International Journal of Research in Advent Technology (E-ISSN: 2321-9637) Special Issue
National Conference "NCPCI-2016", 19 March 2016
Available online at www.ijrat.org

Survey on SELinux in Android

Prof. S.S.Sambare, ssambare69@gmail.com
Ayushi Agarwal, ayushi3a@gmail.com
Sneha Bhambhani, sabhambhani@gmail.com
Ritesh Tejwani , tejwanii.riteshh@gmail.com
Prerana Rai, preranarai0409@gmail.com
Dept of Comp Engg,
Pimpri Chinchwad College of Engineering, Pune.

Abstract: With the increase in technology, the current use of mobile phones is increasing with a rigorous amount and so we need to assure that the information stored in our cell phones is secure and is not being misused. The apps when installed in Android do not provide high level security to the information present in our cell phones and thus the implementation of SELinux helps in securing the information more effectively. Android being a Linux based system can support SELinux and thus provide users with a robust Mandatory Access Control (MAC) to ensure full system security. It not only provides flexible security but also helps in reduction of performance overhead only by implementing a limited chip area.

Keywords: - ***Android, SELinux, Security, Mobile devices, MAC.***

1.0 Android Introduction

Android is the most popular mobile operating systems. It was released by Google in 2008.It is an open source operating system, primarily based on the Linux Kernel.

Android applications are written in Java and run on virtual machines. The open nature of Android attracts a variety of third-party application marketplaces. These, either provide an alternative for the devices that are not allowed to ship with Google Play Store, or provide applications that cannot be offered on the Google Play Store due to policy violations, or for some other reason.

Android is mainly designed for use in smart phones, tablets etc. Recently, we also have AndroidTV for televisions, AndroidAuto in cars, AndroidWear for watches and many more.

Malware attacks propagating into smart phones include cellular networks, Bluetooth, the Internet, USB, and other peripherals. Security mechanisms such as anti-malware and anti-spam software, host-based intrusion detection tools, and firewalls are available, but not widely used.

Today, Android has the largest base among all the operating systems.

Hence, security in Android is of immense importance and is required to be very strong.

1.1 ANDROID FRAMEWORK

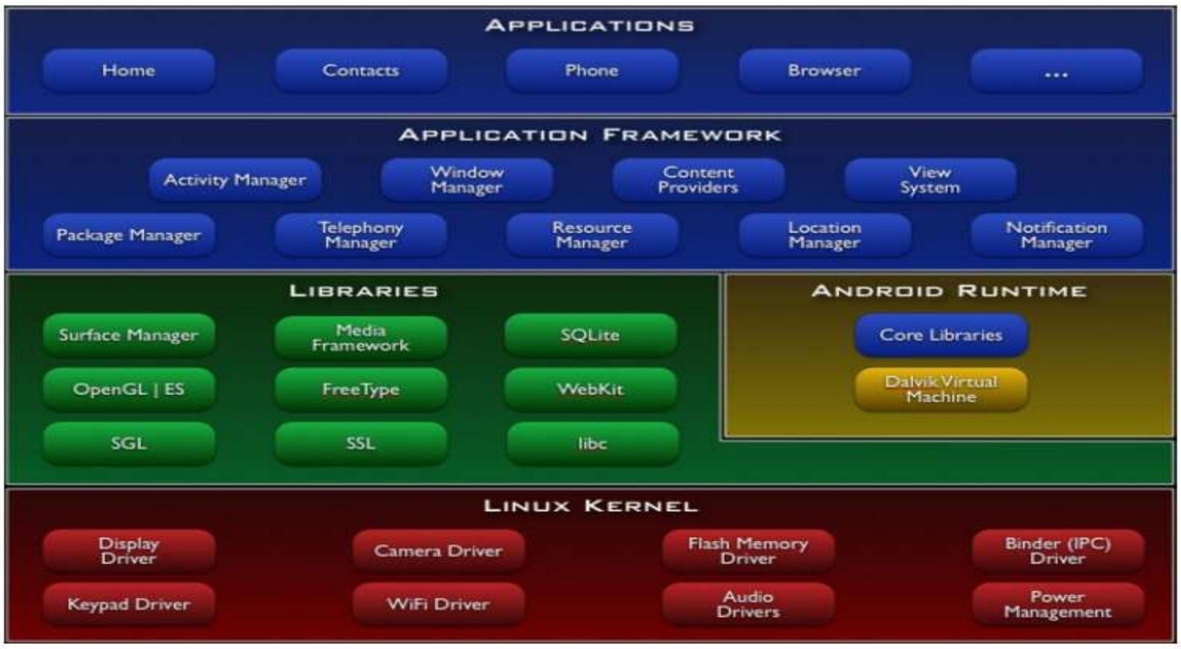

Fig. 1. Android Framework

The Android software stack is divided into five main layers:

1. Linux Kernel.

2. Native Libraries.

3. Android Runtime

4. Application Framework

5. Application.

The Linux kernel is responsible for providing core services to the Android software stack. These services consist of networking, memory management, file system, device drivers, power management etc. The Native Libraries in Android are written in C/C++. Standard libraries such as libc were developed mainly for low memory consumption.

