Overview of Security Enhanced Android's Security Architecture

Chaowen Zheng
Information Engineering University
Computer Science and Technology
Zhengzhou, China
cvhjo@163.com

***Abstract*—Security Enhanced Android is the integration of Android with SE Linux launched by NSA to strengthen security. This system adopting Mandatory Access Control prevents attacks and enforces application isolation. Also, it provides an implementation of SE Linux in current Android environment. Therefore the capabilities of defeating root overflow attacks and deficiency in applications are significantly strengthened. In this paper, we discussed the security mechanisms of SE Android, and introduced the difficulties and solutions about implementing the system from the kernel and user space level.**

Keywords-Security Enhanced Android; security mechanism; Mandatory Access Control

I. INTRODUCTION

Based on Linux and mainly used in smart devices, Android is the most popular open source operating system, resulting in the great concern about the system's security. Access control, malicious code and kernel vulnerabilities motivate the need to strengthen security mechanisms in the kernel as well as the user space.

DAC (Discretionary Access Control) is used by Android to restrict access between apps. Ordinary apps can only access system resources through system services. Moreover, DAC isolates apps based on users and groups. Each app is assigned a unique (user id, group id) pair when installed, which is used by its all processes and private data files. Only data owners and app creators have access to the data. But DAC has many flaws: (1) cannot prevent vulnerabilities, (2) leaded data by malicious apps, (3) coarse privileges management, (4) cannot limit or control system daemons, and (5) cannot prevent malicious behavior with root privileges.

SE Android (Security Enhanced Android), which was originally launched in 2012 by the NSA based on SE Linux (Security Enhanced Linux), is aimed at full compatibility with the integration of SE Linux and Android.

SE Linux imported MAC (Mandatory Access Control) to overcome the defect of DAC. MAC provides a more in-depth authority management, limiting any process including the root process for all resources access and preventing privilege escalation. The entire security architecture of SE Linux is called Flask. Flask's security policy has four child policies: Multilevel Security Policy, Type Enforcement Policy, and Role-based Access Control Policy. Security access must meet the requirements of each child policy. The logic and common interfaces of security policy are packaged in separate components in the operating system. And the common interfaces are used to obtain security policy decisions.

SE Linux has three running models: (1) Disabled, namely disable SE Linux, (2) Permissive, namely enable SE Linux and record violations without any policy to intercept, and (3) Enforcing, namely interception model.

From Android 4.3, SE Linux was officially introduced into Android, running in the permissive mode. The implementation details are invisible to app developers, and users can choose whether to enable or not.

II. NEW SECURITY ARCHITECTURE

A. Android's Security Architecture

Firstly, rights separation. Android requires the application to provide the user ID and group ID to be identified, and one application has no valid access to another. Thus, after the attack on the application, attackers cannot easily jump to another component.

Secondly, privileges assignment. Apps must be assigned corresponding privileges to access resources when designed. When users check privileges when installed, it starts to function. Specially, it fails when one app wants to run with root authority on Non-Jail broken device.

Thirdly, applications code signing. All apps must be signed in order to run, whether the certificate is authoritative or self-signed.

B. Optimized SE Android's Security Architecture

SE Android imported the MAC mechanism and added some new components on the existing Android architecture without changing original features and attributes. At the same time, SE Android ensured independent operation of each app. Currently, SE Android code is open-source. Users have to recompile the code to use it.

The central idea of SE Android is that even root permissions are usurped, malicious behaviors also can be prevented. For example, once the device was jail-broken, though the *su* privilege can be achieved, administrators could also set policies about modifying system files limiting access.

SE Android offered a range of security control mechanisms, which added different types of mandatory restrictions on Android permissions model. Currently these policies include Install-MAC, Intent MAC, Content Provider MAC, Permission Cancellation and Permission Tab Propagation.

Firstly, Install-MAC. It means that checking the app's claimed permissions through *mac_permission.xml* file when installed. Its actions contain allow, deny and allow all permission. The policy defined the *package* label and *signature* label to specify the security context of apps. But it only functions on pre-installed applications. All third-party apps have to be matched by another label-*default*.

