smk_access(task_security(from),
          task_security(to), MAY_WRITE);
    rc2 = smk_access(task_security(to),
          task_security(from), MAY_WRITE);
    return (rc1+rc2 != 0);
}
```

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.

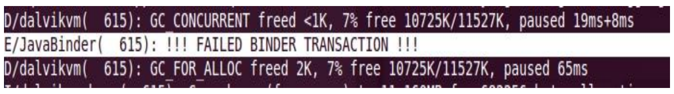
V. System Testing

To test the effectiveness of the access control framework, we selected 360 contact, Baidu contact, Kugou music, DroidDream[11] applications as testing cases. We create “game” role for 360 contact, “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 360 contact, the result showed that SMS sending failed shown as fig.3.



```
# cat /smack/load
1001 sms rwxa
10017 sms rwxa
1001 10017 rwxa
10017 1001 rwxa
_ 10017 rwxa
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
_ 10000 rwxa
10000 _ rwxa
_ 10045 rwxa
_ 10039 rwxa
_ 10037 rwxa
_ 10034 rwxa
_ 1001 rwxa
1001 _ rwxa
```

Fig. 2. smack rules

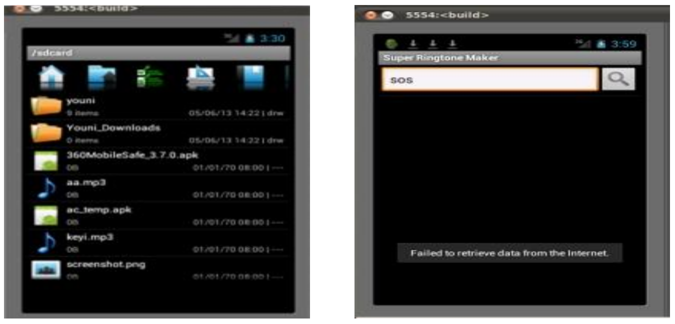


```
D/dalvikvm(  615): GC_CONCURRENT freed <1K, 7% free 10725K/11527K, paused 19ms+8ms
E/JavaBinder(  615): !!! FAILED BINDER TRANSACTION !!!
D/dalvikvm(  615): GC_FOR_ALLOC freed 2K, 7% free 10725K/11527K, paused 65ms
```

Fig. 3. Testing 360 contact application

360 contact process whose uid is 10034 really exchanged IPC binder data with “1001” radio, the hook in the binder driver stopped 360 contact from sending SMS because of a rule “10034 1001 -”.

Fig.4 shows, Advanced File Manager and Super Ringtone Marker which are two variants of DroidDream malware cannot access SDcard or network.



Failed to retrieve data from the Internet.

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

To test the performance of the access control framework, we modified LmBench3 source codes and cross-compiled it so that it can run in Android4.0, then ran LmBench3 independently 10 times to compare the performance overhead without and with our framework on Android system. The comparison results shown in Table 1 show that our framework implementation causes little performance loss.

Table 1. Performance comparison

Measure	**Android**	**Android With Our Framework**	**Slowdown**
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

In the recent years, several security extensions have been developed that aim to enhance Android security[13] in different ways. CRePE[4] 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[5] 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[6] 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[16] 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[7,18] 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

In this paper, we present an access control framework for Android which consists of mandatory access control in Android kernel layer and role-based access control in Android framework layer. Its design and implementation is detailed. Its function and performance test results show that the framework provides fine-grained mandatory access control which can defend malwares, adds little overload to Android system. Our system allows users to specify role-based security policy to restrict the usage of sensitive resources requested by Android applications. Our access control framework can provide users the integrity and confidentiality protection which are useful for related effort in Android security area.

REFERENCES

[1] http://www.networkworld.com/news/2013/051413-android-threats-growing-in-number-269748.html.

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

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

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

[5] 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.

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

[7] NSA, http://seLinuxproject.org/page/SEAndroid.

[8] Marshall D. Abrams, Leonard J. LaPadula, Kenneth W. Eggers, Ingrid M. Olson., "A generalized framework for access control, An informal description". In Proceedings of the 13th National Computer Security Conference, pp. 135–143, October. 1990.

[9] The Smack Project，http://schaufler-ca.com/

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

[11] DroidDream, virus analysis, http://virus.netqin.com/Android/a.spread.DroidDream.z/

[12] 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.

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

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

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

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

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

[18] 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.