Research on SEAndroid Security Strategy

Xiang Wang[a]

Information and Educational Technology Center, Southwest Minzu University, Chengdu 610041, China.

[a]wangxiang@swun.edu.cn

Keywords: SEAndroid, Security mechanism, Anomaly detection, Access control, Security policy.

Abstract: As an important part of Android system security mechanism, SEAndroid is directly related to system security. Due to its open source, programmable software framework, and the nature of networked devices, Android is vulnerable to smartphone viruses. Android devices are tightly protected under normal conditions, but an attacker is likely to find weaknesses in a kernel module or core library to gain maximum access and attack. This paper starts with the system architecture of Android, analyzes the existing security mechanism and security risks of Android, and finally gives a security solution for security risks.

1. Introduction

The Android operating system is an open source mobile operating system based on the Linux kernel. It is continuously led and developed by Google's Open Handheld Alliance (OHA). Android is now the world's largest operating system. The original Android relied on its Linux-based autonomous access control (DAC) mechanism to provide security boundaries. The core view of autonomous access control is based on the concept of user ID and group ID. Users are relatively isolated. Files and programs have their own owners, that is, users. Users can only obtain the corresponding authorization. Perform related operations and communication on resource files or processes owned by other users. The same group ID is a combination of users with related attributes, and specifies that the corresponding group of users have the right to operate the corresponding resources. However, the autonomous access control mechanism has significant shortcomings, such as flawed or malicious applications that can leak sensitive data, and cannot restrict any system daemons or segued programs that run with root privileges. The SEAndroid security policy directly determines the security status of the system. Improper configuration will bring serious security problems. For example, if the application is incorrectly over-granted with unnecessary access rights, it will lead to privilege promotion (CVE-2015-4640, CVE-2015- 4641) [1-2]. Therefore, it is necessary to analyze and detect security vulnerabilities in the SEAndroid security policy.

2. Introduction to Android

2.1. Android platform framework

Android is the environment in which applications are executed on mobile devices. The Android platform framework consists of five parts: The Android software stack is based on the Linux 2.6 kernel, which provides core system services such as security, driver, memory management, process management, and network protocol stack. The Linux kernel above is the Android native library, which is a C/C++ library that is called by various system components at the top. These libraries are merged through the Java Native Call (JNI) implementation within the Android application. The Android runtime environment includes the Dalvik virtual machine and core library. Dalvik runs. dex files, a file that is considered to be more concise and memory-saving than Java class files. The core library is written in the Java language and provides a number of subclasses of the Java 5 SE package and some Android-specific libraries. The application framework is still written in the Java language, which is the basis for developers to develop Android. This layer is mainly composed of components directly called by the developer, such as View, Notification Manager, and Activity Manager. An application is also a program written in the Java language and running on a virtual machine. The entire Android platform framework is shown in Figure 1.