Also, the policy can check the list of permissions one app requests. Once the request corrupted with the policy, the app would not be able to be installed. Even if the app had been installed, if the updated policy conflicted, it also could not run again. If some permissions weren't be explicitly declared as *allow* by the *package* label and the *signature* label or there existed some unpermitted rights, the pre-installed apps would update failed; if some permissions were be declared as *deny* by the *default* label, the third-party app would be also failed to update and run.

Secondly, intent MAC. Its role is to determine whether one intent can be distributed to other components. It will block out any disallowed distributions. Now it supports control of use, read, read and write permissions.

Thirdly, content provider MAC. Its role is to determine whether a request to access Content Provider is allowed. It will block out all disallowed requests, supporting control of *use, read, read and write* permissions currently.

Fourthly, permission cancellation. It can cancel the app's some privileges through authority revocation list when checking applications' privileges. The list lies in *external/mac_policy/revoke_permission.xml*, and it is for each app package to maintenance. The xml file defining revocatory permissions takes effects when system starts.

Lastly, permission tabs propagation. The policy is the application of stain tracking methods. Android's permissions are mapped to one policy configuration file as abstract labels. Not entirely similar to install-MAC's policy file, the configuration file has many different xml tags. Each app's original labels are set based on the permissions it gets. When communications begin between different components, each app will be "contaminated" because of their own label collections. If in forced mode, communications would undermine the policy rules, blocking the traffic.

C. SE Android's Policy Files

SE Android's security policies were configured by policy files.

TABLE I. POLICY FILES TABLE

file	***notes***
seapp_contacts	Located in external/setpolicy,
	mark app's all process and files
property_contacts	Defines different attributes for every system service
	Defines security associations between attributes to check privileges
mac_permissions.xml	The policy configuration file of install-MAC to define what types of resources one app can access

Developers of SE Android are still studying Android's security, so these policies are always changing. And these policies are first to be loaded in the boot procedure.

III. DIFFICULTIES OF IMPORTING SE LINUX

The difficulties lie on the kernel layer and user space.

Firstly, in the kernel layer, SE Linux requires the file system to support security labelling. However, Android's file system was *yaffs2*, not the mainstream part of Linux kernel. The system didn't support extended attributes. Even the newest version has met it, it still can't mark newly created files automatically. In addition, SE Linux doesn't support some kernel subsystems and drivers of Android. For example, some newly created kernel subsystems are responsible for communications between Android apps and specific attributes. SE Linux hasn't supported control of these subsystems. Thus, SE Linux can't fully control over all communications and interfaces.

Secondly, in the user space layer, though Android's kernel is based on Linux, Android's user space is wholly different from Linux distributions. The work how SE Linux integrated into Linux user space couldn't be quoted. Also, the way by which Android started apps is different form Linux. The *zygote* process loaded the Dalvik virtual machine firstly, then forked a child process for each app upon request and loaded the app's private classes into child processes. But SE Linux normally converts safe environment automatically when the process is in the execution. It can't naturally run apps. Still more, some share supports appear at the middleware level. But the communication can only been seen on the kernel level not the middleware level.

Lastly, in the policy configuration, the reference strategy of SE Linux is based on the user space and method of Linux distribution. But Android has unique user space and software stack, and its file system layout and model aren't compatible from SE Linux. SE Linux's strategy is too complex to be suitable for smart devices. To make things worse, SE Linux's strategy requires end users and publishers to configure and communicate. It wasn't feasible on Android.

IV. SOLUTIONS TO DIFFICULTIES

In the kernel layer, Android had to add support for SE Linux kernel, building Android-<board>-<version> branch for various versions of different hardware platforms. In the user space, it also had to build branches for different programs to support apps. What's more, Android imported some code libraries and tools.

A. Kernel Changes

Firstly, basic supports added for SE Linux and its file system. Modify *yaffs2* file system's *getattr* file, which is header files listing extended attributes' values. Modify *yaffs2* to make newly created files be automatically set security tabs.

The new Android devices have used the ext4 file system, supporting extended attributes and security labelling.

Secondly, modify some kernel subsystems, such as *Binder*.

Binder is the most original system of communication between apps. It providers management for multi-process

shared elements and support for transparent call for objects. When system starts, *service manager* program registers itself as *Binder Context Manager* Process via */dev/binder*, providing an interface for apps to call other service requests. Different android framework service register object references through *service manager*. If have received a pointer to an object reference to another process, the app would initiate communication on a given Binder object. The solution is to inset new LSM (Linux Security Module) security hooks into the binder driver. When the binder is running, these hooks are used to check permissions, especially when checking communications between apps.