The Android Runtime consists of Java core libraries and Dalvik Virtual machine. It enables every app to run in its own process, with its own instance of the Dalvik virtual machine. [1]

The Application Framework consists of various frameworks written in Java. These provide an abstraction of the underlying native libraries and Dalvik capabilities to applications. It includes tools provided by Google as well as proprietary extensions or services. Each application is packaged into a .apk archive for installation.

2.0 Security in Android

The core of the security at application-level in Android is its permission mechanism. Contradicting a typical Linux-based Personal system, different applications in Android are executed as different users. This preventive measure causes the bar to rise on successful exploitation because one application can't affect others in a normally. However, more processes could run as the same user, and, particularly, several system daemons run as system, radio and root users. The security mechanisms of both Android-specific and Linux inherited are insufficient and too coarse-grained to deal with the security issue. [1]. While installing an application, the application displays a dialog indicating the permissions requested and asks if it should continue the installation. The user can not accept or refuse individual permissions – he must accept or refuse all the requested permissions only as a block. The security model of Android is based on the concept of sandbox. Every application runs in the separate individual sandbox. Beginning with the 4.3 Android version, SELinux further describes the boundary limits of the Android application sandbox. Android uses SELinux as a part of its security model, which enforces MAC(mandatory access control) over other processes, including the processes running with superuser/root privileges. Android security is enhanced by SELinux that confines privileged processes and automating the creation of security policy. Many contributions have been made to it by various organizations. Using SELinux, Android can do better confinement of system services, access control to application and system data and logs, reducing the ill-effects of malicious code, protecting users from flaws in code on mobile and other handheld devices.

2.1 Security using SELinux

Default denial is the ethos on which SELinux works. Anything which is not explicitly allowed is refused. SELinux opeartes in two global modes: permissive mode, in which permission refusals are logged but not forced, and enforcing mode, denials are both logged and enforced in which . SELinux even supports a per-domain mode in which particular processes can also be made permissive while keeping the remaining of the system in global enforcing mode [3]. Per-domain permissive mode enables applications of SELinux to an ever-increasing part of the system. Per-domain permissive mode also enables the development of policy for new services while keeping the rest of the system enforcing. SELinux was introduced to Android and it was evaluated on an HTC G1 device. The experiment indicated that running SELinux on Android enabled devices is comfortable and that an enhanced level of security can be achieved by providing a relevant policy.[2] This security has the property of operating with a regular user without disturbing in any noticeable way. In fact, the user need not be cautious that SELinux has been applied. Applications need to continue functioning on SELinux-enabled devices without any modification.

When deciding upon customization of SELinux, developers should remember to: Give SELinux policy for every new daemon, Use earlier defined domains whenever neccessary, assign a domain to any process spawned as an init service, know the macros before writing policy, make changes to core policy to AOSP And not to.[5]Create inappropriate policy Allow application user policy customization Allow customizing of MDM policy make users aware with policy violations Add backdoors Seeing the Security Features of Kernel section of the Android Compatibility Definition document for specific requirements. SELinux makes use of a white list approach, in which all access must be allowed explicitly in order to be granted. Since Android's default SELinux policy already supports the Android Open Source Project, there is no need of OEM to modify SELinux settings in any way. If they customize SELinux settings, they should take utmost care not to break existing applications. Divide the software components into modules which conduct singular tasks. Creating SELinux policies that isolates the processes from unrelated functions. Puting these policies in *.te files. [4] Within the /device/manufacturer/device-name/sepolicy directory and use BOARD_SEPOLICY variables to include them in your build. In an ideal software development process, SELinux policy changes only, when the software model changes and not the actual implementation. As device developers begin the customization of SELinux, they must priorly patent their additions to Android [4]. If they've featured a component that does a new function, the developers will need to assure that the component satisfies the security policy, as well as any related policy crafted by the OEM, before turning on enforcing mode. This expresses that all application domains are allowed to write and read files labelled as app_data_file. Intents are asynchronous messages that grants permission to application components to request functionalities from other Android components (Fig2). This represent the higher-level Android Interprocess Communication (IPC) technique, and this underlying transport mechanism used is known as binder. This means that a common person developing an application refrains to use this mechanism to prevent its own app from malicious requests by other applications[8].

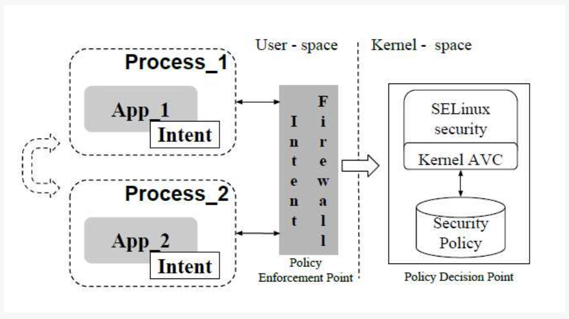

Fig. 2.SELinux-IntentFirewall architecture

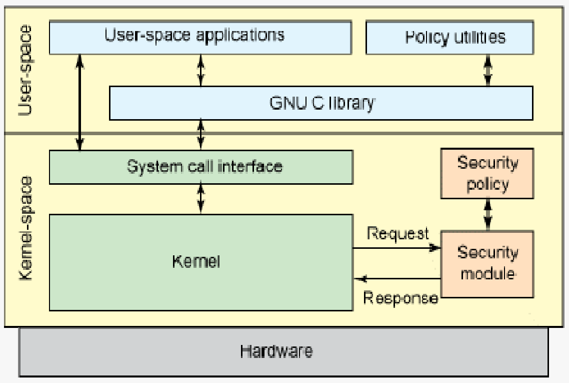

Fig. 3.Integration of Security module(SELinux)