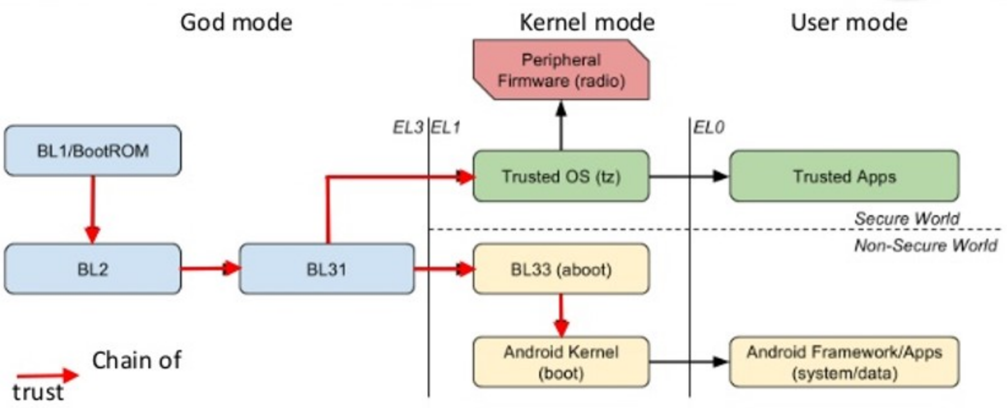


Figure 1. Android platform framework

2.2. Android application composition

The regular publishing format for Android applications is a digitally signed. apk file package, similar to the standard Java Jar, which contains all the code and non-code files for the program. One of the XML files, the Android Manifest file, contains basic information about the application, such as package name, component description, and permission declaration. An Android application consists of four types of components (sub-building blocks): activity, service, content provider, and broadcast receiver. The Activity works in the foreground of the mobile phone screen to communicate with the user, and the service works without a user interface in the background. The content provider provides data storage for applications, and the broadcast receiver helps program components communicate with each other. Each component is independently instantiated and executed while also interacting with other components, if necessary, by other programs.

3. SEAndroid

SEAndroid access control includes three access control models: type enforcement, role access control [3], and multi-layer security [4]. In SEAndroid, each subject and object have a security context, and the format is described as follows: user: role: type: security _ level. Among them, user represents SEAndroid user, role is for RBAC, type is for TE, and security _ level is for MLS. Since in SEAndroid, the system only defines one user u and two RBAC roles r and object _ r, TE is the most important access control model in SEAndroid. This paper also uses the TE model part in SEAndroid as our main research object.

In TE, each subject (process) and object (such as file, TCP socket) has a type, and the type of the subject is also a domain. Types that have commonality will be grouped together to form a collection called attributes, such as the appdomain attribute, which is a collection of application-related domains. Objects can be divided into different categories according to their nature. Common categories include files, directories, sockets, and so on. The system defines a set of permission sets for each category [5].

TE uses the allow rule to control the subject's access to the object, the type _ transition rule to manage the domain transition and assign the type to the newly generated object. The allow rule format is: allow domain type: class {permission sets}. The format of the type _ transition rule is: type _ transition source _ type target _ type: class default _ type.

When the Android system starts, all the policy configuration files are packaged and compiled into a binary file sepolicy, and the system loads sepolicy into the Linux security module of the kernel space. The LSM contains an access vector cache and a secure server. When a process subject accesses the object with some kind of permission, SEAndroid first searches for the access vector in

AVC according to the domain and object type of the process. If it exists, it allows access, otherwise it will continue to search in the security server. If the access vector is present in the secure server, access is allowed and the access vector is written to AVC, otherwise access will be disabled.

4. Android system security threats

Although the Android system has a strong security mechanism, this is not absolutely secure, because the source code of the Android system is free and open, and anyone can get it. This allows the attacker to modify the source code or use the source code. The vulnerability in the system maliciously attacks the system. Here are a few examples of security threats for Android systems.

4.1. Android malware intrusion

The invasion of malware is the main security threat facing Android. The main types of Android malware are the following: Rom built-in malware, deducted malware, stealing privacy malware and tariff traffic consumption malware. These types of malware are generally implemented by making calls, sending short messages, sending multimedia messages, connecting to the Internet, Wi-Fi transmission, Bluetooth transmission, and the like.