Anonymous Shared Memory Area is indicated by file descriptors, and it can be used to support SE Linux. The *ashmem_specific_ioctl* command also can be used to view hook security and check permissions.

B. User Space Changes

Firstly, support for the C language libraries and dynamic links. SE Linux uses Linux's system calls for extended attributes to get and set file security tag. So extend the *bionic* file which is implementation of C language library in Android, and repackage the system calls for extended attributes. Then modify Android dynamic linker to identify and use the auxiliary vector *AT_SECURE* to notify the user space whether the security space conversion has occurred.

Secondly, add SE Linux's proprietary libraries and tools. Select the smallest collection from SE Linux API libraries that is suitable for Android. And modify the compiler of *libsepol* library and *libsemanage* library to make it available on the Mac. Then add support for SE Linux on the command *built-in* in the *init* program, and add SE Linux's tools in the Android toolbox.

Thirdly, file labelling. *Mkyaffs2image and make_ext4fs* tools are responsible for generating Android file system image. But they didn't support file security labelling. So modify them to mark files when establishing the Android file system and remark installed apps. Also, there being no need to remark files, extend recovery and update program to ensure there still exists file labelling.

Fourth, modify initialization process. Extend the *init* program to load SE Linux's strategies prior to other processes. And extend the *uevented* program to mark the device node. Also set the security context of early-init section in the file *init.rc*. Finally mark some special system services.

V. EFFECTS OF SE ANDROID IN SECURITY

A. Prevention of root attacks

SE Android can prevent some root attacks like GinderBreak and Exploid. For such attempt to exploit vulnerabilities in net link socket, SE Android can suspend their attacks.

B. Prevention of app attacks

- Skype. The app's service is mainly realized by VOIP. However, Skype didn't encrypt sensitive user data in data directories. The information included the user' data of birth, home address, contacts, chatting logs and so on. In DAC, the file permissions were controlled by each app. But in SE Android, the policy is controlled by policy editors, and act on all apps. Through assign files unique MLS level, SE Android ensures that no other app can read or write files.
- Opera Mobile. The cache files of the app could be read and written. As like Skype, SE Android can prevent illegal access.

VI. CONCLUSION

As Android is developing more and more popular not only in phones but also in smart devices like wearable devices, protecting the system's security becomes more and more important. This paper introduces a new idea about security mechanism inspired by SE Linux, thus making it more concrete and feasible. From the kernel layer and the user space, the paper discusses the changes, difficulties and solutions to import SE Linux into existing Android architecture. Finally SE Android takes effects in preventing some attacks. Now SE Android has been officially imported into Android devices, we will continue with the study of seamless integrating SE Android and quest for underlying security of Android in order to protect the system without sacrificing some convenience.

REFERENCES

[1] Stephen Smalley, Robert Craig: Security Enhanced (SE) Android: Bringing Flexible MAC to Android. In: Proceeding of 19th Network and Distributed System Security Symposium, San Diego (Feb 2012)

[2] Sheran Gunasskera: Android Apps Security. Publishing House of Electronics Industry, Beijing (2013)

[3] Wu, Q., Zhao, C.-X., G, Y.: Android Secuirty Mechanism and Application Practices. China Machine Press, Beijing (2013)

[4] William Enck, Damien Octeau, Patrick McDaniel, and Swarat Chaudhuri: A Study of Android Application Security. In: Proceeding of 20th USENIX Security Symposium, San Francisco (August 2012)

[5] Paul Pearce, Adrienne Porter Felt, Gabriel Nunez, David Wagner: AdDroid Privilege Separation for Applications and Advertisers in Android. In: Proceeding of 7th ACM Symposium on Information, Computer and Communication Security (May 2012)

[6] "what is SE for Android", [Online] Available: http://selinuxproject.org/page/SEforAndroid (Feb 2, 2014)

[7] "Validating Security-Enhanced Linux in Android", [Online] Available: http://source.android.com/devices/tech/security/se-linux.html#introduction (Feb 3, 2014)