3.0 Comparison between SELinux and AppArmor.

	SELinux	**AppArmor**
Integration	In Android since 4.3	Integrated to Synology's DSM 5.1 Beta in 2014.
Identification	Identifies filesystem objects by inode	Identifies file system by path.
Operation Set	Set of operations are larger.	Set of operations are smaller.
Granularity	Better	Much less.
Multi-threading	Supports implicit use of POSIX capabilities.	No controls for categorically boundary.
Rule Sets	Incredibly complex	AppArmor and Smack are straight forward.
Security	Potentially much secure if profiles are built well.	Lesser secure since identifies entire path.
Isolation	Processes are well isolated.	Lesser control on bounding prevents isolation.
Current Usage	Android 4.3 onwards.	Ubuntu, Fedora, etc.

4.0 Steps in detail

1. Enable SELinux in the kernel: CONFIG_SECURITY_SELINUX=y
2. Change the kernel_cmdline parameter so that: BOARD_KERNEL_CMDLINE := androidboot.selinux=permissive.
3. Boot up the system in permissive and see what denials are encountered on boot: On Ubuntu 14.04 or newer:
4. On Ubuntu 12.04: adb shell su -c dmesg | grep denied | audit2allow
5. Evaluate the output. See Validation for instructions and tools.
6. Identify devices, and other new files that need labeling.
7. Use existing or new labels for your objects. Look at the *_contexts files to see how things were previously labeled and use knowledge of the label
9.
10. help reveal those that remain running (but ALL services need such a treatment): $ adb shell su -c ps -Z | grep init $ adb shell su -c dmesg | grep 'avc: '
11. Review init.<device>.rc to identify any which are without a type. These should be given domains EARLY in order to avoid adding rules to init or otherwise confusing init accesses with ones that are in their own policy.
12. Setup BOARD_CONFIG.mk to use BOARD_SEPOLICY_* variables. See the

This is only for initial development of policy for the device. Once you have an initial bootstrap policy, remove this parameter so that your device is enforcing or it will fail CTS.

adb shell su -c dmesg | grep denied | audit2allow -p out/target/product/*board*/root/sepolicy

meanings to assign a new one. Ideally, this will be an existing label which will fit into policy, but sometimes a new label will be needed, and rules for access to that label will be needed, as well.

8. Identify domains/processes that should have their own security domains. A policy will likely need to be written for each of these from scratch. All services spawned from init, for instance, should have their own. The following commands

README in external/sepolicy for details on setting this up.

13. Examine the init.<device>.rc and fstab.<device> file and make sure every use of "mount" corresponds to a properly labeled filesystem or that a context= mount option is specified.
14. Go through each denial and create SELinux policy to properly handle each. See the examples within Customization [8].
15.

16. CONCLUSION

❑ In comparison of SELinux with other LSMs (like Apparmor, Smack, etc), it is observed that the rule sets are comparatively more complex, but gives more control over process isolation which makes it potentially much secure. The notion of multi-level security and the concept of remote policy server gives SELinux an edge over Apparmor. The integration of SELinux into Android is thus a significant step towards the realization of more robust and flexible security services.

❑

❑ REFERENCES

❑ [1]. Asaf Shabtai, Yuval Fledel and Yuval Elovici at Ben-Gurion University,"Securing Android-Powered Mobile Devices using SELinux", IEEE 2010

❑ [2]. Asaf Shabtai Deutsche Telekom Laboratories at Ben-Gurion University, "Malware Detection on Mobile Devices", Eleventh International Conference on Mobile Data Management, 2010.

❑ [3]. Kashif Ahmad Khan, Muhammad Amin ,"SELinux IN and OUT" – IEEE 2011.

- [4]. Leandro Fiorin, Alberto Ferrante, Konstantinos Padarnitsas, Francesco Regazzoni in Switzerland,"IEEE_2012 Security Enhanced Linux on Embedded Systems ".ICSAI_2012 Analysis For SELinux Security Policy
- [5]. Prof. Dr. Frank Bellosa, Stefan Brahler, "Analysis of the Android-Architecture ", IEEE 2015.
- [6]. B. Vogel and B. Steinke, "Using SELinux Security Enforcement in Linux-Based Embedded Devices, Proc. 1st Int'l Conf. Mobile Wireless Middleware, Operating Systems, and Applications (MobilWare 08)
- [7] https://source.android.com/security/selinux
- [8]. Simone Mutti, Enrico Bacis, "An SELinux-based Intent manager for Android ",
- IEEE CNS 2015 Poster Session.