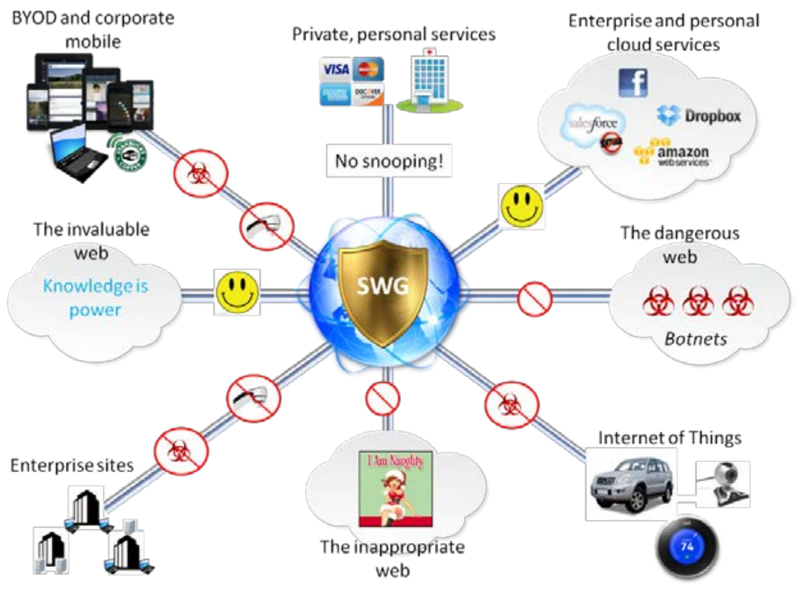


Figure 2. Android security threat

4.2. Illegal access to the root permissions of the Android system

The root privilege is the highest admin right of the Android system. It is easy to delete or change the parts of the system by obtaining the root privilege. Therefore, the attacker will use the flash or software vulnerability to obtain the root privileges of the Android system to modify any file and data of the system, which is also a major security threat facing the Android system.

4.3. Users lack security awareness

(1) Root mobile phone [6]: Root mobile phone refers to the user obtaining root permission through third-party software or flashing machine during the use of the mobile phone. Usually, for security reasons, the manufacturer generates a mobile phone that does not have root privileges after leaving the factory. Therefore, users will have many restrictions when using it, so many users will choose the active root mobile phone. The root mobile phone can be easily installed and installed. Uninstall any mobile app, which is convenient for users to use, but the root phone will bring security risks, such as system instability, virus intrusion, and privacy data exposure.

(2) App download channel confusion: Unlike Apple's only app store, Android's third-party app store is arrogant, and compared to Google's official app store Google Play, third-party store

management is more confusing and faces bigger Security risks. Most users choose to download programs in these third-party app stores, which undoubtedly increases security risks.

5. SEAndroid security policy analysis

In this part, we will introduce the specific design implementation of the system's fact collector, graph generator and path analyzer separately.

5.1. Fact collector

The fact collector collects system state information and SEAndroid security policy configuration information and declares it as a Prolog fact. The SEAndroid security policy configuration information includes a security context and a security policy. In the SEAndroid security mechanism, the subject is generally a process, and the object is generally a file. So, we mainly collect the security context of files and processes. SEAndroid's security policy is mainly stored in a file suffixed with te, and all te files will eventually be packaged into a binary file, sepolicy. We use SE Tools to parse sepolicy into a text file. In all security policies, we focus on the access vector with the rule name allow and the type _ transition type rule.

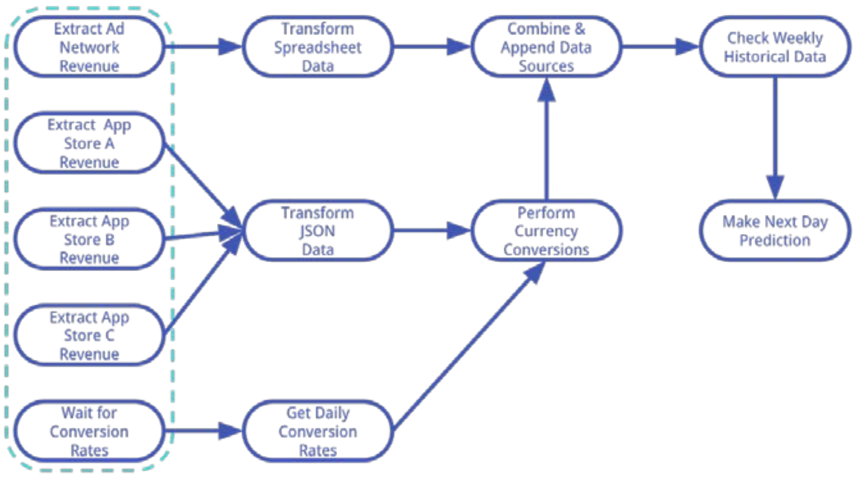
Table.1. System fact statement format

Prolog facts	Meaning
File _ info (Path, Type, Owner, Group, Uper, Gper, Oper, Setuid, Setgid, Sticky, Se _ user, Se _ role, Se _ type)	File information
User _ info (Username, Uid, Gids)	User Info
Process _ network (Pid, Protocol, Port)	Network port information
Process _ running (Pid, Uid, Gid, Program, Se _ user, Se _ role, Domain)	Running process information
Process _ reading (Pid, Filename)	Process open file
Se _ domain (Domain)	Domain information
Se _ type (Se _ type)	Type information
Dom _ priv (Domain, Se _ type, Class, OpList)	Permission information
Se _ typetrans (Old _ dom, New _ dom, Se _ type)	Type conversion
Se _ attribute (Se _ type, Attribute)	Attribute information

5.2. Graph Generator

The capability dependency graph generator generates a capability dependency graph by inputting system facts, access control mechanisms, and derivation rules. Definition: Capability dependency graph: A capability dependency graph is a directed graph $G=(C_a \cup C_o, A, E)$, where C_a is a set of capability nodes, C_o is a set of conditional nodes, A is a set of action nodes, and E is a set of directed edges, $E \subseteq ((C_a \cup C_o) \times A) \cup (A \times C_a)$.

There are two types of directed edges of a capability-dependent graph: the edge pointing to the action node and the edge pointing to the capability node. A directed edge can point to an action node from a capability node and a conditional node, indicating that the action can only be performed if these capabilities and conditional preconditions are met. The directed edge can also point from the action node to the capability node, indicating the new capabilities that are generated after the action is executed.



Extract Ad Network Revenue
Extract App Store A Revenue
Extract App Store B Revenue
Extract App Store C Revenue
Wait for Conversion Rates
Transform Spreadsheet Data
Transform JSON Data
Get Daily Conversion Rates
Combine & Append Data Sources
Perform Currency Conversions
Check Weekly Historical Data
Make Next Day Prediction

Figure 3. Example of a capability dependency graph

Dac _ can _ access (Uid, Gid, FileName, Mode): Check whether the user id with the group id of Uid or Gid under the autonomous access control can perform the Mode operation on the FileName object.

Dac _ execv (OUid, OGid, NUid, NGid, Program): Check whether the user id with the group id of OUid and OGid under the autonomous access control can clone a user id and the group id is NUid and NGid by executing the Progrom program. New process

Se _ domain _ privilege (domain (Domain), type (Type), class (Class), op (Op)): Check whether the subject whose domain is Domain under the type mandatory access control can have an object of type and class of Class. Perform an Op operation;

Se _ can _ access (Domain, FileName, Mode): Check whether the principal whose domain is Domain under the type mandatory access control can perform Mode operation on the file FileName;

Se _ execv (Domain, New Domain, Type): Check whether the principal whose domain is Domain under the type mandatory access control can clone the new process with the domain New Domain by executing the object of type;

Privilege _ enhancing (Uid, Gid, Domain, NewUid, NewGid, New Domain): Checks whether the attacker's ability has improved after a new process is generated. The basis for the check: 1) The original Uid, Gid is not 0 (root) or 1000 (system) and the user id and group id are changed; 2) the original domain attribute is not unconfined domain and the domain changes.

5.3. Path Analyzer

The Path Analyzer uses the capability dependency graph as input and uses the depth-first search algorithm (DFS) to search all attack paths from the initial capability node to the target node. The algorithm first traverses the initial set of capability nodes, uses each initial capability node vi as an entry parameter, invokes DFS, and combines the obtained path set Paths with the attack path set APS to obtain a new APS. In the DFS search process, we have made some restrictions to reduce unnecessary actions: (1) There is no loop on the attack path; (2) The length of the attack path does not exceed 10; (3) The intermediate node of the attack path does not contain the target node. We chose 10 as the threshold for the length of the attack path because, based on our experiments, we found that most of the attack path lengths in the Android system are 7 and 9. If the threshold is reduced, a large number of attack paths will be lost, resulting in a higher false negative rate; After increasing the threshold. The number of newly added attack paths is only a few dozen, and these new long attack paths are much more laborious for the attacker than the large number of short attack paths. The intermediate steps are more likely to fail and are easily discovered by the audit system. Not attractive to attackers. The choice of 10 as the threshold of the attack path length can meet the performance requirements and the attack surface that the system bears.

5.4. Anomaly Detection

The intrusion detection system framework in [7] continuously samples a variety of system parameter indicators, using machine learning methods to infer the state of the device, such as whether the collected data is normal (benign) or abnormal (malicious)). The main idea of this method is to generate system parameter metrics (such as CPU usage, number of packets sent via Wi-Fi, number of processes running, power consumption, etc.), triggered by known malware. The system parameter indicators are compared to detect the same point, and then discover new malware that has not been encountered before. Because of the lack of ready-to-use Android malware, we first developed four malwares and then evaluated the ability to detect new malware based on known malware sample detection. We evaluated several combined experiments, including different classification algorithms and anomaly detection algorithms; different feature selection methods; different top features were selected. The purpose of the research is to recognize the detection algorithm, the feature selection method, and how the highest number of features is selected to distinguish other benign software and malware that are not included in the training group. When training and testing are performed on different devices, specific features that produce the greatest detection accuracy should be found. The empirical results show that this proposed framework is usually effective in detecting malware, especially on Android (accuracy rate 87.4%, false positive rate 0.126).

6. Conclusion

The Android system is currently the most widely used intelligent terminal operating system, and its biggest feature is openness. Openness requires that Android must have a robust security mechanism. Android system layered security mechanism design runs through all levels of Android system architecture, but there are still security risks. Only Android system continuously improves its security, users improve security awareness in the process of using intelligent terminals, and constantly improve Android malware detection. The technology is three-pronged, and it is believed that the Android system will become more and more secure while bringing convenience to people.

Acknowledgments

This work was supported by Central University Basic Research Business Fee Special Fund " BYOD security strategy research " Project Number is 2017NZYQN33.

Xiang Wang (1978-), Male, Master, Chengdu, sichuan. Information and Educational Technology Center Southwest Minzu University, Engineer, Engaged in the study of network operation and management.

References

[1] Yang Zhonghuang, Liang Shanqiang, Zhan Weixiao. Remote Management of Mobile Devices Based on SEAndroid. Journal of Xi'an University of Posts and Telecommunications, Vol. 3 (2018) No.23, p. 17-24.

[2] Huang Jing, Zhao Haiyan, Sun Lingling. Research and Design of Android Terminal System Security Mechanism. Telecommunications Engineering Technology and Standardization, Vol. 10 (2017) No.15, p. 103-105.

[3] Wen Hanxiang, Li Yujun, Hou Mengshu. Research on Privacy Protection Mechanism Based on SEAndroid. Computer Science, Vol. 2 (2015) No.17, p. 329-332.

[4] Zhang Weidong, Yang Xiuzhi. Design of Data Security Protection Scheme for Set Top Box Based on Android Platform. Cable TV Technology, Vol. 4 (2017) No.32, p. 85-89.

[5] Yang Zhonghuang, Dong Lunming. Improving the Security of Mobile Systems Based on SEAndroid Combined with Remote Control. Journal of Xi'an University of Posts and Telecommunications, Vol. 5 (2016) No.19, p. 1-10.

[6] Xiao Chengwang, Lu Jun, Yu Ligeng. Design and Verification of New Preventive Model for Android Mobile Rights Raising. Computer and Modernization, Vol. 9 (2017) No.14, p. 56-60.

[7] Song Xinlong, Zheng Dong, Yang Zhonghuang. Mobile Device Management System Based on AOSP and SELinux. Information Network Security, Vol. 9 (2017) No.35, p. 103-106.